Interactive visualization of volumetric data with WebGL in real-time

John Congote*
EAFIT University
CAD-CAM-CAE Laboratory
Medellín, Colombia

Luis Kabongo†
Vicomtech Research Center
Donostia - San Sebastian, Spain

Aitor Moreno‡
Vicomtech Research Center
Donostia - San Sebastian, Spain

Alvaro Segura§
Vicomtech Research Cente
Donostia - San Sebastian, Spain

Jorge Posada¶
Vicomtech Research Center
Donostia - San Sebastian, Spain

Oscar Ruiz‖
EAFIT University
CAD-CAM-CAE Laboratory
Medellín, Colombia

Abstract

This article presents and discusses the implementation of a volume rendering system for the Web, which articulates a large portion of the rendering task in the client machine. By placing the rendering emphasis in the local client, our system takes advantage of its power, while at the same time eliminates processing from unreliable bottlenecks (e.g. network). The system developed articulates in efficient manner the capabilities of the recently released WebGL standard, which makes available the accelerated graphic pipeline (formerly unusable). The dependency on specially customized hardware is eliminated, and yet efficient rendering rates are achieved. The Web increasingly competes against desktop applications in many scenarios, but the graphical demands of some of the applications (e.g. interactive scientific visualization by volume rendering), have impeded their successful settlement in Web scenarios. Performance, scalability, accuracy, security are some of the many challenges that must be solved before visual Web applications popularize. In this publication we discuss both performance and scalability of the volume rendering by WebGL ray-casting in two different but challenging application domains: medical imaging and radar meteorology.

Keywords: Volume Rendering, Ray Casting, Real-Time visualization, WebGL, Weather Radar Volume, Medical Imaging

1 Introduction

Real-time 3D computer graphics systems usually handle surface description models (i.e. B-Rep representations) and use surface rendering techniques for visualization. Common 3D model formats such as VRML, X3D, COLLADA, U3D (some intended for the Web) are based entirely on polygonal meshes or higher order surfaces. Real-time rendering of polygon models is straightforward and raster render algorithms are implemented in most graphics accelerating hardware. For many years, several rendering engines, often via installable browser plug-ins, have been available to support 3D mesh visualization in Web applications.

However, some scientific fields (e.g. medicine, geo-sciences, meteorology, engineering) work with 3D volumetric datasets. Volumetric datasets are irregular or irregular samples of either scalar ($f : \mathbb{R}^3 \rightarrow \mathbb{R}$) or vector ($f : \mathbb{R}^3 \rightarrow \mathbb{R}^3$) fields. For the purpose of this article, we will use the term *volumetric data sets* to refer to scalar fields and will ignore for the time being vector fields. Surface-based raster rendering techniques are obviously not suitable for visualizing such datasets and specific Volume-Rendering algorithms are needed, which are not available for the Web. Therefore, our work uses *Volume ray-casting*, which is a common technique in Computer Graphics for volume visualization originally presented in [Levoy 1988] and further studied in [Hadwiger et al. 2009]. The rendering is not photo-realistic but shows the important characteristics of the set.

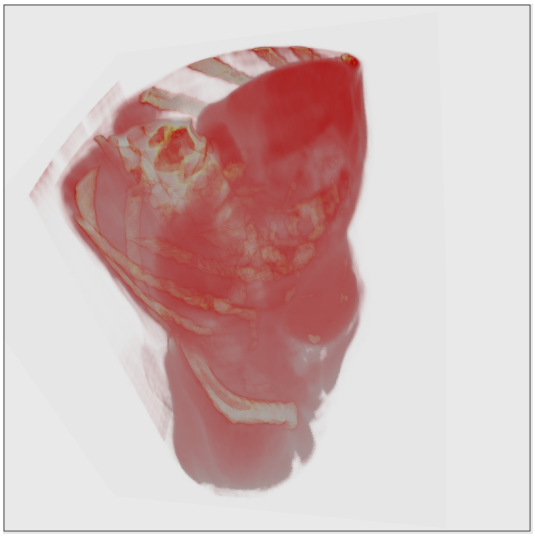


Figure 1: *Medical data rendered with volume ray-casting*

In medical imaging, diagnostic techniques such as computer tomography (CT), magnetic resonance imaging (MRI) and positron emission tomography (PET) produce sets of parallel slices that form a volumetric dataset. Volume rendering is a common technique for visualizing volumetric datasets along with multi-planar reconstructions (MPR). Storage and distribution of these 3D images usually requires a Picture Archiving and Communication Systems (PACS), which normally uses specialized workstation software ([Meyer-Spradow et al. 2009], [web f]) for interactive visualization ([Mahmoudi et al. 2009]). Few solutions exist for regular desktop computers ([Kabongo et al. 2009]). Weather radar data is also a volumetric dataset. A weather radar scans a volume around it by collecting values in families of circular, elliptical or conical scan surfaces. Doppler radars sample several physical variables (reflectivity, differential reflectivity, radial velocity and spectral width) for each location of the sky. Radar data is usually visualized for a single slice of the volume, either conical, as in *plan position indicator* (PPI), or planar, as in *constant altitude plan position indicators* (CAPPI). Initial approaches for Web-based Volume Rendering of radar data rely on pre-computed in-server rendering ([Sundaram et al. 2008]), which normally hinders interaction in the visualiza-

*e-mail:jcongote@eafit.edu.co
†e-mail:lkabongo@vicomtech.org
‡e-mail:amoreno@vicomtech.org
§e-mail:asegura@vicomtech.org
¶e-mail:jposada@vicomtech.org
‖e-mail:oruiz@eafit.edu.co

tion.

WebGL is a new standard for accelerated 3D graphics rendering in the Web that complements other technologies in the future HTML5 standard ([Marrin]). Some of the major Web browsers, including Google Chrome, Mozilla Firefox, WebKit, Safari and Opera have already implemented it in their latest releases or release candidates. WebGL is basically a Javascript binding of the OpenGL ES API and enables low level imperative graphics rendering based on programmable shaders.

1.1 Contribution of this Article.

Our contribution is a practical implementation of a volume rendering system for the Web, based on the Volume Ray-casting algorithm and implemented on WebGL. Our system is capable of obtaining interactive visualization with diverse volume datasets (Fig. 1). The original Volume Ray-casting algorithm was slightly modified to work with the input structures needed for the Web environment. Special care was taken to avoid the use of dynamic server content. This avoidance allows for the algorithm to be used without increasing the demands on the server and shifts, as much as possible, the processing to the client. Our work is tested in two scenarios with volumetric datasets: medical imaging and radar meteorology. Their main practical difference is the pattern in which the volume is sampled: for medical data a uniform cartesian grid is used. For weather data a non-uniform spherical coordinate grid is used. A side - effect of our work is the proof-of-concept that interactive platform - independent visualization of 3D data on the Web is feasible by means of the WebGL standard.

