# Community-driven Extensions to the X3D Volume Rendering Component

Ander Arbelaiz
Vicomtech-IK4
Paseo Mikeletegi 57
Donostia / San Sebastián, Spain 20009
aarbelaiz@vicomtech.org

Aitor Moreno
Vicomtech-IK4
Paseo Mikeletegi 57
Donostia / San Sebastián, Spain 20009
amoreno@vicomtech.org

Luis Kabongo
Vicomtech-IK4
Paseo Mikeletegi 57
Donostia / San Sebastián, Spain 20009
Biodonostia Health Research Institute
Donostia / San Sebastián, Spain
lkabongo@vicomtech.org

Nicholas Polys
VirginiaTech
USA
npolys@vt.edu

Alejandro García-Alonso
University of the Basque Country
Paseo Manuel de Lardizabal 1
Donostia / San Sebastián, Spain 20018
alex.galonso@ehu.es

## ABSTRACT

Recent developments in Web-based volume rendering have gained recognition by Web users and professionals in several fields. The ISO-IEC Standard Extensible 3D (X3D) version 3.3 specifies the integration and visual styling of volumetric data for real-time interaction. The specification is an important milestone describing a framework for expressive presentation. However, it was written before the emergence of WebGL and the HTML5 platform. This paper describes our work to adapt the X3D Volume rendering nodes to the Web platform and to enhance their functionality based on feedback provided by the X3D and X3DOM open source communities. These include: a description of a new volume data node and an application of such node to create 4D volume rendering real time visualizations. We present functionalities that are currently not part of the standard: the edition of Transfer Functions, Multi Planar Reconstruction (MPR), intersection of the volume with 3D objects, clipping planes with volume data and control in the quality of the generated volume visualization. These additions should be considered for inclusion in future revisions of the X3D ISO volume rendering component.

## CCS CONCEPTS

**•Human-centered computing →Scientific visualization; Visualization systems and tools;** *Ubiquitous and mobile computing systems and tools;* **•Computing methodologies →***Rendering;*

## KEYWORDS

Extensible 3D (X3D), Volume Rendering, X3DOM, WebGL


*Web3D '17, Brisbane, QLD, Australia*

DOI: http://dx.doi.org/10.1145/3055624.3075945

**ACM Reference format:**
Ander Arbelaiz, Aitor Moreno, Luis Kabongo, Nicholas Polys, and Alejandro García-Alonso. 2017. Community-driven Extensions to the X3D Volume Rendering Component. In *Proceedings of Web3D '17, Brisbane, QLD, Australia, June 05-07, 2017,* 9 pages.
DOI: http://dx.doi.org/10.1145/3055624.3075945

## 1 INTRODUCTION

In recent years, with the adoption of WebGL in modern browsers, research and development in Web-based hardware-accelerated volume rendering has flourished. Agreements and conventions are required in order to sustain the exchange of volumetric content, the interoperability between Web and non-Web applications, and to maintain a cross-device support. Therefore we have focused in the ISO-IEC Standard of X3D (Web3DConsortium 2017a), which is the internationally-ratified specification allowing the interchange and delivery of declarative volume rendering scenes over the Web.

Nowadays, the Web has become the medium to expose rich multimedia content to the general public. For instance, in the medical field, some initiatives have already emerged that are currently working towards the transition of DICOM file format support for the Web platform (Cornerstone 2016). Using the Web as unified access point, volume rendering could become another tool for professionals and casual users alike. In order to reduce the gap between the expected capabilities of web-based tools and their desktop counterparts, we have engaged with the user community and assessed their requests and priorities to deliver the best functionalities within the current limitations of the Web platform.

The evolution of 3D graphics in the Web is slow in comparison to traditional desktop solutions. This is mainly due to two factors: *i)* security: applications developed by third-party must be sand-boxed within the browser context and *ii)* wider cross-device support: they tend to reach to a wider range of devices and GPUs. In despite of this, the Web ecosystem has benefited from a great surge in 3D graphics content since the introduction of the WebGL API. This has provided an opportunity to create communities of both users and developers alike. They help with both the adoption and with the

improvement of this technology into the future. Without a doubt, one of the communities that has been supporting the exchange of 3D content over the net is the one behind the X3D (Extensible 3D) ISO (Web3DConsortium 2017a).

In this paper we propose a volume data node definition to be considered for inclusion in the X3D Volume Rendering component to allow WebGL based ray-casting algorithms access the volumetric data in order to compute the visualization output. Furthermore, we will show the use of this node with one of X3D's reference implementations X3DOM (Behr et al. 2009) to create a 4D volume rendering visualization.

The paper is structured as follows: Section 2 introduces previous work and establishes a time-line of the current state of the X3DOM Volume Rendering implementation against the X3D specification. Section 3 presents our WebGL compatible volume data node proposal. Section 4 proposes community-driven features to be included in the standard and Section 5 finalizes with the conclusions and future work.

## 2 BACKGROUND

This section describes previous work to contextualize the proposed enhancements at Section 3 and Section 4. First, we introduce the X3D standard and the X3D Medical Working group. Then, we present the related work and contributions of web based volume rendering over time.

### 2.1 X3D and X3D Medical Working Group

The Extensible 3D (X3D) (Web3DConsortium 2017a) is an ISO-IEC ratified standard to represent and communicate 3D computer graphics. It is maintained and developed by the Web3D consortium. X3D is composed by a rich set of extensible components targeting different computer graphics areas (e.g CAD, Geospatial, Humanoid-animation, NURBS, etc.).

X3D has been evolving with each iteration. Due to its Component and Profile -based architecture, new additions can be added independently with ease and in collaboration with its corresponding working group. The Web3D Consortium's Medical Working Group (MWG) (Web3DConsortium 2017c) specifies and implements open standards to support cross-platform representation of medical visualization from a wide variety of image modalities and medical data exchange capabilities.

