Volume Ray Casting in WebGL

John Congote, Luis Kabongo, Aitor Moreno,
Alvaro Segura, Andoni Beristain, Jorge Posada
Vicomtech Research Center
Spain

Oscar Ruiz
EAFIT University
Colombia

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.

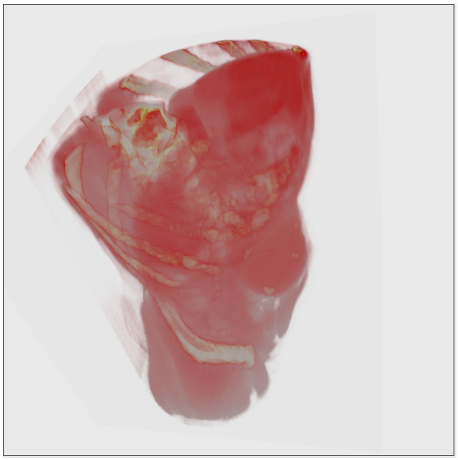


Fig. 1. Medical data rendered with volume ray-casting

However, some scientific fields (e.g. medicine, geo-sciences, meteorology, engineering) work with 3D volumetric datasets. Volumetric datasets are regular 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 chapter, 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 Direct Volume Rendering algorithms are needed, which are not available for the Web. Ray Casting is a common technique for volume visualization which displays the saliend characteristics of the volume set, although it is not photo-realistic.
Therefore, our work uses *Volume Ray-Casting*, which is a common technique in Computer Graphics for volume visualization. Originally presented by Kajiya Kajiya & Von Herzen (1984) as an extension of *Ray-tracing* algorithm, then Levoy Levoy (1988) defines the volume render-

ing. This has been further studied in Hadwiger et al. (2009). The aforementioned rendering is not photo-realistic, however it shows important characteristics of the dataset.
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), Fogal & Kruger (2010)) for interactive visualization (Mahmoudi et al. (2009)). Reference Kabongo et al. (2009) presents on some of the few implementations of volumetric data displays.
WebGL is a new standard for accelerated 3D graphics rendering on the Web that complements other technologies in the future HTML5 standard (Marrin (2011)). Some of the major Web browsers, including Google Chrome, Mozilla Firefox, WebKit, Safari and Opera have already implemented WebGL 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.
We present in this chapter an implementation of a volume rendering system for the Web, based on the Volume Ray-casting algorithm and implemented on WebGL. The system is capable of obtaining interactive visualization with diverse volume datasets (Figure 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.
This work was tested in bioinformatic scenarios with volumetric datasets such as medical imaging. Also, a metereological prototype was developed to visualize doppler radar datasets.
This chapter is organized as follows. Section 2 presents a brief status of the different technologies present in this work: Volume Rendering, Web rendering, medical and confocal visualization. Section 3 presents our methodology for volume rendering with special attention to the modifications of the algorithm for the Web environment. Section 4 shows the implementation of volume rendering for doppler wheather radars. Section 5 presents the output obtained by the implemented algorithm and the performance values in different conditions. Section 7 presents the conclusions of our work and future directions.

2. Related Work

2.1 Direct volume rendering techniques

In 3D scalar field interactive visualization, two solutions prevail: Surface Rendering and Direct Volume Rendering. Surface Rendering, which has the advantage of being easy to compute due to its low geometric complexity. It's main disadvantages are: i. A surface must be synthesized first, which is not a trivial task as it depends on the quality of the sample; ii. Since it must be precalculated, the result is static and cannot be easily adjusted in real time.
Recent advances in Direct 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 of images does not intervene in the images since there is no part of the DVR of the computations even when the camera is displaced. In order to project the volumetric

data, several methods exist (Meißner et al. (2000)). Westover Westover (1991) discusses *Volume Splatting* and represents each scalar value by a simple geometrical shape that will face the camera, allowing fast rendering. It's main disadvantage is the loss of quality. A technique called *Shear Warping* (Lacroute & 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, the method is quite fast, but it's 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 of re-slicing the volume depending on the orientation of the camera viewpoint, and representing all of the slices at once taking advantage of eventual occlusion optimizations (Hibbard & Santek (1989)), , *but the lack of specialized visualization methods in this algorithm has made it unussable for profesional appliactions such as medical imaging.*
Volume Ray-casting was initially presented by Kajiya Kajiya & Von Herzen (1984) as an extension of the Ray-Tracing algorithm for volumetric shapes. Later the methodology was formalized by Levoy Levoy (1988). Since then, Volume Ray-casting 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 makes it's usage initially difficult in real-time interactive representations, allowing other approximations to establish. Nowadays, the increasing computational power of graphic cards allows fast calculations (Kruger & 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 a starting point for the theoretical foundations in our implementation and for technical details. Open Source implementations such as Meyer-Spradow et al. (2009) and Fogal & Kruger (2010) were also used.

2.2 Web 3D rendering

The fact that the Web and 3D graphics are currently ubiquitous in desktop and handheld 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, and implementations on general purpose plug-ins, etc. A review of these techniques can 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 on-server rendering (Blazona & Mihajlovic (2007)). In this approach, the rendering process is 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 unresponsive and unsuitable for smooth interactive visualization.
Unresolved 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 chapter, 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 the GLSL language, which 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. It's imperative model enables 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, due to their low speed. However, the performance of JavaScript interpreters are 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

Across 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 important in this domain. Medical Visualization has already produced several implementations of volumetric visualization on the 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 & Alexa (2001)), visualization of volumes in the Web has been achieved.