The paper is organized as follows. Section 2 presents a brief status of the different technologies present in this publication: Volume Rendering, Web rendering, medical and radar visualization. Section 3 presents our methodology for volume rendering with special attention to the modifications of the algorithm for the Web environment. Section 4 presents the output obtained by the implemented algorithm and the performance values in different conditions. Section 6 presents the conclusions of our work and future directions.

2 Related Work

2.1 Volume rendering techniques

In 3D scalar field interactive visualization, two solutions prevail: Surface Rendering and Volume Rendering. Surface Rendering has the advantage of being easy to compute due to its low geometric complexity. Its main disadvantages are: (1) A surface must be synthesized first, which is not a trivial task as it depends on the quality of the sample. (2) Since it must be precomputed, the result is static and cannot be easily adjusted in real time.

Recent advances in Volume Rendering and graphic card capabilities allow the representation of volumes with good quality by projecting volumetric data into a 2D image, depending on the position of a virtual camera. The main advantage of this technique is the visualization of all inner characteristics at once.

Preprocessing does not intervene since most of the computations are performed when the camera is displaced. In order to project the volumetric data, several methods exist ([Meißner et al. 2000]). Reference [Westover 1991] discusses Volume Splatting and represents each scalar value by a simple geometrical shape that will face the camera, allowing fast rendering. Its main disadvantage is the loss of quality. A technique called *Shear Warping* ([Lacroute and Levoy 1994]), consists of applying shear warp transformations to the volume slices to imitate the real orientation of the camera. Since the technique is based on simple transformations this method is quite fast but its main drawback is a low sampling power. With the constant improvement in graphic card capabilities, the *Texture Mapping* method has been popularized in video-games. It consists in re-slicing the volume depending on the orientation ot the camera viewpoint and representing all the slices at once taking advantage of eventual occlusion optimizations ([Hibbard and Santek 1989]).

Volume Ray-casting was initially presented in [Levoy 1988] and has become one of the most common methods for volume rendering. The set of rays from the camera reach the 3D scene and hit the objects, generating parametric (scalar) landmark values. By defining a blending function it is possible to give priorities to the different values encountered along the ray, allowing the visualization of different internal structures. Additional modifications to the algorithm, such as *transfer functions*, and *Phong illumination* ([Phong 1975]) were developed in order to improve the perception and make the volume look realistic. Compared to the other techniques, this one is older and more accurate in sampling. However, the computational power required made initially difficult its usage in real-time interactive representations, allowing other approximations to settle. Nowadays, the increasing computational power of graphic cards allows fast calculations ([Kruger and Westermann 2003]) which give new interest to Volume Ray-casting. Reference [Hadwiger et al. 2009] presents a tutorial with all the basic explanation on volume ray-casting. We used this tutorial as starting point for the theoretical foundations in our implementation and for technical details. Open Source implementations such as [Meyer-Spradow et al. 2009] and [web f] were also used.

2.2 Web 3D rendering

The fact that the Web and 3D graphics are currently ubiquitous in desktop and palm devices makes their integration urgent and important. Several standards and proprietary solutions for embedding 3D in the Web have been devised, such as VRML, X3D or vendor-specific Web browser plug-ins, implementations on general purpose plug-ins, etc. A review of these techniques could be found in [Behr et al. 2009].

In the case of Medical Imaging and other computing-intensive visualization scenarios, a partial solution has been the use of in-server rendering ([Blazona and Mihajlovic 2007]). In this approach, the rendering process is remotely performed in the server and its resulting image is sent to the client. This solution increases the load on the server when many clients are present. In addition, the high latency times make the system non - responsive and unsuitable for smooth interactive visualization.

Outstanding issues among solutions for Web 3D graphics are: dedicated languages, plug-in requirements for interpretation, portability across browsers, devices and operating systems and advanced rendering support. While writing this article, the Khronos Group released the WebGL 1.0 specification, which has been under development and testing. In practice, the WebGL 1.0 is a Javascript binding of the OpenGL ES 2.0 API. Calls to the API are relatively simple and serve to set up vertex and index buffers, to change rendering engine state such as active texture units or transform matrices, and to invoke drawing primitives. Most of the computation is performed in vertex and fragment shaders written in GLSL language, which are run natively on the GPU hardware. Unlike previous Web 3D standards which define declarative scene description languages, WebGL is a low-level imperative graphic programming API. Its imperative character enables a great flexibility and exploits the advanced features of modern graphics hardware.

The WebGL 1.0 standard takes advantage of already existing OpenGL-based graphics applications, such as accurate iso-surface computation ([Congote et al. 2010]) or optimized shader programming ([Marques et al. 2009]). The usage of an interpreted language to manage the behavior of scene elements and animations might be considered as a drawback. However, the performance of Javascript interpreters is constantly improving. Current optimized just-in-time compilation in the latest engines provides performance not far from that of natively compiled languages.

2.3 Medical visualization

From the different scientific fields, Medical Visualization is one of the most challenging since the user interpretation directly translates into clinical intervention. Quality is one of the most important factors, but fast interactive response is also central in this domain. Medical Visualization has already produced some implementations of volumetric visualization in Web, mainly for educational purposes ([John et al. 2008][John 2007]). These approximations require third party systems for the correct visualization, or the presence of a rendering server ([Poliakov et al. 2005], [Yoo et al. 2005]), which limits the scalability of the application. Using standards such as VRML and Texture Mapping ([Behr and Alexa 2001]) visualization of volumes in the Web has been achieved.

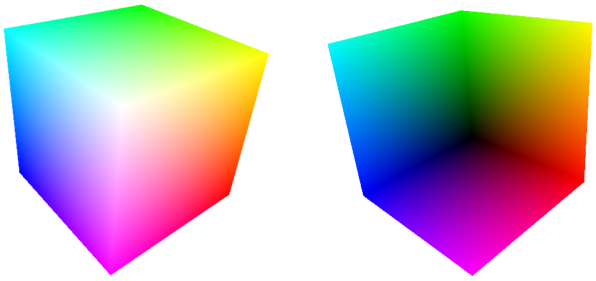
2.4 Radar visualization

Radar data visualization also poses new challenges as the data are acquired in a spherical coordinate system ([Riley et al. 2006]). This particularity makes difficult the optimization of ray-casting, which usually traverses cubic-shaped volumes. Nevertheless, this problem has already been addressed in [Goenetxea et al. 2010].

3 Methodology

Volume Rendering is a set of Computer Graphics algorithms to generate representations of a 3D volumetric dataset. The produced image is a 2-dimensional matrix $I : [1, h] \times [1, w] \rightarrow \mathbb{R}^4$ (w: width and h: height in pixels). A pixel has a color representation expressed by four-tuple (R, G, B, A) of red, green, blue and alpha real-valued components, $(R, G, B, A \in [0, 1])$.