### 2.2 Related Work

X3D is actively being used in the scientific community. Researchers are publishing modification proposals and enhancements to concrete components. For example, for the Geospatial Component McCann et al. (2009) proposed enhancements to correct deficiencies at viewing data in a globally set context, to improve browser rendering for terrain data and to spread the adoption of the component.

Centered in the visualization of volume data, the Medical Working Group (MWG) has created the X3D Volume Rendering Component (Web3DConsortium 2017) and the MedX3D profile (John et al. 2007). During the standardization process of the Volume Rendering Component, Jung et al. (2008) presented a specialized endoscopic training simulation system based on an extended X3D, showing one of the multiple use case cases of application for volume rendering. Polys et al. (2011) evaluated the proposed X3D Volume Rendering Component for its suitability in the visualization of several volume image data types from different scientific fields. Furthermore, to sustain X3D as a reproducible volume scene declaration presentation interchange format, Polys and Wood (2012) have evaluated the volume component specification under several criteria: representation, implementation, interaction and interoperability/integration.

Applications of X3D based volume rendering are clear in the medicine field, from surgical planing to educational purposes. With the price reduction and availability of stereo head mounted displays (HMD), a new frontier of application has been opened for X3D. Towards immersive VR environments Behr et al. (2007) presented extensions to support different interactions and navigation tasks and Polys et al. (2013) described the challenges and capabilities of X3D in an immersive volume rendering implementation.

### 2.3 Web volume rendering evolution

Volume rendering is a computationally expensive rendering technique that can be implemented with different algorithms. Direct volume rendering (DVR) introduces several methods that take advantage of the GPU capabilities due to the fact that DVR algorithms present parallelizable features.

Initial approaches to bring volume rendering to the web ecosystem used indirect methods: *i)* polygonal surface data was extracted from the volumetric data or *ii)* a server-side remote rendering approach (Kaspar et al. 2013) were used.

With the arrival of WebGL to web browsers, Congote et al. (2011) presented an interactive volume rendering ray-casting (DVR) algorithm that enabled client-side rendering in the browser. This method has been later on revised by Noguera et al. (2012) focusing their work in the deployment of volume rendering techniques into mobile devices with the OpenGL ES 2.0 API, the same API in which WebGL 1.0 API is based on. Noguera and Jiménez (2016) have made a survey where performance comparisons of the ray-casting method against texture slicing methods are shown. Additionally, Movania et al. (2012a; 2012b; 2014; 2012) have also improved the multi-pass ray-casting algorithm with a single-pass version and compared it with a texture slice method. Wangkaoom et al. (2015) have made use of WebGL as a light rendering client by using a client and server based hybrid rendering solution.

X3DOM is a DOM-based implementation of X3D (Fraunhofer IGD 2014) that enables declarative X3D in the Web. Arbelaiz et al. (2016b) presented a volume rendering component for X3DOM based in Congote el al. (2012; 2011) approach. This component implementation offers X3D's volume visualization reproducible and declarative features and it has been the reference to obtain feedback from the community (X3DOM Community 2015a,b, 2016a,b, 2017).

Additional works have addressed different challenges related to the volume rendering component, presenting contributions and advances in the interaction and exploration of volumetric datasets. Yang et al. (2015) have contributed to X3DOM for weather data visualization in conjunction with terrain data. Arbelaiz et al. (2017) have explored the use of DICOM medical data exchange format in combination with X3DOM for medical volume visualization.

Figure 1 shows a time-line with the presented contributions related to WebGL based DVR (above) and some of the community requested enhancements (below). The time-line is divided in three periods: *i)* initial contributions of DVR algorithms applicable in the Web (orange area). *ii)* elapsed time in which initial developments of the X3DOM volume rendering component were made (green area). *iii)* contributions to web based DVR, including contributions to the X3DOM volume rendering component in response to community feedback. (blue area).

## 3 WEB COMPATIBLE X3D VOLUME RENDERING PROPOSAL

In this section we propose a new node to enable the interactive visualization of volumetric data in Web based GPU accelerated volume rendering algorithms.

First, we present the motivation behind the proposal of this node in the Web platform at Section 3.1. Then, at Section 3.2 we describe our proposed Web centered *ImageTextureAtlas* node. Finally, Section 3.3 shows the usability of this node in the Web with a specialized use case: 4D volume rendering.

### 3.1 WebGL-based volume rendering

As stated in the X3D volume rendering component, volume rendering requires data to be represented in a volumetric form (Web3DConsortium 2017). In GPU based volume rendering 3D textures are used to store volume data. Thus, it is defined in the X3D ISO that volume data shall be declared using X3D's *Texturing3D* component. Unfortunately, the current WebGL 1.0 API does not support this type of texture.

We have overcome this limitation of WebGL 1.0 with the method proposed by Congote et al. (2011). Using a 2D texture we can emulate a 3D texture by resampling the 2D texture and performing trilinear interpolation in the fragment shader at a pixel level.

With the upcoming WebGL 2.0 API, 3D textures will be supported by browsers, allowing to make better use of current GPU memory capabilities. Nevertheless, our proposed *ImageTextureAltas* will still be necessary for an ubiquitous volume rendering deployment, as this method is valid for both WebGL APIs.

### 3.2 ImageTextureAtlas | X3DTexture2DNode

Volumetric data, specially those obtained from a MRI or a CT scan, can be seen as a set of 2D image slices in an array. Our proposed *ImageTextureAtlas* node allows to represent the 3D volume data by composing all the 2D slices into a single 2D texture (Congote et al. 2011). Instead of adding a Z dimension to the texture, we arrange all 2D slices into one image with a matrix configuration. Listing 1 shows the basic X3D definition of the proposed node.