3. Methodology

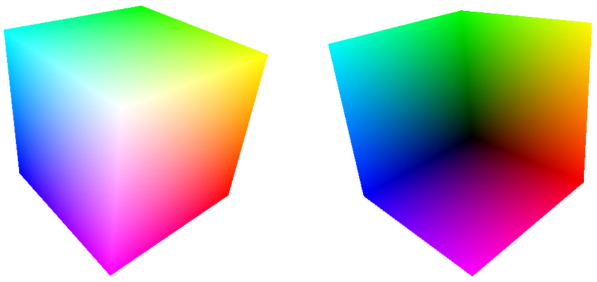

(a) Front Faces (b) Back Faces

Fig. 2. Color cube map coordinates

Direct 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 $I : [1, h] \times [1, w] \to \mathbb{R}^4$ (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 $V : [1,H] \times [1,W] \times [1,D] \to [0,1]$ (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 & Zisserman (2003)). The pin-hole camera model uses an intrinsic $K \in M_{3\times 4}$ and an extrinsic $R \in M_{4\times 4}$ real-valued matrices. These matrices project a 3D point $p \in \mathbb{P}^3$ onto a 2D point $p' \in \mathbb{P}^2$.
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 an approximation of the scalar field $V(p_s)$ is calculated, usually by 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 **??**. As the 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

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 beside the 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 the quality in the image is explained in Figure **??**. The number of images per row, the number of rows, and the total number of slices, must be given to the shader.

In medical images the sample bit depth is commonly greater than 8 bits per pixel. This is dificult to handle in Web applications where commonly supported formats are limited to 8 bits per sample. In this chapter, 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.
For the correct extraction of the value of the volume, two equations were implemented. The equations 5-17 show how to obtain 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 shows 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.

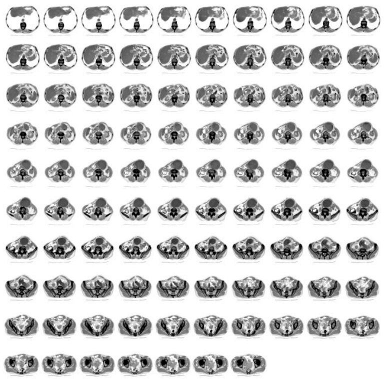


Fig. 3. Aorta dataset in mosaic form to be read by the shader

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

3.1.1 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 **??**)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.
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 relative to the specific signal measured by the equipment. 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.

3.1.2 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.

3.1.3 Transfer function

Transfer functions (TF) assign optical properties (color and opacity) to the original volume data values i, $F_{rgb\alpha}(i) = (r, g, b, \alpha)$, in order to improve the visualization of the internal parts of the volume data. In volume ray-casting, when the ray traverses the data, TF are used to obtain the optical properties of the volume data resulting representation at each ray step. These are then blended using the composition function. In general, two transfer functions are defined, the color transfer function, which obtains an RGB value and the opacity transfer function, which obtains an $Alpha$ value. For a recent review on transfer functions, the reader is referred to Arens & Domik (2010); Pfister et al. (2001).
Transfer functions can have many dimensions. One-dimensional transfer functions, like the one presented in this section, make a direct mapping between the original voxel data scalar values and the resulting optical values. On the other hand, multidimensional transfer functions do not only use the volume data values, but also additional information such us first and second derivatives Kniss et al. (2002) of the volume data, or even the position (based on a segmentation mask). This additional information allows a better separation between materials, and thus, better visualizations. The order of interpolation in the discrete space and the application of the TF defines the difference between *pre-* and *post- classification*. In pre-classification, TF values for the closest neighbors to the input raw data are obtained and then an interpolation is performed on the different TF color and opacity values. Using post-classification on

the other hand, first the raw data is interpolated to sample voxel data and then the TF is applied on it. Both approaches produce different results, whenever the interpolation does not commute with the transfer functions. As the interpolation is usually non-linear, it will only commute with the TF if the TF are constant or the identity. Post-classification is the "right" approach in the sense of applying the transfer functions to a continuous scalar field defined by a mesh together with an interpolation prescription. Nevertheless, some works try to achieve the correctness of post-classification in pre-classification approaches with a lower computational burden, like in Engel et al. (2001).
Independently of the TF dimensions, creating a suitable TF in order to highlight the most meaningful data for the user and an specific data set is a difficult, and usually an ad-hoc task which requires experienced users, and where an a-priori knowledge about the specific dataset is important too. The most common approach is to have a generic TF definition and then adapting or tuning it manually in an iterative trial and error process, until the meaningful data becomes visually salient. This is a time comsuming task completely dependant on the user's expertise and is why many contributions have been proposed in order to improve this workflow Pfister et al. (2001). Some improve the GUI tools to perform this manual tuning which simplify the procedure, like those present in the latest versions of the OsiriX Imaging software for DICOM visualization, while others include some kind of semiautomatic TF tunning based on the underlying data set, like in Wu & Qu (2007). Finally, some papers even present automatic TF tunning approaches, like in Zhou & Takatsuka (2009), but they are only valid for generic tasks since they are not specifically tailored to each particular need of the user.
All these works stand on the assumption that there is a correspondence between the cells formed by the TF input parameters and the different materials in the volume data. It is important to take this fact into account when selecting the TF input parameters and for a possible preliminary data processing step. Multidimensional functions permit a better separation between materials, since they consider more input variables, at the expense of being more difficult to tune and increased storage and computation requirements.
In practice, TF are usually precomputed and stored as lookup tables. This raises two issues: dynamic range and dimensionality. Ideally, the TF should store one resulting value per possible value in the input parameter space. Therefore, for one-dimensional 8-bit input data, the TF should have 256 entries, and for 16-bit input data, the TF should have 65536 different entries. In the same sense, a bidimensional TF with both parameters being in the range of 256 would require 256^2 values. The current hardware limitations, especially in GPU based implementations, require the reduction in size of the TF lookup tables, which in turn affects to the accuracy of the TF or requires methods to compress the TF. Finally, as stated by Kniss et al. (2003), it is not enough to sample the volume with the Nyquist frequency of the data field, because undersampling artifacts would still become visible. This problem is exacerbated if non-linear transfer functions are allowed. That is, the narrower the peak in the transfer function, the more finely we must sample the volume to render it without artifacts. Similarly, as more dimensions are added to the transfer function we must also increase the sampling rate of the volume rendering.
In GPU based direct volume rendering, the color and opacity TF are stored in the GPU as one or more textures $t_{x,y,z}$. Then, the value of the texture $t_{x,y,z}$ is used to identify the color and alpha to be used in the composition function (Eq:1), usually in a post-classification scheme, such as the one presented in this section. When the composition function reaches the end of the ray in the cube or the accumulated alpha A_a reaches its maximum, early termination, the