The volume is a 3-dimensional array of real values $V : [1, H] \times [1, W] \times [1, D] \rightarrow [0, 1]$ (H: Height, W: Width, D: Depth of the represented volume, in positive integer coordinates). Therefore, $V(x, y, z) \in [0, 1]$. The volume-rendering algorithm is a projection of a 3D model into a 2D image. The projection model used in this work is known as a pin-hole camera ([Hartley and Zisserman 2003]). The pin-hole camera model uses an intrinsic $K \in M_{3\times4}$ and an extrinsic $R \in M_{4\times4}$ real-valued matrices. These matrices project a 3D point $p \in \mathbb{P}^3$ onto a 2D point $p' \in \mathbb{P}^2$.



(a) Front Faces (b) Back Faces

Figure 2: *Color cube map coordinates*

A volume is normally represented as a set of images. Each image represents a slice of the volume. Usually slices are parallel and evenly - spaced but this is not always the case. For example, volumes can also be sampled in spherical coordinates with the angular interval being variable. Both cases (cartesian and spherical samples) are handled by our algorithm.

Volume ray-casting is an algorithm which defines the color for each pixel (i, j) in the image or projection screen I, calculated in function of the values of a scale field $V(x, y, z)$ associated with the points (x, y, z) visited by a ray originated in such a pixel. The ray is casted into the cuboid that contains the data to display (i.e the scalar field V). The ray is equi - parametrically sampled. For each sampled point p_s on the ray one computes an approximation of the scalar field $V(p_s)$, by usually calculating a tri-linear interpolation. In addition, a shade might be associated to p_s, according to the illumination conditions prevailing in the cuboid. The color associated to p_s might be determined by axis distances as shown in figure 2. As last step, the pixel in the image which originated the ray is given the color determined by the sampled point p_s nearest to the screen, in such a ray.

Alternatively, the samples on the ray may also cast a vote regarding the color that their originating pixel will assume by using a composition function (Eq:1-4), where the accumulated color A_{rgb} is the color of the pixel (i, j), and A_a is the alpha component of the pixel which is set to 1 at the end of the render process. Given an (x, y, z) coordinate in the volume and a step k of the ray, V_a is the scalar value of the volume V, V_{rgb} is the color defined by the transfer function given V_a, S are the sampled values of the ray and O_f, L_f are the general Opacity and Light factors.

$$S_a = V_a * O_f * \left(\frac{1}{s}\right) \tag{1}$$

$$S_{rgb} = V_{rgb} * S_a * L_f \tag{2}$$

$$A^k_{rgb} = A^{k-1}_{rgb} + \left(1 - A^{k-1}_a\right) * S_{rgb} \tag{3}$$

$$A^k_a = A^{k-1}_a + S_a \tag{4}$$

3.1 Data Processing and volume interpolation

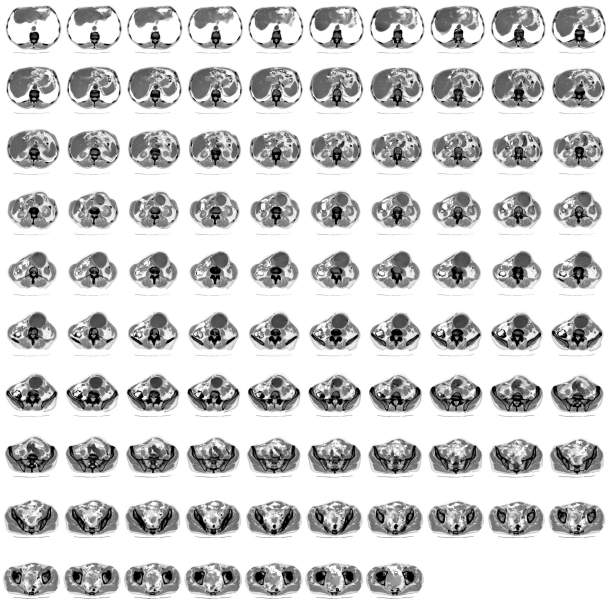


Figure 3: *Aorta dataset in mosaic form to be read by the shader*

3.2 Contribution

The images for one volume are composed into a single image containing all slices that will be stored in a texture as shown in Figure 3. This texture is generated by tilling each slice one besides other in a matrix configuration, this step was implemented as a preprocessing step in our algorithm. The size of the texture in GPU memory could change from 4096×4096 in PC to 1024×1024 for handheld devices. The reduction in quality in the image is explained in Figure 7. The number of images per row and the number of rows as well as the total number of slices must be given to the shader.

In medical images the sample bit depth is commonly bigger than 8 bits. This is hard to handle in Web applications where commonly

supported formats are limited to 8 bits per sample. In this work, medical data sets were reduced to 8 bits.

Higher depths could be supported using more than one color component to store the lower and higher bits of each pixel but this representation is not currently implemented in our shader.

$$s_1 = \text{floor}(z * S) \tag{5}$$
$$s_2 = s_1 + 1 \tag{6}$$
$$dx_1 = \text{fract}(\frac{s_1}{M_x}) \tag{7}$$
$$dy_1 = \frac{\text{fract}\left(\frac{s_1}{M_y}\right)}{M_y} \tag{8}$$
$$dx_2 = \text{floor}(\frac{s_2}{M_x}) \tag{9}$$
$$dy_2 = \frac{\text{fract}\left(\frac{s_2}{M_y}\right)}{M_y} \tag{10}$$
$$tx_1 = dx_1 + \frac{x}{M_x} \tag{11}$$
$$ty_1 = dy_1 + \frac{y}{M_y} \tag{12}$$
$$tx_2 = dx_2 + \frac{x}{M_x} \tag{13}$$
$$ty_2 = dy_2 + \frac{y}{M_y} \tag{14}$$
$$v_1 = \text{tex2D}\left(tx_1, ty_1\right) \tag{15}$$
$$v_2 = \text{tex2D}\left(tx_2, ty_2\right) \tag{16}$$
$$V_a(x, y, z) = \text{mix}\left(v_1, v_2, (x \times S) - s_1\right) \tag{17}$$

For the correct extraction of the value of the volume two equations were implemented. The equations 5-17 show how to get the value of a pixel in coordinates x, y, z from images presented in an cartesian grid. S is the total number of images in the mosaic and M_x, M_y are the number of images in the mosaic in each row and column as the medical dataset show in Figure 3.

The functions presented in the equations are defined by the GLSL specification. This allow us to manipulate the images as continuous values because the functions of data extraction from the texture utilize interpolation.

3.2.1 Identification of Ray coordinates

The geometry of a cube is generated with coordinates from $(0, 0, 0)$ to $(1, 1, 1)$. This cube represents the boundary of the volumetric dataset and is painted with colors representing the coordinates at each point x, y, z coordinates (Figure 2)are stored in the r, g, b color component of each pixel. The cube is then rendered in the scene from the desired view point. The rendering process has several steps. The first two steps are the rendering of the color cube with the depth function change. Then one of the passes presents the closest region of the cube to the camera (Figure 2(a)), and the second pass presents the far region (Figure 2(b)).