**Listing 1: X3D definition for the *ImageTextureAtlas***

```
ImageTextureAtlas : X3DTexture2DNode
 SFNode [in,out] metadata NULL [X3DMetadataObject]
 SFBool [] repeatS TRUE
 SFBool [] repeatT TRUE
 SFNode [] textureProperties NULL [
      TextureProperties]
 MFString [in,out] url [] [URI]
 SFInt32 [] numberOfSlices 0 [0,∞)
 SFInt32 [] slicesOverX 0 [0,∞)
 SFInt32 [] slicesOverY 0 [0,∞)
 SFBool [] hideChildren TRUE
}
```

Figure 2 shows an atlas of slices for the proposed *ImageTextureAtlas* node. Listing 2 definition represents how *X3DVolumeData* derived nodes, such as the *VolumeData*, should allow a 2D texture input to accept the *ImageTextureAtlas* as a parameter.

**Listing 2: Allow *ImageTextureAtlas* as an input to *X3DVolumeData* nodes, definition example for the *VolumeData* node**

```
VolumeData : X3DVolumeDataNode {
 SFVec3f [in,out] dimensions 1 1 1 (0,∞)
 SFNode [in,out] metadata NULL [X3DMetadataObject]
 SFNode [in,out] renderStyle NULL [
      X3DVolumeRenderStyleNode]
 SFNode [in,out] voxels NULL [X3DTexture2DNode,
      X3DTexture3DNode]
 SFVec3f [] bboxCenter 0 0 0 (−∞,∞)
 SFVec3f [] bboxSize −1 −1 −1 [0,∞) or −1 −1 −1
}
```

The surface normals of the volume data are required to apply the volume rendering styles defined in the X3D ISO Volume Rendering Component (Web3DConsortium 2017). Surface normals are specified to be provided as an additional 3D texture. As stated before, WebGL 1.0 does not support this texture type, so we have to make use of the *ImageTextureAtlas* again. Arbelaiz et al. (2016b) have extended the *ImageTextureAtlas* approach to adapt its use to the other nodes described in the current X3D v3.3 ISO specification (Web3DConsortium 2017). In this manner, we can proceed to apply illustrative and non-photorealistic styles to the volume visualization.

The surface normals are approximated with the gradient computation of the volume data. The gradient computation outputs a three component vector for each voxel in the volume. Each component matches with the derivative of the volume data on each axis direction. Using the same approach as before, we can create an *ImageTextureAtlas* using the color channels of the 2D texture to store the vector information. We encode the gradient vector for each pixel in the atlas in the RGB color channels R: X, G: Y, B: Z.

As described before, we propose to update the X3D specification to support the *ImageTextureAtlas* as a valid field to declare the surface normals of a volume node. The same proposed node in Listing 1 is valid for the surface normals input. Listing 3 shows an example declaration of the gradient data.

**Listing 3: An *ImageTextureAtlas* declaration for the surface normals data**

```
<ImageTextureAtlas containerField="surfaceNormals"
      url="gradient.png" slicesOverX="8"
     slicesOverY="8"></ImageTextureAtlas>
```

The *containerField* and *URL* attributes are the only modifications required for the *ImageTextureAtlas* declaration. The *containerField* attribute is used to target the volume data *voxels* field in a *X3DVolumeDataNode* or gradient data *surfaceNormlas* field in

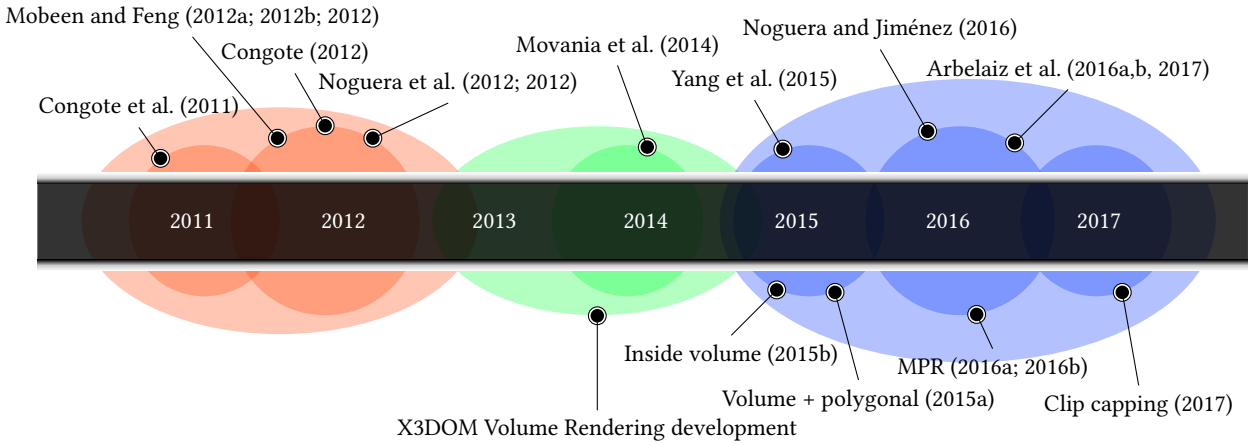


**Figure 1: Time-line of contributions to WebGL based volume rendering and community feedback**

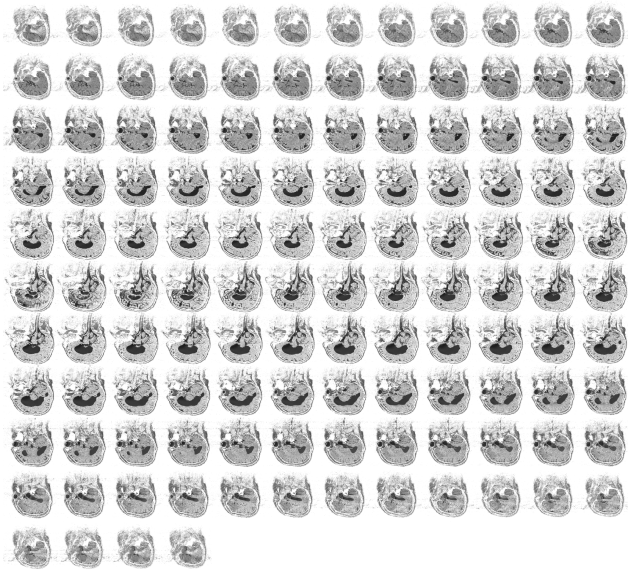

**Figure 2: A 2D image representing an *ImageTextureAtlas* of the *MRI ventricles* dataset (Volvis 2017) (color inverted and contrast enhanced) as a set of 2D slices tiled into a matrix configuration**

a *X3DVolumeStyleNode*. This also requires, for the *X3DVolumeStyleNode* type nodes with a *surfaceNormals* attribute, to accept a *X3DTexture2DNode* as an input argument like it is shown in Listing 2 for the *VolumeData* node.