ray is interrupted and the resulting color A_{rgb} for the ray in the corresponding pixel is the cumulated value.

4. Meteorology

Doppler wheather radars are a type of remote sensing device used in meteorology. This use case scenario is described in its own section due to the particular issues it raises and the modifications required in the general volume rendering algorithm. A weather radar scans the space around it measuring several physical variables useful for the study of the current state of the local atmosphere (Segura et al. (2009)). A full scan of a Doppler radar produces a discretely sampled volume in which each sample contains several scalar values, namely reflectivity, differential reflectivity, radial velocity and spectral width. The distribution of samples in space is not uniform which poses specific challenges for analysis and visualization. Reflectivity in particular is linked to the amount of water present in the atmosphere and is especially representative of hydrometeors. Volume rendering of this variable may provide insight into the spatial shape of water masses in the air affecting different meteorological phenomena.

4.1 Radar visualization challenges

Even if the dataset is volumetric radar data is usually visualized in 2D representations, such as the *plan position indicator* (PPI) extracted at constant angular elevation, or the *constant altitude plan position indicators* (CAPPI) extracted at a constant altitude above sea level. 3D visualization is less common, but has even been presented in Web applications, for example in Sundaram et al. (2008). In this case, all rendering was precomputed in the server using OpenGL and Nvidia's Cg shading language and remotely displayed in the client by a VNC client applet. The authors mention a problem with scalability as a lot of processing is centralised. Also obviously, depending on network performance animations and interaction may not be smooth in such a setup.
Radar data visualization also poses new challenges as the data are acquired in a spherical coordinate system (Riley et al. (2006)) producing non-uniform sampled grids in space unlike the regular voxel volumes commonly found in medical datasets. This problem was Goenetxea et al. (2010) by rendering textured conic surfaces corresponding to each elevation. That method is only an approximation that does not fill the space between the cones. In order to represent the entire space using a ray-casting algorithm, the data set can be previously resampled in a uniform cartesian grid producing a voxel-based representation suitable for the common rendering algorithm. But given that resolution is not uniform in the radar data, resampling has its problems: if all detail is to be kept the resampled volume is huge, if memory use is to be limited, detail is lost. The method described below does not require preprocessing for resampling and does not suffer from said problems.

4.1.1 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). At each one-degree space direction a ray is emitted and a number of samples along the ray are measured back from its echoes (here 400 samples called buckets). The radar then proceeds step by step increasing elevation at each successive swept scan. Elevation angles are not normally uniformly incremented because most data of interest is at the lower levels. Our datasets use 14 such elevations from which only 5 had relevan information.

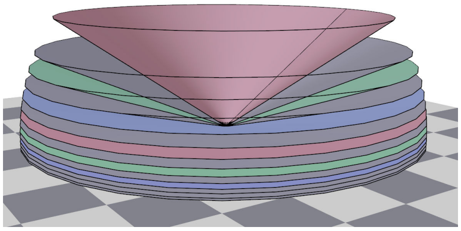


Fig. 4. Simplified geometry of a radar scan. Each scan can be approximated as a cone. Therefore, a radar volume dataset is approximated as a set of co-axial conic slices with the radar in the common apex.