With these two renders a ray is calculated from each point in the cube for the render with the faces closest to the eye and the end of the ray with the point of the back region. The colors in the cube represent the exact coordinates of the ray for each pixel in the image. We store the color information of the cube as 24 bit RGB values. This range of values seems to be small and not precise enough for big images, but color interpolation gives enough precision for the ray coordinates.

Cartesian coordinates Most voxel-based volume datasets are arranged in a cartesian uniform grid. A medical CT or MRI scanner, computes parallel slices of the specimen at different positions with a normally constant spacing. Each image contains a matrix of samples of whatever variable the specific equipment measures. By stacking all slices aligned together, a discretely sampled volume is defined. Each sample can be addressed by cartesian x, y, z coordinates, one being a slice selector and the other two coordinates of a point in that slice image.

Spherical coordinates A weather radar scans the surrounding sky in successive sweeps. Beginning at a low angular elevation, the radar performs a 360° azimuth scan (Figure 4, fig:radar). At each one-degree space direction a ray is emitted and a number of samples along the ray are measured back from its echoes (400 samples or buckets in our case). The radar then proceeds step by step increasing elevation at each successive swept scan. Elevation angles are not normally uniformly incremented because most interesting data is at the lower levels. Our datasets use 14 such elevations.

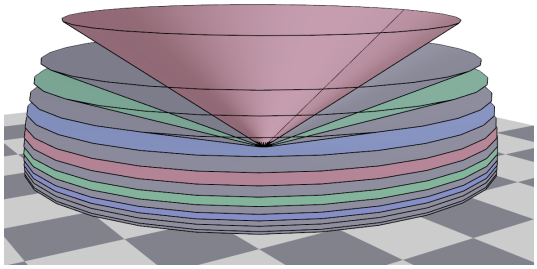


Figure 4: *Simplified geometry of a radar scan. Each scan can be approximated as a cone. Therefore, a radar volume dataset is approximated as a set of co - axial cones with the radar in the common apex.*

Such a process results in a discrete sampling of the sky volume in which each sample has *elevation*, *azimuth* and *range* coordinates. Thus, samples can then be addressed by spherical coordinates. In the conversion of raw radar data into input images suitable for the WebGL implementation the sample values become pixel values. Each swept scan for a fixed elevation angle forms one image in which pixel columns correspond to each azimuth direction (there are 360 columns), and rows correspond to distances along each ray (400 rows). Each image maps to a conical surface in space as shown in figure 4. The images from consecutive elevations are joined to form a single image for the whole volume, presented in figure 5 (the figure is rotated 90°).

$$r = \sqrt{x^2 + y^2 + z^2} \tag{18}$$
$$\varphi = \arctan(y, x) + \pi \tag{19}$$
$$\theta = \arccos(z/\varphi) \tag{20}$$

For the spherical coordinates volume dataset from a radar the following Equations 18-20. Where used. The interpolation process used for the identification of the volume value in an arbitrary point is presented in Equations 5-17. We use a simple interpolation method because the data is expected to be normalized from the capture source. The problems presented in this topic were explained by [Segura et al. 2009].

3.2.2 Ray generation

The ray is generated for each pixel in the image I, geometrically the start and end positions of the ray are extracted from the previous render passes with the information of the color cube. The ray is divided by S steps, which indicates the number of samples of the volume. For each sample the x, y, z inside the volume is calculated and the value of that position is interpolated from the texture.

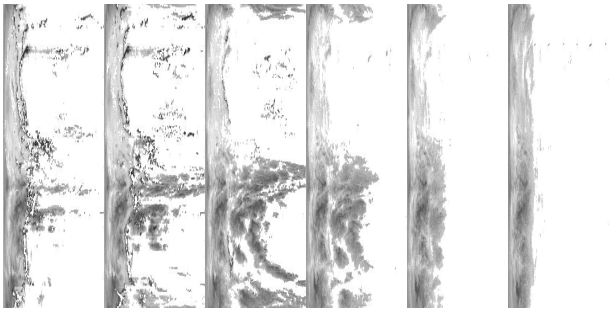


Figure 5: *Original Doppler radar image. Each vertical band represents data along the cones described in Figure 4*

Terminology	Meaning
Chrome	Chrome 9.0.597.98
Opera	Opera 11.50 Labs b24661
FirefoxDX	Firefox Minefield 4.0b13pre
	with Angle and shader validation
FirefoxGL	Firefox Minefield version
	without Angle or shader validation

Table 1: *Terminology specification.*

3.2.3 Transfer function

This value of the texture $t_{x,y,z}$, is then used to identify the color to be used in the composition function (Eq:1). When the composition function reach to the end of the ray in the cube or the accumulated alpha $A_a$is completed the ray finishes. and the color A_{rgb} for the ray in each pixel is assigned with this value.

4 Results

The proposed GPU implementation of the Volume Rendering technique presented in the previous sections has been tested with different settings and with different datasets. As the interactive and real - time results depend on both hardware and software, it is very important to begin with the platform specification used for the testing. In the following sections, medical volumetric datasets and weather radar volume samples are used to validate that WebGL is a valid and promising technology for real time and interactive applications.

4.1 Hardware and Software configuration

The tests for this article have been conducted using an Intel Quad Core Q8300 processor, 4GB of RAM and a GeForce GTX 460, Windows 7 PRO 64 bits (Service Pack 1) with the latest stable graphics drivers. Among all the Web browsers with full implementation of WebGL standard, we have selected the following ones: FireFox Minefield 4.0b12Pre (2011/02/16) ([web d]), Chrome 9.0.597.98 ([web b]) and Opera 11.50 labs (build 24661 - [web h]). It is worth to point out that both Chrome and Firefox in its default configuration use Google's Angle library ([web a]) to translate WebGL's native GLSL shaders to Microsoft's HLSL language and compile and run them through the DirectX subsystem. This procedure improves compatibility with lower-end hardware or older graphics drivers. Firefox Minefield has been configured with two different settings by modifying some keys in the configuration page *about:config*: (1) the default value of *webgl.prefer-native-gl* was set to `TRUE`. (2) The default value of *webgl.shader_validator* was set to `FALSE`. These changes basically disable Angle as the rendering back-end end validator of shaders, thus directly using the underlying native OpenGL support.

The reader is invited to visit Table 1 for terminology precisions.

Browser	N. Steps	Load Time (msec)	frame rate (frame/sec)	Memory (MB)
FirefoxDX	80	16677	69	204
FirefoxGL	80	308	94	102
Chrome	80	18062	77	153
Opera	80	82	108	74
FF-nA	140	302	60	95
Opera	140	118	66	73
FirefoxGL	170	281	51	95
Opera	170	102	54	77
FirefoxGL	200	312	44	96
Opera	200	111	48	81

Table 2: *Results table for the medical dataset.*

A LightTPD Web server ([web g]) was installed and configured in the same computer, to serve the dataset images, the sample webpages (HTML and Javascript files) and the vertex and fragment shaders.