The *SegmentedVolumeData* node allows the user to discern regions of the volume and apply different rendering styles to each one. The segmented region must be labeled per voxel. Consequently, the use of an *ImageTextureAtlas* is mandatory in order to make the *SegmentedVolumeData* compatible with WebGL 1.0.

Figure 3 shows the difference between a volume data slice (R color channel), a segmented data slice (R color channel) and the surface normals slice (RGB color channels).

The amount of volume data that can be stored in a 2D *ImageTextureAtlas* is limited by the GPU's 2D texture size limit. However, some strategies can be followed to allow the visualization of bigger datasets. Noguera and Jiménez (2012) used the ignored color channels (Green Blue and Alpha color channels) of a volume data atlas to store larger datasets.

To make use of the multiple color channels from the 2D texture the *ImageTextureAtlas* requires new fields. Listing 4 shows the addition of *channels* and *sortOrder* fields. The *channels* field defines in which color channel of the texture volume data is being stored. The default behavior is to store the volume data in the Red color channel by specifying the "R" value. When, a larger volume is required to be converted into an *ImageTextureAtlas* additional channels can be specified with "R", "G", "B", "A" characters. For instance, to store up to three times more data. A "RGBA" value in the *channel* field will indicate that all color channels of the texture are being used to store the data. Once multiple color channels are defined, the order in which 2D slices are tiled into the atlas must be set in the *sortOrder* field. The default behavior is to tile the 2D slices that represent the Z axis of the volume data in *"ROW"* order. For each slice of the *ImageTextureAtlas* the next slice in the Z axis is the contiguous slice in the row of the matrix of slices. When the *sortOrder* is set to *"CHANNEL"* the next slice of the atlas in the Z axis direction is stored in the the contiguous color channel.

**Listing 4: X3D definition for the *ImageTextureAtlas* with multiple color channels**

```
X3DVolumeDataNode : X3DTexture2DNode
 SFNode [in,out] metadata NULL [X3DMetadataObject]
 SFBool [] repeatS TRUE
 SFBool [] repeatT TRUE
 SFNode [] textureProperties NULL [
     TextureProperties]
 MFString [in,out] url [] [URI]
 SFInt32 [] numberOfSlices 0 [0,∞)
 SFInt32 [] slicesOverX 0 [0,∞)
 SFInt32 [] slicesOverY 0 [0,∞)
 SFBool [] hideChildren TRUE
 SFString [] channels "R"
```

(a) Volume data (b) Segmented data (c) Surface normals data

**Figure 3: Slice types of the *MRI ventricles* dataset (Volvis 2017) to be converted into an *ImageTextureAtlas*: a) voxel data slice, b) segmented data slice and c) gradient data slice**

```
  SFString [] sortOrder "ROW" ["ROW", "CHANNEL"]
}
```

Textures provide a mechanism to input data to the GPU updating directly the visualization. In the Web platform we can take advantage of the 2D canvas API to create or modify images that can be copied to the GPU. This is of special interest to reflect dynamic changes in the data and, for instance, to perform the construction of the atlas in the browser. Listing 5 shows an *ImageTextureAtlas* node declaration with a canvas. Note that the *hideChildren* attribute hides the atlas from the user, but makes it available to be modified with JavaScript.

**Listing 5: *ImageTextureAtlas* declaration linked with the 2D HTML5 canvas API**

```
<ImageTextureAtlas containerField="voxels" url=""
     slicesOverX="8" slicesOverY="8"
     hideChildren="true">
   <canvas id="v" style="width:2048px; height:2048px;">
   </canvas>
</ImageTextureAtlas>
```

## 3.3 4D volume rendering

There are a number of use cases when the volume visualization evolves over time (4D volume rendering). For example, in the medical field, 4D ultrasound imaging and MRI capture are already possible, or in the geosciences field with simulations in longer period of times of the evolution of a system (Ho and Jern 2008). Using our proposed *ImageTextureAtlas* and the capabilities of current modern browsers, 4D visualizations can be defined.

Our approach exploits the native video reproduction and 2D canvas API features of HTML5 modern browsers. Figure 4 summarizes the architecture of this approach: We create an atlas for each time step of the 4D volume data. Then, all the atlases are converted and encoded into a video which will be playing in a loop. This atlas sequence video is being played in the background hidden to the user. Listing 6 shows the definition of the video declaration in HTML5.

**Listing 6: HTML5 video declaration for an *ImageTextureAtlas***

```
<video autoplay loop style="width:2048px;
     height:2048px;display:none">
   <source src="atlas_video.mp4" type="video/mp4">
   </source>
</video>
```

With JavaScript we link the hidden video with a dynamic *ImageTextureAtlas* by defining a canvas element of the same size of the video and updating its content through the canvas 2D API with a fixed timer. The *ImageTextureAtlas* follows the sames declaration pattern as shown in Listing 5. The underlying volume rendering implementation of X3DOM will update the texture data at the GPU and visualizing the changes in real-time.