The real process is more complicated because rays do not follow a straight line in the atmosphere. Salonen et al. (2003) and Ernvik (2002) explain this double phenomenon. Due to refraction, rays suffer changes in direction when crossing atmospheric layers with different index of refraction. In a standard atmosphere this makes beams bend downwards. On the other hand, as the Earth has a curved surface, a straight line leaving a radar has its altitude increasing non-linearly. If our vertical coordinate axis represents altitude, then straight rays in space appear to bend upwards in our coordinate system. Also, radar movement is slow so that the time at which the first and last elevation are scanned differ in several minutes.
For the purposes of our visualization radar rays will be considered straight, which is not a bad approximation for our limited range as both mentioned phenomena have an opposite effect. But in more accurate, larger scale scenarios, or when integrating data from more than one radar, they should be taken into account.
Such a scanning process results in a discrete sampling of the sky volume in which each sample has *elevation*, *azimuth* and *range* coordinates. Thus, samples can 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 columns correspond to azimuth directions (there are 360 columns spaced an angle of one degree), and rows correspond to distance along each ray (400 rows spaced 250 m in range). Each image maps to a conical surface in space as shown in figure 4. In order to store the entire volume as one image, the images from consecutive elevations are stacked vertically as seen in figure 5 (the figure is rotated 90° for presentation).

$$r = \sqrt{(x-0.5)^2 + (y-0.5)^2 + (z-0.5)^2} \tag{18}$$

$$\varphi = \arctan(y, x) + \pi \tag{19}$$

$$\theta = \arccos(z/\varphi) \tag{20}$$

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).

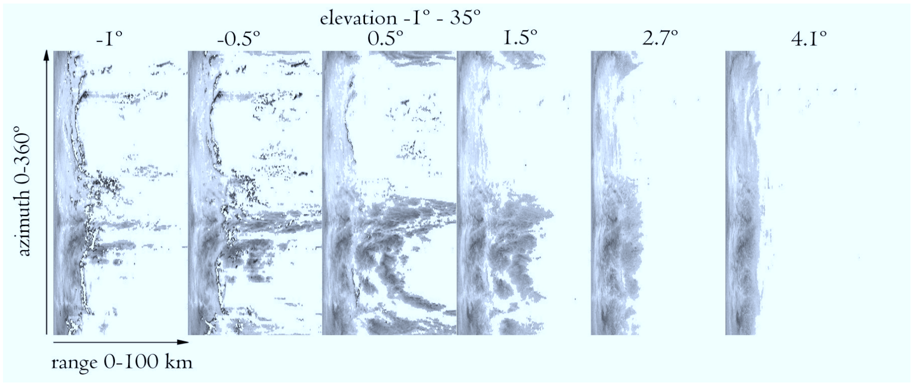


Fig. 5. Original Doppler radar image. Each vertical band represents data along a cone from figure 4 corresponding to an elevation angle

4.2 Weather radar volume rendering implementation

Our method extends the general volume ray casting algorithm previously presented in order to take into account the specific spherical coordinate system of the samples, and the layout of the dataset in memory. Thus, when traversing the volume, at each step the point's cartesian coordinates are transformed into spherical coordinates with 18. These coordinates are used to look up the volume value in the dataset. The non-uniform separation of elevations adds further complexity as it prevents a direct elevation to slice number conversion and imposes the use of a search process. As in the general algorithm, after the two slices above and below the sample are found, a value is interpolated from them.
It is important to note that each ray is traversed in tens to hundreds of steps, so this method requires a very high number of non-trivial computations and loops for each pixel, considerably more than in the general case. All of this conversion and search takes place in the programmed shader and results in very complex compiled shaders. At the time of writing only some WebGL implementations are capable of compiling and running them, specifically *Mozilla Firefox* in native OpenGL mode and *Opera*, and relatively powerful hardware is needed to achieve smooth rendering. Again, the technique would not be feasible for real-time use without the advent of modern programmable GPUs.

4.2.1 HTML user interface

We implemented a simple HTML user interface using jQuery UI (Figure 6) to interact with the volume rendering shader. It allows the tuning of parameters such as the window (zoom and offset), the quality (number of steps) and the transfer function (adapted specifically for this weather radar information), with immediate visual feedback.
The zoom and pan controls allow users to conveniently navigate the radar data, which is not as regular as medical images. For instance, the useful information is found in the bottom of the volume (i.e. near the ground). In addition, the resolution in outer areas is lower than near the radar source. Due to their geometrical configuration, the large area directly over radars is rarely scanned. Therefore, additional navigation controls for zooming and panning have been implemented in the sample Web page, allowing user interaction of zooming in and out, and panning the view.

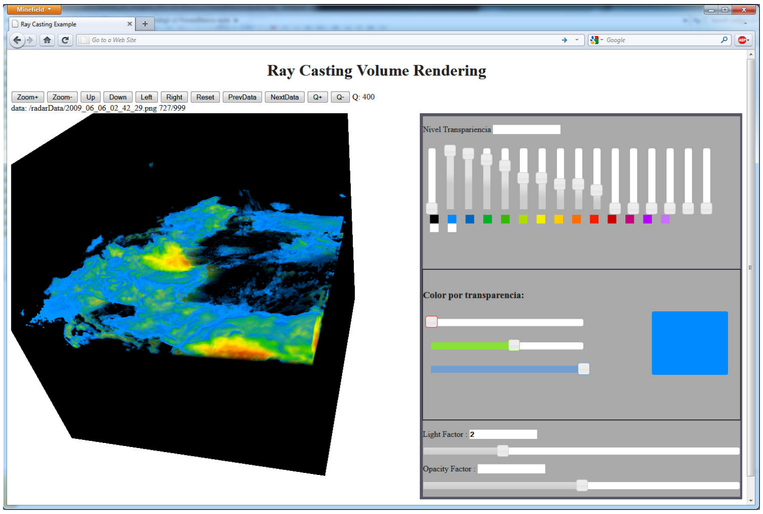


Fig. 6. Radar reflectivity visualization Web application with support for interactive definition of transfer functions.