4.2 Medical Dataset

Figure 6 shows some graphical output for the medical dataset introduced in the previous section. The 6 different axial views have been generated using 80 steps in the shaders implementation (800×800 canvas rendered in the *FirefoxDX* configuration). Table 2 displays the recorded statistics: Column 1: Browser configuration. Column 2: Number of steps selected in the shaders. Column 3: Loading times (milisec). Column 4: Qualitative frame rate (fps). Column 5: Memory usage (MB). The dataset parameters and volume rendering shaders proved to be very influential in the experiment with WebGL rendered volume datasets. In order to realize the tests under comparable conditions, the web-browsers were stopped and their caches emptied after each test execution. For numbers of steps larger than 80 *Chrome* and *FirefoxDX* failed to compile the shader. Therefore, only *Opera* and *FirefoxGL* statistics are shown. For numbers of steps smaller than 80, the measured FPS was truncated to 128 in all configurations due to the selected measurement method.

4.2.1 Memory Usage

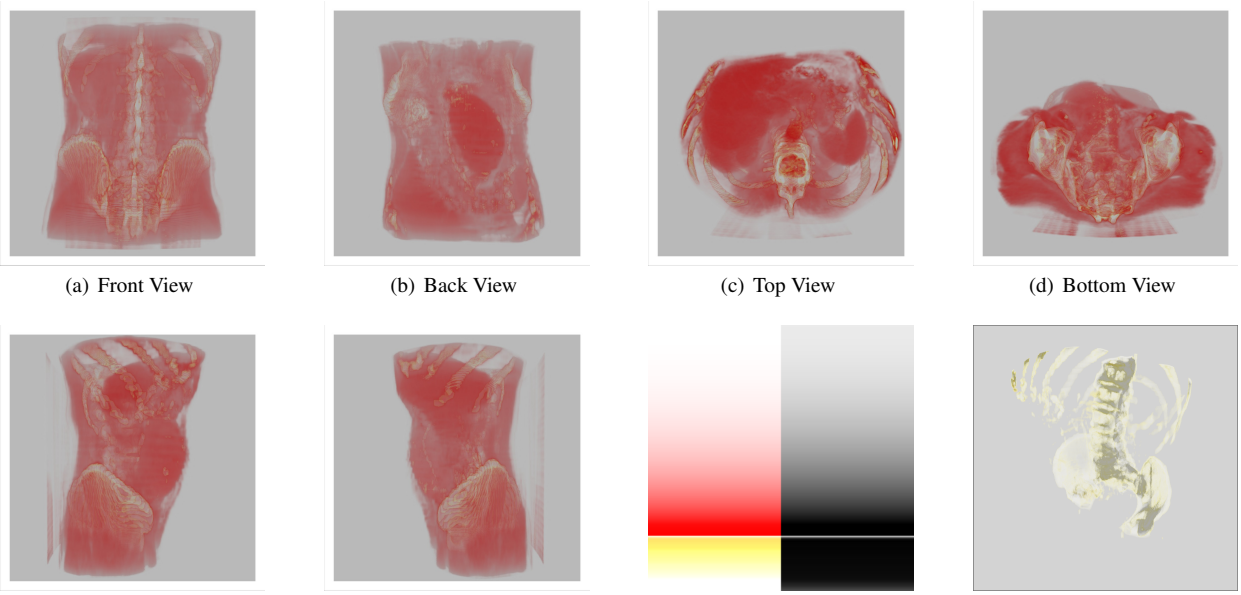
Regarding memory usage, *Opera* showed to be the least demanding browser, being *FirefoxDX* the most consuming one. *FirefoxGL* dropped the memory usage resembling the one of *Opera*. This fact leads us to infer that the current Angle implementation is the key factor in the memory management since *Chrome* has consumption similar to *FirefoxDX*. For *Opera*, *FirefoxDX* and *FirefoxGL*, a simple subtraction between the final and initial memory allocation suffices to estimate the memory consumption. On the other hand, since *Chrome* implements the *Out of Process* technology, we have included *all* running processes of *Chrome* (2 processes at the starting time and 3 processes after WebGL was loaded).

4.2.2 Loading Times

The loading time includes the (1) downloading of all the required files (assuming a locally installed Web server), (2) compilation of the shaders and (3) first rendering of the webpage, including the volume rendering. The results follow a similar schema in which *Opera* and *FirefoxGL* are significantly faster than *Chrome* and *FirefoxDX*, due to the longer compiling time and shader validation of Angle.

4.2.3 Frame Rate

Since the frame rate cannot be precisely determined, we have devised an empirical measuring method. We forced the GL canvas to be redrawn continuously and then we have counted how many times the scene was rendered in a 5-seconds interval. We have repeated this procedure 5 times, choosing the median value (i.e. after



(a) Front View (b) Back View (c) Top View (d) Bottom View

(e) Left Side View (f) Right Side View (g) Used Transfer Function (h) Applying Other TF

Figure 6: *Subfigures (a), (b), (c), (d), (e) and (f) illustrate renderings of the axial views of the sample volume dataset. The output was generated in 800×800 with 80 steps using FirefoxDX. Subfigure (g) depicts the applied transfer function, where the left side represents the color and the right side the transparency (black=opaque, white=transparent). With different transfer functions other outputs are obtained, as subfigure (h) shows.*

removing the two highest and the two smallest values) as the effective frame rate. The results show that the frame rate is truncated to 128 fps at maximum (not shown in Table 2). This is considered to be a side effect of the chosen measurement method, based on the Javascript `setTimeout()` function. Even with a parameter of 0 ms the browsers take a minimum time to call the corresponding function, being it 10 ms in average for desktop Web browsers. Therefore, it is preferable to increase the number of steps in order to get smaller frame rates and reduce this side effect. With higher values of steps (only usable with *Opera* or *FirefoxGL*) *Opera* is slightly faster, consuming less memory and requiring smaller loading times.

4.2.4 Dataset Resolution

This qualitative test was intented to show how the input dataset resolution affects the final rendering quality. Using the same dataset, a modified version was created by reducing the input resolution per slice from 5120×5120 to 1280×1280 (a reduction to 25% per dimension or to 6.25% globally). The number of steps in the shaders were also varied, using 20, 50 and 80 steps with *FirefoxDX* setup and a selection of the results can be shown in Figure 7. If the number of steps is small the banding artifacts of the algorithm are noticiable, some aproximations could be implemented to solve this problem as show by [Marques et al. 2009].