There are some considerations to take into account like the potential loss of accuracy due to the employed video encoding or the refresh time of the texture data. Nevertheless, this approach demonstrates the current capabilities of the Web technology stack and the proposed Web centered *ImageTextureAtlas*.

# 4 COMMUNITY DRIVEN ENHANCEMENTS TO X3D

The X3D volume rendering component has been unchanged for a period of time since the release of version v3.3 (Web3DConsortium 2017). This section presents some additions and enhancements which are centered in issues and problems we have received from Web users at the X3D and X3DOM communities. We also add suggestions of how these enhancements could be added to the current X3D volume rendering component specification.

## 4.1 Transfer function

A transfer function (TF) allows to filter and enhance intensity ranges in the volume data by mapping each volume value to a given color and opacity. This enables the visualization through some layers of specific densities and the increment of the opacity in other layers. For that purpose, a texture is used as a look-up table. It is the most used method to add color to the volume visualization and the default rendering style node in X3D.

Current X3D specification contemplates both 3D and 2D textures as valid input fields for the *OpacityMapVolumeStyle*, being the default behavior the use of 2D TFs. The Web platform can support natively 2D textures. However, a connection to the 2D texture that

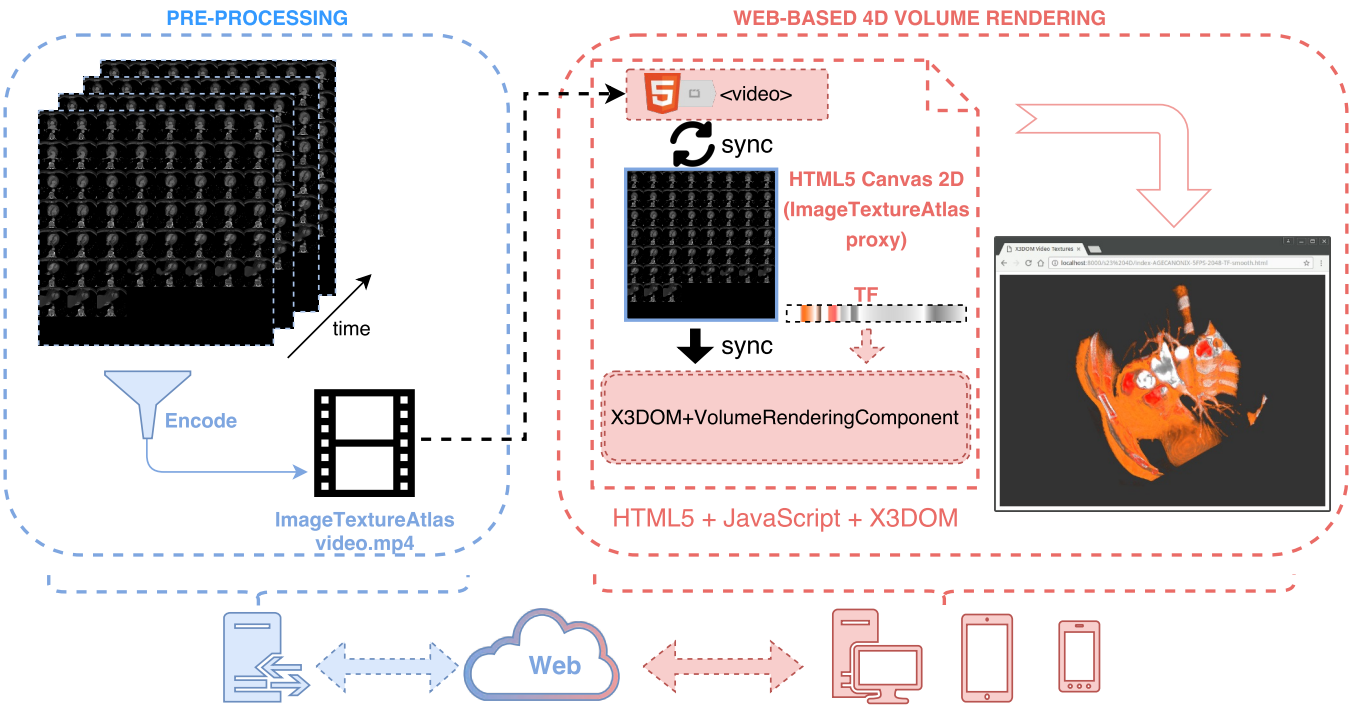


**Figure 4: Scheme of the proposed architecture for a 4D volumetric visualization with the *AGECANONIX* dataset (OsiriX 2017)**

directly influences the TF is required in order to allow the creation of tools such as a native HTML5 transfer function editor. Thus, as stated in Section 3.2 and Listing 5 we have to make use of the canvas 2D API in order to modify dynamically the TF.

## 4.2 Multi-planar reconstruction (MPR)

The Multi-Planar Reconstruction (MPR) is a wide spread rendering technique used in the medical field. Essentially, it enables the user to define arbitrary planes through the data. The rendering algorithm resamples the volume data to reconstruct the desired plane. Usually, the following planes will be defined: Axial, Sagital, Coronal and Oblique.

This technique is less memory expensive in comparison to a full volume rendering visualization. Only the volume data in the neighborhood of the defined plane is used in the computation. Less powerful GPU devices, like mobile devices, will handle easier this type of rendering.

An additional interesting enhancement to MPR is the ability to add a transfer function (TF) in order to illustrate or filter regions in the reconstructed planes. In fact, this is necessary in many domains, for example, in physics simulation it is used to correlate the visual output with the obtained results. Listing 7 shows the proposed X3D MPR volume rendering style definition.