Additional controls have been added to the user interface to allow users to modify the number of steps, a variable directly linked to the shader, and its modification triggers a recompilation. This option has been added only for scientific purposes, since final users should not be aware of such a concept.
A very simple HTML-based editor for transfer functions was implemented, which allows the proper inspection of the radar data by changing the values and colors with the provided sliders. Figure 7 shows different visualization of the same radar sample, obtained by changing the transfer function (affecting colors and opacity for different reflectivity values). 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.

4.2.2 Animation support

The visualization of a single radar volume is useful for static analysis, but the nature of the atmosphere is essentially dynamic. As the radar scans the atmosphere every 10 minutes, a collection of continuous 3D information is available for visualization. Animated series of CAPPI images are useful to visualize the evolution of a storm, or to analyze the generation process.
The utilization of volume rendering in animated series is referred as 4D data visualization, as it adds the time variable. In this case, the volumetric visualization of the radar data is used in combination with specific values of the transfer function, chosen to highlight specific parts of the volume. It is quite common to use these techniques to filter the more intense zones of the data, normally associated with the more dangerous rain or hail types.
In the Web GUI implemented, simple temporal navigational functionality has been added to enable users to access the next or previous volume in the sequence. One of the possible problems to allow a smooth visualization is the loading time, as the whole load and visualization

process has to be made for each individual volumetric dataset. Fortunately, the loading time is rather short and the possibility to create fully interactive 4D animations is open. Additional tests have shown that the limit is normally imposed by the size of the dataset and the bandwidth of the network, since the data must be downloaded from the server.

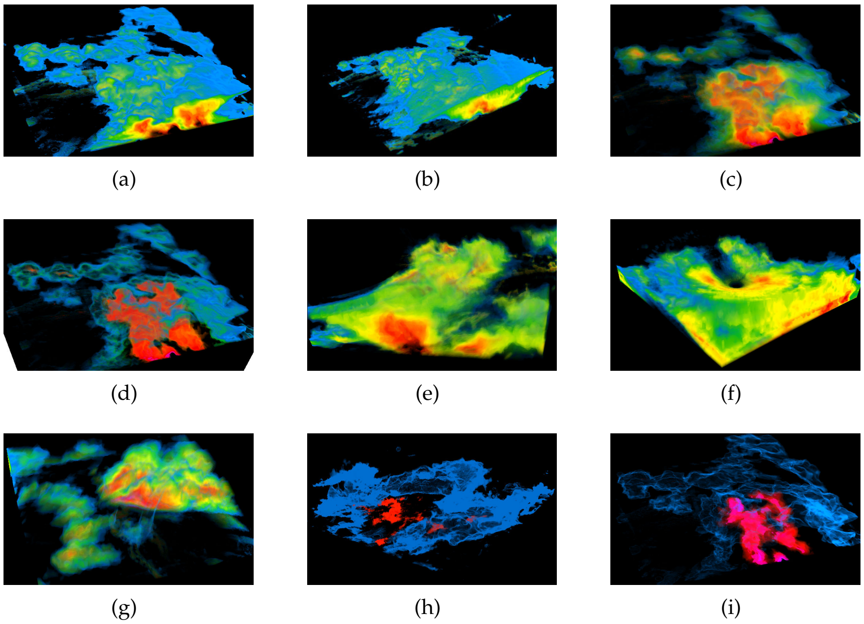


Fig. 7. Different weather radar volume renderings. Images (a) and (b) use typical color mapping for reflectivity scans (measured in decibels, dBZ). Images (c), (d), (e), (f), (g), (h) and (i) have been generated by varying the transfer function (color and transparency) and the window zoom and pan.

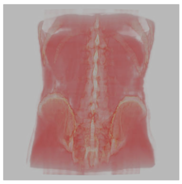
5. Results

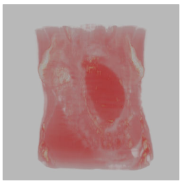
The proposed GPU implementation of the Volume Rendering technique presented in the previous sections has been tested with different settings and with different datasets. As the interactive and real-time results depend on both hardware and software, it is very important to begin with the platform specification 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.

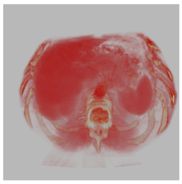
5.1 Hardware and Software configuration

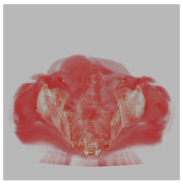
The tests for this chapter have been conducted using an Intel Quad Core Q8300 processor, 4GB of RAM and a GeForce GTX 460, Windows 7 PRO 64 bits (Service Pack 1) with the latest stable graphics drivers. Amongst all of the Web browsers with full implementation of WebGL

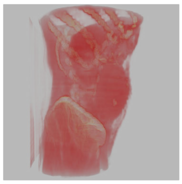
standard, we selected FireFox 6.0.2 for the test, other browsers are know to work with the implementation like Chrome 9.0.597.98[1] and Opera 11.50 labs (build 24661[2]).
It is worth while pointing out that both Chrome and Firefox, in default configuration, use Google's Angle library [3] to translate 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 `TRUE`. These changes basically disable Angle as the rendering back-end end validator of shaders, thus directly using the underlying native OpenGL support.