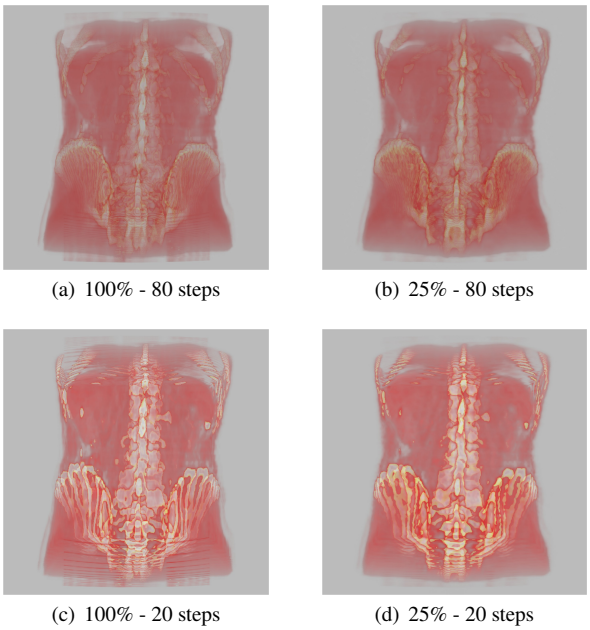
4.3 Medical Dataset in Portable Devices

The Mozilla Firefox Development Group has released a mobile version of the browser for ARM devices called Fennec ([web c]). We have tested it on 2 Android-based devices: Samsung Galaxy Tab ([web j]) and Samsung Galaxy S smartphone ([web i]). Taking into account the hardware limitations of such devices, we have scaled down the *Aorta* dataset to half resolution, reduced the HTML canvas size and chosen a suitable number of steps to get quality results with the highest possible interactivity. The test under this browser was quite straight-forward. No further modification in the implementation of the shaders, Glue Javascript code or HTML Web page were required. Although we achieved a low frame rate (about 2 or 3 frames per second), it was proved as possible the rendering of volume datasets in such mobile devices. Further optimizations in the data or the implementation of the shaders, specifically oriented to such devices, might result in better overall performance. We left such issues for future efforts.

4.4 Weather Radar Volumetric Dataset

Weather radars are devices used to scan the surrounding atmosphere and determine its structure and composition, typically using the Doppler effect ([Segura et al. 2009]). Radar scans are represented as 2D images in the form of either PPI (plan position indicator) or CAPPI (constant altitude PPI) formats. The volumetric radar - scanned data may be approximated by a set of concentric cones (figure 4), with each cone containing a sample set of the volume (figure 5). The volume - rendering algorithms, therefore, must work with spherical coordinates for this purpose. This implementation of the volume rendering can only be tested under *Opera* and *FirefoxGL*. Otherwise the shader compilation fails.

In the case studies we have created a simple HTML user interface using jQuery (Figure 9) to interact with the shader. It allows the tunning of parameters such as the window (zoom and offset), quality (number of steps) and the transfer function (adapted specifically for this weather radar information). The zoom and pan allow the users to conveniently interact with the radar data, which is not as regular as the medical images. For instance, the useful information is found in the bottom of the volume (i.e. near the terrain). In addition, the resolution in outer zones is lower than near the radar source. Due to their geometrical configuration, the large area over radars is rarely scanned. Therefore, additional navigation controls for zoom and pan have been implemented in the sample Web page, allowing in- and out- zooming in the volume and panning the view. The radar captures periodic scans every 10 minutes. Therefore, some navigational functionality has been added to help users to access the next or previous data in the sequence. As the



(a) 100% - 80 steps (b) 25% - 80 steps

(c) 100% - 20 steps (d) 25% - 20 steps

Figure 7: *Resolution qualitative test. Cases (a) and (b) use 80 steps in the rendering. Cases (c) and (d) use 20 steps. Cases (a) and (c) correspond to the full resolution dataset. Cases (b) and (d) correspond to the reduced dataset. Even with the dramatic reduction of the resolution, the volume render allows to identify the main structures.*

loading time is rather short, the possibility to create fully interactive 4D animations is totally open.

A very simple HTML-based editor for transfer functions was implemented, which allows the proper analytic inspection of the radar data by changing the values and colors with the provided sliders. Figure 10 shows different visualization of the same radar sample, obtained by changing the transfer function (affecting colors and opacity). The figure also shows the effect of variations in camera parameters (zoom, pan and view orientation). The chosen number of steps was high enough to display a 800×800 canvas with high quality images and yet to keep the visualization frame rate above 26 frames/second.

5 Contribution and Complexity Analysis

Our contribution is an implementation of a volume rendering system for the Web. The system is based on the Volume Ray-Casting algorithm with a complexity of $O(M * S)$, where M is the number of pixels to be drawn and S is the number of steps of the ray that traverses the volume. Since the algorithm is implemented in WebGL, its visualization speed is similar to native applications because it uses the same accelerated graphic pipeline. The original algorithm has been slightly modified to work with the input structures because of the lack of Volume Textures in WebGL. Therefore, our algorithm simulates the 3D data by using a 2D tiling map of the slices of the volume maintaining the tri-linear interpolation, so there is not a real loss in quality because the interpolation is the same as the used in the GPU. Even thought a slight impact in performance could be generated for this interpolation, this is minimal and very difficult to perceive because the browsers are not capable of handle such fast events. This is so because the browsers are heavily dependent on several layers such as the shader compilers, hardware architecture, graphic drivers, etc. Our algorithm was designed to run entirely in the client (which is the novelty of our proposal). Some delays are

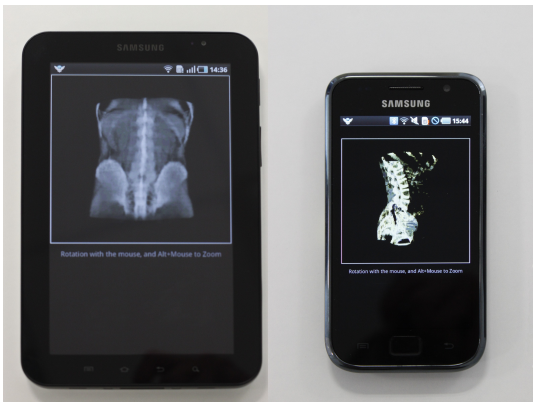


Figure 8: *A Samsung Galaxy Tab (left) and a Galaxy S Smartphone (right) volume - rendering medical datasets.*

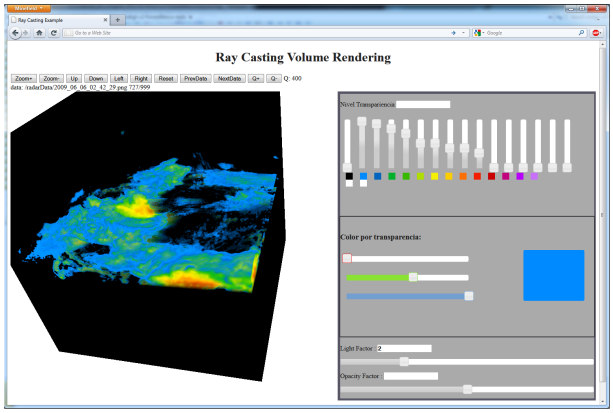


Figure 9: *Application to define the transfer function for radar reflectivity visualization.*

obviously expected because of the network performance, and the interpreted nature of Javascript. Our implementation[1] does not evidence a real overhead for the server to present the data in 3D as occurs in [Mahmoudi et al. 2009], therefore allowing more clients to be connected simultaneously. At the same time, more power-full clients are required to handle this approximation.