**Listing 7: X3D definition proposal for the MPRVolumeStyle node**

```
MPRVolumeStyle : X3DVolumeRenderStyleNode {
 SFBool [in,out] enabled TRUE
 SFNode [in,out] metadata NULL [X3DMetadataObject]
 SFNode [in,out] transferFunction NULL [
     X3DTexture2DNode,X3DTexture3DNode]
 MFNode [in,out] plane NULL [MPRPlane]
}
```

The proposed MPR node (shown in Listing 7) defines a *transferFunction* attribute with the same functionality as the one already defined in the *OpacityMapVolumeStyle*. This node also defines a *planes* attribute to allow the user to declare arbitrary planes. Listing 8 presents an arbitrary plane definition for the *MPRVolumeStyle* node. The *normal* attribute defines the normal vector of the plane, while the *pos* attribute establishes the position of the plane from the origin of the volume in the *normal* direction.

**Listing 8: X3D definition proposal for an arbitrary MPR volume plane**

```
MPRPlane : X3DNode {
 SFBool [in,out] enabled TRUE
 SFNode [in,out] metadata NULL [X3DMetadataObject]
 SFVec3 [in,out] normal 0 0 1
 SFFloat [in,out] pos 0.0 [0,1]
}
```

## 4.3 Quality control

The X3D ISO specification is an abstract declaration unaware of the underlaying implementations. But, we consider that to deploy X3D scenes in as many devices as possible, some performance or implementation aware requisites should be considered.

A Web-based ubiquitous volume rendering implementation allows a volumetric scene to be deployed in a wide range of devices. From an ubiquitous deployment perspective, a scene with multiple rendering styles may be plausible for a desktop PC with a dedicated GPU, but could also be too computationally expensive for a mobile device. Not all devices have the same GPU features and computational power. Consequently, this situation can make volume rendering unavailable to some of these devices. In order to allow one X3D volume rendering scene to be deployed into different devices we suggest to add an output quality control mechanism. This mechanism should focus on the target devices and it should regulate the amount of computation to be performed by the device via the concrete implementation.

A *quality* field could be defined with a qualitative value (provided as a profile) or with a quantitative value (provided as ranged numerical scalar value). Listing 9 shows an additional qualitative field for the *X3DVolumeDataNode*, where the *quality* field accepts three levels of quality *"LOW", "MEDIUM", "HIGH"*. In this manner, each possible implementation of the X3D standard could adequate the amount of computation to be performed at different levels.

**Listing 9: X3DVolumeDataNode definition with output quality control**

```
X3DVolumeDataNode : X3DChildNode, X3DBoundedObject
 SFVec3f [in,out] dimensions 1 1 1 (0,∞)
 SFNode [in,out] metadata NULL [X3DMetadataObject]
 SFVec3f [] bboxCenter 0 0 0 (−∞,∞)
 SFVec3f [] bboxSize −1 −1 −1 [0,∞) or −1 −1 −1
 SFString [in,out] quality "HIGH" ["LOW", "MEDIUM", "HIGH"]
}
```

Another alternative is shown at Listing 10, where the *quality* field shows a quantitative value. For implementations where the quality of the output can not be regulated, always a *"HIGH"* value or a *"1.0"* factor shall be expected.

**Listing 10: X3DVolumeDataNode definition with output quality control**

```
X3DVolumeDataNode : X3DChildNode, X3DBoundedObject
 SFVec3f [in,out] dimensions 1 1 1 (0,∞)
 SFNode [in,out] metadata NULL [X3DMetadataObject]
 SFVec3f [] bboxCenter 0 0 0 (−∞,∞)
 SFVec3f [] bboxSize −1 −1 −1 [0,∞) or −1 −1 −1
 SFFloat [in,out] quality 1.0 [0,1]
}
```

## 4.4 Inside volume data exploration

Some regions of the volume data can be occluded even after filtering the data with a transfer function (TF). Consequently, the ability to explore the inside of the volume has been requested in the X3DOM Community (2015b). As a response, we have already provided this feature in X3DOM (Arbelaiz et al. 2017).

Allowing to place the viewers virtual camera inside the volume provides a new perspective to analyze the data. The current X3D ISO does not define the behavior of the volume rendering algorithms in relation to the location of the camera inside the volume. We suggest to define a new attribute to enable or disable this behavior. In one hand, all X3D conformance implementations will follow the same behavior. In the other hand, to explicitly enable or disable this functionality will allow to avoid the extra computation required. Conditional branches execution can be expensive in GPU's and this type of extra operations can be avoided if this feature is disabled.

Our proposal is to declare a new boolean attribute *allowViewPointInside* to all nodes inherited by the *X3DVolumeDataNode*. Listing 11 shows the definition of the *X3DVolumeDataNode* with the proposed attribute.

**Listing 11: *X3DVolumeDataNode* definition allowing the user to explore from inside the volume**

```
X3DVolumeDataNode : X3DChildNode, X3DBoundedObject
 SFVec3f [in,out] dimensions 1 1 1 (0,∞)
 SFNode [in,out] metadata NULL [X3DMetadataObject]
 SFVec3f [] bboxCenter 0 0 0 (−∞,∞)
 SFVec3f [] bboxSize −1 −1 −1 [0,∞) or −1 −1 −1
 SFBool [in,out] allowViewPointInside TRUE
}
```

For the *VolumeData* node the default definition will be the following:

```
<VolumeData allowViewpointInside='true'></
    VolumeData>
```

By default we suggest to allow the inspection of the volume data and if the user does not require to do so, it can be explicitly disabled.

## 4.5 Intersection of 3D polygonal meshes

A hybrid rendering of 3D polygonal meshes in conjunction with a volume object can be of great interest. It opens new use cases for several scientific fields. As a reference, Yang et al. (2015) has already presented a GIS use case for the visualization of volumetric weather radar data and a polygonal terrain with X3DOM.

The current X3D volume rendering component does not specify nor describe any polygonal and volume data intersection behavior. From a technical point of view, it is already feasible to perform such hybrid rendering in a Web context (Arbelaiz et al. 2016a).