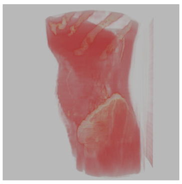

(a) Front View

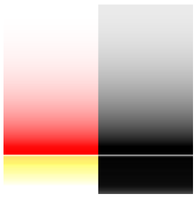

(b) Back View


(c) Top View


(d) Bottom View


(e) Left Side View


(f) Right Side View


(g) Used Transfer Function

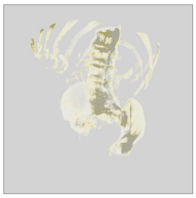

(h) Applying Other TF

Fig. 8. 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$\times$800 with 200 steps. 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.

A LightTPD Web server[4] 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.

[1] http://www.google.com/chrome
[2] http://snapshot.opera.com/labs/webgl/Opera_1150_24661_WebGL_en.exe
[3] http://code.google.com/p/angleproject
[4] http://www.lighttpd.net

5.2 Medical Dataset

Figure 8 shows some graphical output for the medical dataset introduced in the previous section. The 6 different axial views have been generated using 200 steps in the shaders implementation (800×800 canvas rendered in Firefox).

5.2.1 Dataset Resolution

This qualitative test was intended 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 4096^2, 2048^2, 1024^2m 512^2. The number of steps in the shaders were also varied, using 30, 80, 140 and 200 steps with *Firefox*. A selection of the results are shown shown in Figure **??**. If the number of steps is small the banding artifacts of the algorithm are noticiable, which is a problem. Some aproximations could be implemented to resolve this as show by Marques et al. (2009).

5.3 Medical Dataset in Portable Devices

The Mozilla Firefox Development Group has released a mobile version of the browser for ARM devices called Fennec[5]. We have tested it on 2 Android-based devices: Samsung Galaxy Tab[6] and Samsung Galaxy S smartphone[7]. 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 obtain quality results with the highest possible interactivity. The test using 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), this demonstrated the possibilty to render volume datasets on handheld devices. Further optimizations in the data or the implementation of the shaders, specifically oriented to such devices, might result in better overall performance. We set aside such issues for future work.

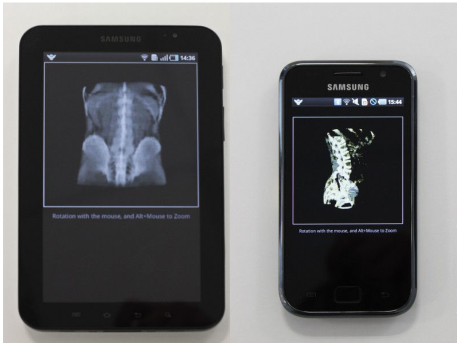

SAMSUNG
Rotation with the mouse, and Alt+Mouse to Zoom
SAMSUNG
Rotation with the mouse, and Alt+Mouse to Zoom

Fig. 10. A Samsung Galaxy Tab (left) and a Galaxy S Smartphone (right) volume - rendering medical datasets.

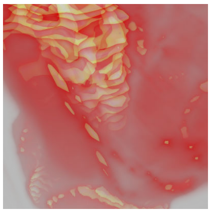
6. Contribution and Complexity Analysis

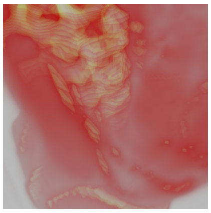
Our contribution is an implementation of a volume rendering system for the Web. The system is based on the Volume Ray-Casting algorithm with a complexity of $O(M * S)$, where M is the number of pixels to be drawn and S is the number of steps of the ray that traverses the volume. Since the algorithm is implemented in WebGL, its visualization speed is similar to

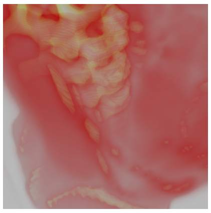
[5] http://www.mozilla.com/en-US/mobile

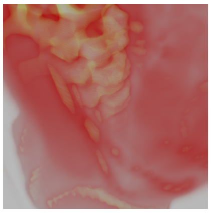
[6] http://galaxytab.samsungmobile.com/2010/index.html