The limitations in our method, even been WebGL compilant, stem from the fact that some browsers do not adequately provide powerful enough shader language implementations to even allow compilation of larger shader programs.

6 Conclusions and future work

Two different case studies (medical- and weather radar-imaging) presented here illustrate the capabilities of complex volume rendering visualization in Web browsers. Although many performance improvements and optimizations are still needed, the material discussed indicates that rendering volumetric data with Web standard technology is applicable to many other technical fields. Such an initiative also re - ignites interest for visualization functions implemented in the past for high-end desktop visualization applications. The integration of our implemented software in the Web follows the upcoming HTML5 standard, namely a Javascript API and the new WebGL context for the HTML5 canvas element (which gives the application a professional look). The implementation of the algorithm in declarative languages as X3DOM is planed.

[1] http://demos.vicomtech.org/volren

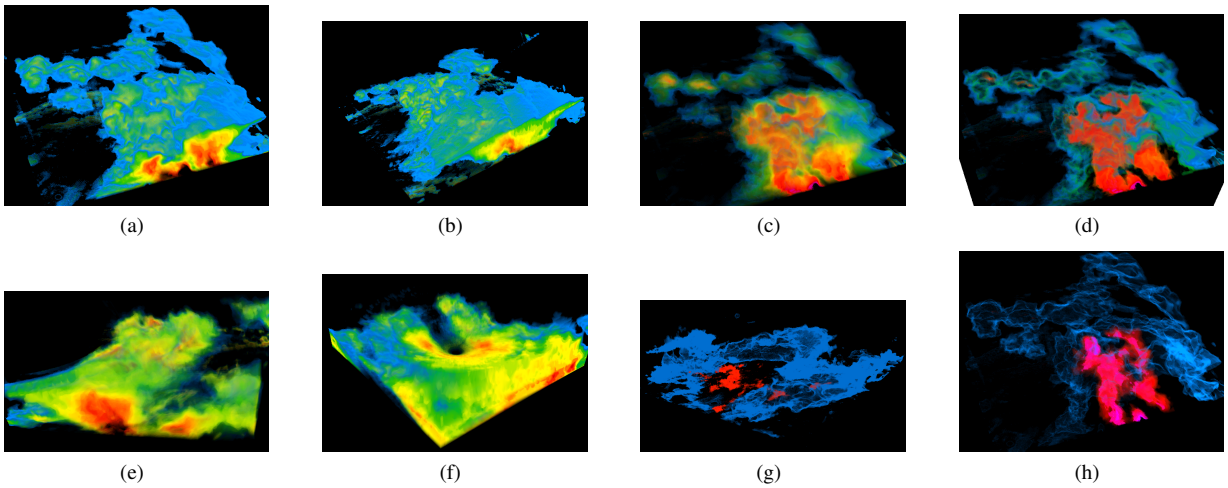


Figure 10: *Different weather radar volume renderings. Images (a) and (b) use the traditional color mapping for reflectivity scans (measured in decibels, dBZ). Images (c), (d), (e), (f), (g) and (h) have been generated by varying the transfer function (color and transparency) and the window zoom and pan.*

The scope of the present article did not include the integration of different rendering styles. However, interactive and complex lighting integration are promising ways to improve render quality. The use of multi - dimensional interactive transfer functions is also an promising direction to explore. The (marginal) optimizations that we already applied in this work allow us to expect that mathematically-planned negotiation between speed performance and quality is a promising research field. An additional goal for minimization is the optimized handling of time-varying datasets using videos instead of images as render input. since video formats already minimize transmitted data by reducing temporal redundancy, it would be possible to diminish the bandwidth currently required, for example, for animated render of radar - scanned weather phenomena.

Another important evolution will be the integration of surface rendering within volume-rendered scenes in order to visualize, for example, segmented areas in medical images or terrain surfaces. Some tests have already been performed on desktop prototypes. This article lays the technical ground that would make the integration of surface render in volume-render (via WebGL) possible and reasonable.

Acknowledgments

This work was partially supported by the Basque Government's ETORTEK Project (ISD4) research programme and CAD/CAM/-CAE Laboratory at EAFIT University and the Colombian Council for Science and Technology –COLCIENCIAS–. Radar datasets were provided by the Basque Meteorology and Climatology Department.

References

BEHR, J., AND ALEXA, M. 2001. Volume visualization in vrml. In *Proceedings of the sixth international conference on 3D Web technology*, ACM New York, NY, USA, 23–27.

BEHR, J., ESCHLER, P., JUNG, Y., AND ZÖLLNER, M. 2009. X3dom: a dom-based html5/x3d integration model. In *Proceedings of the 14th International Conference on 3D Web Technology*, ACM, 127–135.

BLAZONA, B., AND MIHAJLOVIC, Z. 2007. Visualization service based on web services. *Journal of Computing and Information Technology 15*, 4, 339.

CONGOTE, J., MORENO, A., BARANDIARAN, I., BARANDIARAN, J., AND RUIZ, O. 2010. Extending marching cubes with adaptative methods to obtain more accurate iso-surfaces. In *Computer Vision, Imaging and Computer Graphics. Theory and Applications International Joint Conference, VISIGRAPP 2009, Lisboa, Portugal, February 5-8, 2009. Revised Selected Papers*, A. Ranchordas, J. M. Pereira, H. J. Araújo, and J. M. R. S. Tavares, Eds., vol. 68 of *Communications in Computer and Information Science*. Springer Berlin / Heidelberg, January, 35–44.

GOENETXEA, J., MORENO, A., UNZUETA, L., GALDÓS, A., AND SEGURA, A. 2010. Interactive and stereoscopic hybrid 3d viewer of radar data with gesture recognition. In Romay et al. [Romay et al. 2010], 213–220.

HADWIGER, M., LJUNG, P., SALAMA, C. R., AND ROPINSKI, T. 2009. Advanced illumination techniques for gpu-based volume raycasting. In *ACM SIGGRAPH 2009 Courses*, ACM, 1–166.

HARTLEY, R., AND ZISSERMAN, A. 2003. *Multiple View Geometry in Computer Vision*, second ed. Cambridge University Press, Cambridge, UK.

HIBBARD, W., AND SANTEK, D. 1989. Interactivity is the key. In *Proceedings of the 1989 Chapel Hill workshop on Volume visualization*, ACM, New York, NY, USA, VVS '89, 39–43.

JOHN, N., ARATOW, M., COUCH, J., EVESTEDT, D., HUDSON, A., POLYS, N., PUK, R., RAY, A., VICTOR, K., AND WANG, Q. 2008. MedX3D: standards enabled desktop medical 3D. *Studies in health technology and informatics 132*, 189.