We suggest to add the depth information of the previously rendered meshes in the scene to enable the coexistence of polygonal meshes and volume data together. In this way, before rendering the volumetric data, the rendering algorithms can measure any intersection with polygonal surfaces and avoid any occluded computation while also performing the blending with the polygonal surface.

To add the scene depth information, nodes which inherit from *X3DVolumeDataNode* (*VolumeData*, *SegmentedVolumeData*, *IsoSurfaceVolumeData*) should allow to access the depth of the scene. Listing 12 shows the proposed addition for the basic *VolumeData* node.

**Listing 12: *VolumeData* definition with access to the scene depth information**

```
VolumeData : X3DVolumeDataNode {
 SFVec3f [in,out] dimensions 1 1 1 (0,∞)
 SFNode [in,out] metadata NULL [X3DMetadataObject]
 SFNode [in,out] renderStyle NULL [
     X3DVolumeRenderStyleNode]
 SFNode [in,out] voxels NULL [X3DTexture3DNode]
 SFNode [in,out] sceneDepth NULL [X3DTexture2DNode]
```

```
  SFVec3f [] bboxCenter 0 0 0 (−∞,∞)
  SFVec3f [] bboxSize −1 −1 −1 [0,∞) or −1 −1 −1
}
```

### 4.6 Clipping

Clipping planes are supported in X3D since the release of X3D v3.2 (Web3DConsortium 2017b). In the specification, the behavior of clipping is shown with 2D geometry examples. The *X3DVolumeData* derived nodes act as *Shape* nodes that handle volumetric data instead of geometry data. This implies that the current specification of clipping planes should also be applied to volumes.

In a complex scenario, where both geometry and volume data are clipped we should expect different behaviors: clipped volume data by definition will always contain data inside the clipped region, whereas a clipped polygonal object could be empty. A medical application using clipping planes will probably expect intersected 3D geometry to be capped (X3DOM Community 2017). The specification should consider the possibility to cap clipped polygonal surfaces, otherwise this behavior would remain undefined and therefore, any X3D implementation would provide a custom and non-standard implementation.

## 5 CONCLUSIONS AND FUTURE WORK

In this paper we have presented a working proposal of a Web compatible 3D volumetric data input node, *ImageTextureAtlas*. Furthermore, using this node we have showcased how to perform a Web based 4D volume rendering real-time visualization.

We are aware that Web-based volume rendering can not compete with desktop or server based implementations in terms of performance. However, this is not true in terms of interoperability, usability and accessibility.

The Web platform has experimented a rapid evolution, improving its technological offering to cope with the increasing demand for richer multimedia and graphics content. The combination of WebGL, JavaScript, HTML5 and HTML Canvas 2D with the X3D standard by the Web3D Consortium have offered us the base tools to expand the reach of volumetric real-time content to a wider audience.

Taking into account the community feedback we have proposed how some of their requirements could be included in the X3D volume rendering component specification.

In the future we would like to add the proposed features in the X3DOM Framework, acting as reference implementation to support future inclusions of these features in the X3D volume rendering component.

## REFERENCES


Ander Arbelaiz, Aitor Moreno, and Luis Kabongo. 2016a. Deployment of Volume Rendering Interactive Visualizations in Web Platforms With Intersected 3D Geometry. In *Spanish Computer Graphics Conference (CEIG)*, Alejandro Garcia-Alonso and Belen Masia (Eds.). The Eurographics Association. DOI: http://dx.doi.org/10.2312/ceig.20161312

Ander Arbelaiz, Aitor Moreno, Luis Kabongo, and Alejandro García-Alonso. 2016b. X3DOM volume rendering component for web content developers. *Multimedia Tools and Applications* (2016), 1–30. DOI: http://dx.doi.org/10.1007/s11042-016-3743-1

Ander Arbelaiz, Aitor Moreno, Luis Kabongo, and Alejandro García-Alonso. 2017. *Volume Visualization Tools for Medical Applications in Ubiquitous Platforms*. Springer International Publishing, Cham, 443–450. DOI: http://dx.doi.org/10.1007/978-3-319-49655-9_54

Johannes Behr, Patrick Dähne, Yvonne Jung, and Sabine Webel. 2007. Beyond the web browser-x3d and immersive vr. In *IEEE Virtual Reality 2007: Symposium on 3D User Interfaces (3DUI)*, Vol. 2007. Fraunhofer IGD.

Johannes Behr, Peter Eschler, Yvonne Jung, and Michael Zöllner. 2009. X3DOM: A DOM-based HTML5/X3D Integration Model. In *Proceedings of the 14th International Conference on 3D Web Technology (Web3D '09)*. ACM, New York, NY, USA, 127–135. DOI: http://dx.doi.org/10.1145/1559764.1559784

John Congote. 2012. MEDX3DOM: MEDX3D for X3DOM. In *Proceedings of the 17th International Conference on 3D Web Technology (Web3D '12)*. ACM, New York, NY, USA, 179–179. DOI: http://dx.doi.org/10.1145/2338714.2338746

John Congote, Alvaro Segura, Luis Kabongo, Aitor Moreno, Jorge Posada, and Oscar Ruiz. 2011. Interactive Visualization of Volumetric Data with WebGL in Real-time. In *Proceedings of the 16th International Conference on 3D Web Technology (Web3D '11)*. ACM, New York, NY, USA, 137–146. DOI: http://dx.doi.org/10.1145/2010425.2010449

Cornerstone. 2016. JavaScript library to display interactive medical images including but not limited to DICOM. (2016). https://github.com/chafey/cornerstone

Fraunhofer IGD. 2014. X3DOM: Open-source framework and runtime for 3D graphics on the Web. http://www.x3dom.org. (2014). http://www.x3dom.org

Quan Ho and Mikael Jern. 2008. Interacting with 4D oceanographic volume data using GeoAnalytics tools. *National Center for Visual Analytics NCVA* (2008).