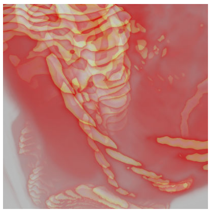
[7] http://galaxys.samsungmobile.com

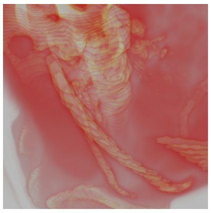

(a) 512px - 30 steps

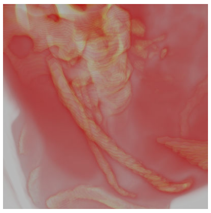

(b) 512px - 80 steps

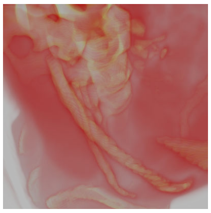

(c) 512px - 140 steps

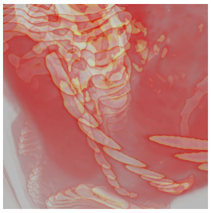

(d) 512px - 200 steps

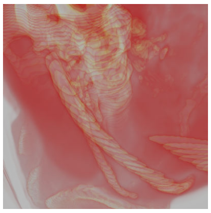

(e) 1024px - 30 steps

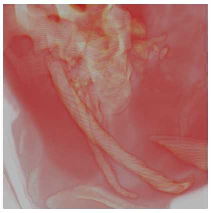

(f) 1024px - 80 steps

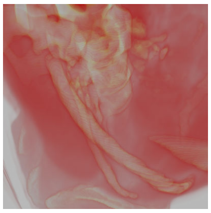

(g) 1024px - 140 steps

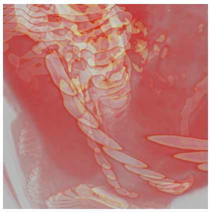

(h) 1024px - 200 steps

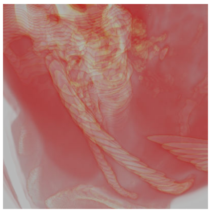

(i) 2048px - 30 steps

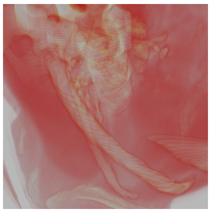

(j) 2048px - 80 steps


(k) 2048px - 140 steps


(l) 2048px - 200 steps


(m) 4096px - 30 steps


(n) 4096px - 80 steps


(o) 4096px - 140 steps

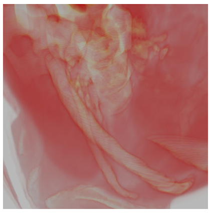

(p) 4096px - 200 steps

Fig. 9. Resolution qualitative test. Even with the dramatic reduction of the resolution, the volume render allows identification of the main structures.

native applications because it uses the same accelerated graphic pipeline. The original algorithm has been slightly modified to work with the input structures due to the lack of Volume Textures in WebGL. Therefore, our algorithm simulates the 3D data by using a 2D tilling map of the slices of the volume maintaining the tri-linear interpolation, so there is no real loss in quality because the interpolation is the same as that used in the GPU. Although 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 handling such fast events. This is due to the browsers been 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 expected because of the network performance, and the interpreted nature of JavaScript. Our implementation[8] does not show a significant 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. As a logical conclusion, more powerful 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.

7. Conclusions and future work

A medical application presented here illustrates the capabilities of complex volume rendering visualization in Web browsers. Although many performance improvements and optimizations are still needed, the material discussed here 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. The implementation of the algorithm in declarative languages as X3DOM is planned.
The scope of the present chapter does 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 a promising direction to explore. The minor optimizations applied to this work allow us to expect a mathematically-planned negotiation between speed performance and quality will be a promising research field. An additional goal for optimization time-varying datasets using videos instead of images as a render input, since video formats already minimize transmitted data by reducing temporal redundancy.
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 chapter lays the technical ground that would make the integration of surface render in volume-rendering (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

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

Council for Science and Technology –COLCIENCIAS–. Radar datasets were provided by the Basque Meteorology and Climatology Department.

8. References

Arens, S. & Domik, G. (2010). A survey of transfer functions suitable for volume rendering., *in* R. Westermann & G. L. Kindlmann (eds), *Volume Graphics*, Eurographics Association, pp. 77–83.
URL: *http://dblp.uni-trier.de/db/conf/vg/vg2010.html#ArensD10*

Behr, J. & Alexa, M. (2001). Volume visualization in vrml, *Proceedings of the sixth international conference on 3D Web technology*, ACM New York, NY, USA, pp. 23–27.

Behr, J., Eschler, P., Jung, Y. & Zöllner, M. (2009). X3dom: a dom-based html5/x3d integration model, *Proceedings of the 14th International Conference on 3D Web Technology*, ACM, pp. 127–135.

Blazona, B. & 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. & Ruiz, O. (2010). Extending marching cubes with adaptative methods to obtain more accurate iso-surfaces, *Computer Vision, Imaging and Computer Graphics. Theory and Applications International Joint Conference, VISIGRAPP 2009, Lisboa, Portugal, February 5-8, 2009. Revised Selected Papers*, Springer Berlin / Heidelberg, pp. 35–44.

Engel, K., Kraus, M. & Ertl, T. (2001). High-quality pre-integrated volume rendering using hardware-accelerated pixel shading, *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, HWWS '01, ACM, New York, NY, USA, pp. 9–16.
URL: *http://doi.acm.org/10.1145/383507.383515*

Ernvik, A. (2002). *3d visualization of weather radar data*, Master's thesis. LITH-ISY-EX-3252-2002.

Fogal, T. & Kruger, J. (2010). Tuvok, an Architecture for Large Scale Volume Rendering, *in* M. Dogget, S. Laine & W. Hunt (eds), *Proceedings of the 15th International Workshop on Vision, Modeling, and Visualization*, pp. 57–66.
URL: *http://www.sci.utah.edu/ tfogal/academic/tuvok/Fogal-Tuvok.pdf*

Goenetxea, J., Moreno, A., Unzueta, L., Galdós, A. & Segura, A. (2010). Interactive and stereoscopic hybrid 3d viewer of radar data with gesture recognition, *in* M. G. Romay, E. Corchado & M. T. García-Sebastián (eds), *HAIS (1)*, Vol. 6076 of *Lecture Notes in Computer Science*, Springer, pp. 213–220.

Hadwiger, M., Ljung, P., Salama, C. R. & Ropinski, T. (2009). Advanced illumination techniques for gpu-based volume raycasting, *ACM SIGGRAPH 2009 Courses*, ACM, pp. 1–166.