JOHN, N. W. 2007. The impact of web3d technologies on medical education and training. *Computers and Education 49*, 1, 19 – 31. Web3D Technologies in Learning, Education and Training.

JUNG, Y., RECKER, R., OLBRICH, M., AND BOCKHOLT, U. 2008. Using x3d for medical training simulations. In *Web3D '08: Proceedings of the 13th international symposium on 3D web technology*, ACM, New York, NY, USA, 43–51.

KABONGO, L., MACA, I., AND PALOC, C. 2009. Development of a commercial cross-platform dicom viewer based on open source software. In *International Journal of Computer Assisted Radiology and Surgery; CARS 2009 Computer Assisted Radiology and Surgery Proceedings of the 23rd International Congress and Exhibition*, Springer, Berlin, Germany, P. H. U. Lemke, P. P. K. Inamura, P. P. K. Doi, P. P. M. W. Vannier, P. P. A. G. Farman, and D. PhD, Eds., vol. 4, International Foundation of Computer Assisted Radiology and Surgery, S29–S30.

KRUGER, J., AND WESTERMANN, R. 2003. Acceleration techniques for gpu-based volume rendering. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, IEEE Computer Society, Washington, DC, USA, 38.

LACROUTE, P., AND LEVOY, M. 1994. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '94, 451–458.

LEVOY, M. 1988. Display of surfaces from volume data. *IEEE Comput. Graph. Appl. 8*, 3, 29–37.

MAHMOUDI, S. E., AKHONDI-ASL, A., RAHMANI, R., FAGHIH-ROOHI, S., TAIMOURI, V., SABOURI, A., AND SOLTANIAN-ZADEH, H. 2009. Web-based interactive 2d/3d medical image processing and visualization software. *Computer Methods and Programs in Biomedicine In Press, Corrected Proof*, –.

MARQUES, R., SANTOS, L. P., LEŠKOVSKÝ, P., AND PALOC, C. 2009. Gpu ray casting. In *17 Encontro Português de Computaçao Gráfica*, En Anexo, Covilha, Portugal, A. Coelho, A. P. Cláudio, F. Silva, and A. Gomes, Eds., 83–91.

MARRIN, C. *WebGL Specification*. Khronos WebGL Working Group.

MEISSNER, M., HUANG, J., BARTZ, D., MUELLER, K., AND CRAWFIS, R. 2000. A practical evaluation of popular volume rendering algorithms. In *Proceedings of the 2000 IEEE symposium on Volume visualization*, Citeseer, 81–90.

MEYER-SPRADOW, J., ROPINSKI, T., MENSMANN, J., AND HINRICHS, K. H. 2009. Voreen: A rapid-prototyping environment for ray-casting-based volume visualizations. *IEEE Computer Graphics and Applications (Applications Department) 29*, 6 (Nov./Dec.), 6–13.

PHONG, B. T. 1975. Illumination for computer generated pictures. *Commun. ACM 18*, 6, 311–317.

POLIAKOV, A. V., ALBRIGHT, E., HINSHAW, K. P., CORINA, D. P., OJEMANN, G., MARTIN, R. F., AND BRINKLEY, J. F. 2005. Server-based approach to web visualization of integrated three-dimensional brain imaging data. *Journal of the American Medical Informatics Association 12*, 2, 140 – 151.

RILEY, K., SONG, Y., KRAUS, M., EBERT, D. S., AND LEVIT, J. J. 2006. Visualization of structured nonuniform grids. *IEEE Computer Graphics and Applications 26*, 46–55.

ROMAY, M. G., CORCHADO, E., AND GARCÍA-SEBASTIÁN, M. T., Eds. 2010. Hybrid Artificial Intelligence Systems, 5th International Conference, HAIS 2010, San Sebastián, Spain, June 23-25, 2010. Proceedings, Part I, vol. 6076 of *Lecture Notes in Computer Science*, Springer.

SCHARSACH, H. 2005. Advanced GPU raycasting. *Proceedings of CESCG 5*, 67–76.

SEGURA, Á., MORENO, A., GARCÍA, I., AGINAKO, N., LABAYEN, M., POSADA, J., ARANDA, J. A., AND ANDOIN, R. G. D. 2009. Visual processing of geographic and environmental information in the basque country: Two basque case studies. In *GeoSpatial Visual Analytics*, R. D. Amicis, R. Stojanovic, and G. Conti, Eds., NATO Science for Peace and Security Series C: Environmental Security. Springer Netherlands, October, 199–208.

STEGMAIER, S., STRENGERT, M., KLEIN, T., AND ERTL, T. 2005. A Simple and Flexible Volume Rendering Framework for Graphics-Hardware–based Raycasting. In *Proceedings of the International Workshop on Volume Graphics '05*, 187–195.

SUNDARAM, V., ZHAO, L., SONG, C., BENES, B., VEERAMACHENENI, R., AND KRISTOF, P. 2008. Real-time Data Delivery and Remote Visualization through Multi-layer Interfaces. In *Grid Computing Environments Workshop, 2008. GCE'08*, 1–10.

TSCHIRLEY, R., KÖCHY, K., AND MÄRKLE, S. 2002. Patient-oriented segmentation and visualization of medical data. *Proceedings of CGIM*, 214–219.

WEB - Angle Project Homepage. http://code.google.com /p /angleproject.

WEB - Chrome Web Browser Homepage. http://www.google.com /chrome.

WEB - Fennec: Mozilla Firefox Mobile. http://www.mozilla.com /en-US /mobile.

WEB - Firefox 4.0b12pre (2011/02/16). http://ftp.mozilla.org /pub /mozilla.org /firefox /nightly /2011-02-16-03-mozilla-central.

WEB - Firefox Minefield. http://www.mozilla.org /projects /minefield.

WEB - ImageVis3D: A Real-time Volume Rendering Tool for Large Data. Scientific Computing and Imaging Institute (SCI). http://www.imagevis3d.org.

WEB - LightTPD Web server Homepage. http://www.lighttpd.net.

WEB - Opera Labs WebGL 11.50. http://snapshot.opera.com /labs /webgl /Opera_1150_24661_WebGL_en.exe.

WEB - Samsung Galaxy S Smartphone Homepage. http://galaxys.samsungmobile.com.

WEB - Samsung Galaxy TAB Homepage. http://galaxytab.samsungmobile.com /2010 /index.html.

WESTOVER, L. A. 1991. *Splatting: a parallel, feed-forward volume rendering algorithm*. PhD thesis, Chapel Hill, NC, USA. UMI Order No. GAX92-08005.

WONG, S. T., WHALEY, P. S., ANG, C. S., HOO, K. S., WANG, J., AND HUANG, H. K. B. 1996. Interactive query and visualization of medical images on the World Wide Web. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, Y. Kim, Ed., vol. 2707 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, 390–401.

YOO, S., KEY, J., CHOI, K., AND JO, J. 2005. Web-Based Hybrid Visualization of Medical Images. *Lecture notes in computer science 3568*, 376.