NW John, M Aratow, J Couch, D Evestedt, AD Hudson, N Polys, RF Puk, A Ray, K Victor, and Q Wang. 2007. MedX3D: standards enabled desktop medical 3D. *Studies in health technology and informatics* 132 (2007), 189–194.

Yvonne Jung, Ruth Recker, Manuel Olbrich, and Ulrich Bockholt. 2008. Using X3D for Medical Training Simulations. In *Proceedings of the 13th International Symposium on 3D Web Technology (Web3D '08)*. ACM, New York, NY, USA, 43–51. DOI: http://dx.doi.org/10.1145/1394209.1394221

Mathias Kaspar, Nigel M Parsad, and Jonathan C Silverstein. 2013. An optimized web-based approach for collaborative stereoscopic medical visualization. *Journal of the American Medical Informatics Association* 20, 3 (2013), 535–543.

Michael McCann, Richard Puk, Alan Hudson, Rex Melton, and Don Brutzman. 2009. Proposed Enhancements to the X3D Geospatial Component. In *International Conference on 3D Web Technology*, Dieter W. Fellner, Alexei Sourin, Johannes Behr, and Krzysztof Walczak (Eds.). The Eurographics Association. DOI: http://dx.doi.org/10.1145/1559764.1559788

M. M. Mobeen and L. Feng. 2012a. High-Performance Volume Rendering on the Ubiquitous WebGL Platform. In *2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems*. 381–388. DOI: http://dx.doi.org/10.1109/HPCC.2012.58

M. M. Mobeen and Lin Feng. 2012b. Ubiquitous medical volume rendering on mobile devices. In *International Conference on Information Society (i-Society 2012)*. 93–98.

Muhammad Mobeen Movania, Wei Ming Chiew, and Feng Lin. 2014. On-Site Volume Rendering with GPU-Enabled Devices. *Wireless Personal Communications* 76, 4 (2014), 795–812. DOI: http://dx.doi.org/10.1007/s11277-013-1354-y

Muhammad Mobeen Movania and Feng Lin. 2012. Mobile visualization of biomedical volume datasets. *J. Internet. Technol. Secured Trans* 1, 2 (2012), 52–60.

José M Noguera and Juan-Roberto Jiménez. 2012. Visualization of very large 3D volumes on mobile devices and WebGL. *WSCG Communication Proceedings* (2012), 105–112.

J. M. Noguera and J. R. Jiménez. 2016. Mobile Volume Rendering: Past, Present and Future. *IEEE Transactions on Visualization and Computer Graphics* 22, 2 (Feb 2016), 1164–1178. DOI: http://dx.doi.org/10.1109/TVCG.2015.2430343

José M Noguera, Juan-Roberto Jiménez, Carlos J Ogáyar, and Rafael Jesús Segura. 2012. Volume Rendering Strategies on Mobile Devices.. In *GRAPP/IVAPP*. 447–452.

OsiriX. 2017. DICOM Image Library. (2017). http://www.osirix-viewer.com/resources/dicom-image-library/

N Polys and Andrew Wood. 2012. New platforms for health hypermedia. *Issues in Information Systems* 13, 1 (2012), 40–50.

Nicholas Polys, Andrew D Wood, and Patrick Shinpaugh. 2011. Cross-platform Presentation of Interactive Volumetric Imagery.

Nicholas F Polys, Sebastian Ullrich, Daniel Evestedt, Andrew D Wood, and Michael Aratow. 2013. A fresh look at immersive Volume Rendering: Challenges and capabilities. In *IEEE VR Workshop on Immersive Volume Rendering, Orlando*.

Volvis. 2017. Volumetric dataset archive. (2017). http://volvis.org

K. Wangkaoom, P. Ratanaworabhan, and S. S. Thongvigitmanee. 2015. High-quality web-based volume rendering in real-time. In *2015 12th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*. 1–6. DOI: http://dx.doi.org/10.1109/ECTICon.2015.7207091

Web3DConsortium. 2017. Extensible 3D (X3D) Volume rendering component: ISO/IEC 19775-1. (2017). http://www.web3d.org/files/specifications/19775-1/V3.3/Part01/components/volume.html

Web3DConsortium. 2017a. Extensible 3D (X3D) specifications. (2017). http://www.web3d.org/x3d/specifications/

Web3DConsortium. 2017b. Extensible 3D (X3D) v3.2. (2017). http://www.web3d.org/standards/version/V3.2

Web3DConsortium. 2017c. Web3DConsortium Medical Working Group (MWG). (2017). http://www.web3d.org/working-groups/medical

X3DOM Community. 2015a. Johannes SchrÃűder-Schetelig - Rendering of volumetric and polygonal data together. (2015). https://sourceforge.net/p/x3dom/mailman/x3dom-users/thread/CAC7R8D5gFBTeKMk36Pb0ayY_g0qE0hHpok2ioO5C339mr9Akdg@mail.gmail.com

X3DOM Community. 2015b. onehalfmv2 - Moving inside a volume. (2015). https://github.com/x3dom/x3dom/issues/537

X3DOM Community. 2016a. Paul - MPR multiple arbitrary planes. (2016). https://sourceforge.net/p/x3dom/mailman/x3dom-users/thread/d9cd0469-497f-03ac-fe72-b6909b2a9b7f%40web.de

X3DOM Community. 2016b. pgruenbacher-TSUS - MPR not include tranfer function. (2016). https://github.com/x3dom/x3dom/issues/613

X3DOM Community. 2017. PCH3DPrintLab - Section Caps for Clipping Planes. (2017). https://github.com/x3dom/x3dom/issues/718

Yeonsoo Yang, Ankit Sharma, and Armand Girier. 2015. Volumetric Texture Data Compression Scheme for Transmission. In *Proceedings of the 20th International Conference on 3D Web Technology (Web3D '15)*. ACM, New York, NY, USA, 65–68. DOI: http://dx.doi.org/10.1145/2775292.2775323