Hartley, R. & Zisserman, A. (2003). *Multiple View Geometry in Computer Vision*, second edn, Cambridge University Press, Cambridge, UK.
URL: *http://dx.doi.org/10.2277/0521540518*

Hibbard, W. & Santek, D. (1989). Interactivity is the key, *Proceedings of the 1989 Chapel Hill workshop on Volume visualization*, VVS '89, ACM, New York, NY, USA, pp. 39–43.
URL: *http://doi.acm.org/10.1145/329129.329356*

John, N., Aratow, M., Couch, J., Evestedt, D., Hudson, A., Polys, N., Puk, R., Ray, A., Victor, K. & 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.
URL: *http://www.sciencedirect.com/science/article/B6VCJ-4GNTFHN-1/2/038248c7a389ba900e10bef7249450da*

Kabongo, L., Macia, I. & Paloc, C. (2009). Development of a commercial cross-platform dicom viewer based on open source software, *in* P. H. U. Lemke, P. P. K. Inamura, P. P. K. Doi, P. P. M. W. Vannier, P. P. A. G. Farman & D. PhD (eds), *International Journal of Computer Assisted Radiology and Surgery; CARS 2009 Computer Assisted Radiology and Surgery Proceedings of the 23rd International Congress and Exhibition*, Vol. 4, International Foundation of Computer Assisted Radiology and Surgery, Springer, Berlin, Germany, pp. S29–S30.

Kajiya, J. T. & Von Herzen, B. P. (1984). Ray tracing volume densities, *SIGGRAPH Comput. Graph.* **18**: 165–174.
URL: *http://doi.acm.org/10.1145/964965.808594*

Kniss, J., Kindlmann, G. & Hansen, C. (2002). Multidimensional transfer functions for interactive volume rendering, *Visualization and Computer Graphics, IEEE Transactions on* **8**(3): 270 – 285.

Kniss, J., Premoze, S., Ikits, M., Lefohn, A., Hansen, C. & Praun, E. (2003). Gaussian transfer functions for multi-field volume visualization, *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, VIS '03, IEEE Computer Society, Washington, DC, USA, pp. 65–.
URL: *http://dx.doi.org/10.1109/VISUAL.2003.1250412*

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

Lacroute, P. & Levoy, M. (1994). Fast volume rendering using a shear-warp factorization of the viewing transformation, *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, ACM, New York, NY, USA, pp. 451–458.
URL: *http://doi.acm.org/10.1145/192161.192283*

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. & 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**: –.
URL: *http://www.sciencedirect.com/science/article/B6T5J-4XYB3W7-1/2/7068a72bb5a9b0c62a3819562ab176f5*

Marques, R., Santos, L. P., Leškovský, P. & Paloc, C. (2009). Gpu ray casting, *in* A. Coelho, A. P. Cláudio, F. Silva & A. Gomes (eds), *17Âž Encontro Português de Computaçao Gráfica*, En Anexo, Covilha, Portugal, pp. 83–91.

Marrin, C. (2011). *WebGL Specification*, Khronos WebGL Working Group.
URL: *http://www.khronos.org/webgl/*

Meißner, M., Huang, J., Bartz, D., Mueller, K. & Crawfis, R. (2000). A practical evaluation of popular volume rendering algorithms, *Proceedings of the 2000 IEEE symposium on Volume visualization*, Citeseer, pp. 81–90.

Meyer-Spradow, J., Ropinski, T., Mensmann, J. & Hinrichs, K. H. (2009). Voreen: A rapid-prototyping environment for ray-casting-based volume visualizations, *IEEE Com-*

puter Graphics and Applications (Applications Department) **29**(6): 6–13.
URL: *http://viscg.uni-muenster.de/publications/2009/MRMH09*

Pfister, H., Lorensen, B., Bajaj, C., Kindlmann, G., Schroeder, W., Avila, L., Raghu, K., Machiraju, R. & Lee, J. (2001). The transfer function bake-off, *Computer Graphics and Applications, IEEE* **21**(3): 16 –22.

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. & 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.
URL: *http://www.sciencedirect.com/science/article/B7CPS-4FJT8T8-7/2/1ec1f4078ec3bf48810583ea08227b32*

Riley, K., Song, Y., Kraus, M., Ebert, D. S. & Levit, J. J. (2006). Visualization of structured nonuniform grids, *IEEE Computer Graphics and Applications* **26**: 46–55.

Salonen, K., Järvinen, H. & Lindskog, M. (2003). Model for Doppler Radar Radial winds.

Segura, Á., Moreno, A., García, I., Aginako, N., Labayen, M., Posada, J., Aranda, J. A. & Andoin, R. G. D. (2009). Visual processing of geographic and environmental information in the basque country: Two basque case studies, *in* R. D. Amicis, R. Stojanovic & G. Conti (eds), *GeoSpatial Visual Analytics*, NATO Science for Peace and Security Series C: Environmental Security, Springer Netherlands, pp. 199–208.

Sundaram, V., Zhao, L., Song, C., Benes, B., Veeramacheneni, R. & Kristof, P. (2008). Real-time Data Delivery and Remote Visualization through Multi-layer Interfaces, *Grid Computing Environments Workshop, 2008. GCE'08*, pp. 1–10.

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

Wu, Y. & Qu, H. (2007). Interactive transfer function design based on editing direct volume rendered images, *Visualization and Computer Graphics, IEEE Transactions on* **13**(5): 1027 –1040.

Yoo, S., Key, J., Choi, K. & Jo, J. (2005). Web-Based Hybrid Visualization of Medical Images, *Lecture notes in computer science* **3568**: 376.

Zhou, J. & Takatsuka, M. (2009). Automatic transfer function generation using contour tree controlled residue flow model and color harmonics, *Visualization and Computer Graphics, IEEE Transactions on* **15**(6): 1481 –1488.