Extensible 3D (X3D)
Part 1: Architecture and base components

Volume Rendering Component (Extension Proposal)

1. Introduction

1.1 Name

The name of this component is "VolumeRendering". This name shall be used when referring to this component in the COMPONENT statement (see ISO FDIS 19775-1:200x 7.2.5.4 Component statement).

1.2 Overview

This clause describes the Volume Rendering component of this part of ISO/IEC 19775. This includes how volumetric data sets are are specified and how they are rendered . Table 1 provides links to the major topics in this clause.

Table 1 — Topics

2 Concepts

2.1 Overview

Volume Rendering is an alternate form of visual data representation compared to the traditional polygonal form used in the rest of this specification. Where polygons represent an infinitely thin plane, volume data represent a three dimensional block of data. When polygonal data representing a volume in space is sliced, such as with a clipping plane, there is empty space. In the same situation volumetric data will show the internals of that volume.

There are many different techniques for rendering volumetric data - plane slicing, use of real time shaders, ray tracing etc. This component does not define the technique used to render the data, only the type of visual output needed. For example, a simple set of opacity data has one representation, while segmented data sets have another. In order to implement some of the higher complexity representations, the implementor may need to use a more complex technique than the simpler representations, though it is not required. Each of the rendering nodes will represent the visual output required, not the technique used to implement it.

2.2 Representing Volumetric Data

2.2.1 Registration and Scaling

Volumetric data represents volume information that typically comes from the real world: for example human body scans or finite element analysis of an engine part. The volumetric data is typically part of a larger environment space and thus needs to be located within that space, so that volumes for different parts (eg arm and leg of a single human) may be presented in a spatially correct manner. Typically volumes are not a unit cube in size, so extra dimensional information must be provided with the volume to indicate its true size in the local coordinate system. The volume data shall be effected by the normal transformation heirarchy, including scaling and shearing.

2.2.2 Data Representation

Volume rendering requires providing data in a volumetric form. This component uses the 3D texturing component (See ISO/IEC 19775-1 PDAM-1 3D-Texturing) to represent the raw volume data, but without rendering that data directly onto polygonal surfaces. Volumetric rendering may make use of multiple 3D textures to generate a final visual form.

Data may be represented using between 1 and 4 colour components. How each colour component is to be interpreted as part of the rendering shall be defined for each node. Some nodes may require a specific minimum number of components, or define that anything more than a specific number will be ignored. Providing extra data may not be helpful to the implementation.

An implementation is free to provide whatever data reduction techniques are desired under the covers. Explicit volume data representations are provided in the *OctTree* node that allows the user to describe progressively more detailed volumetric data. When the user presents data in this form, it shall be followed as the required rendering technique. However, within a specific volume data representation, the implementation may also perform its own optimisation techniques, for example automatic mipmapping.

Volume visualisation data sets are not required to be represented in sizes that are powers of two. Implementations may need to internally pad the texture sizes for passing to the underlying rendering engine, but user-provided content is not required to do this.

2.2.3 Segmentation Information

The volme data may optionally represent segmented data sets. Doing so requires representing the data in a slightly different manner than a standard volume data set, so a separate node is used. Segmentation data takes the form of an additional volume of data where each voxel represents a segment ID value in addition to other values represented in each voxel. The segmentation information is used by the rendering process to control how each voxel is to be rendered. It is not unusual to use segmentation information to render each segment identifier with a different style - for example bone using ISO surfaces, skin using tone shading etc.

2.2.4 Tensor Representation

This specification does not explicitly handle or represent tensor data. Tensor

information may be rendered using the techniques in this profile, even though no direct data is being transmitted. It is recommended that if an application needs to know about the existance of tensor data, that the metadata capabilities of the specification be used.

2.2.5 Visual Representation

Volumetric data is typically given as a rectangular block of information. Turning that into something meaningful where structures may be discernable is the job of the rendering process. However, there is not a one-size-fits-all approach to volume rendering. A technique that is good for exposing structures for medical visualisation may be poor for fluid simulation visualisation.

To allow for these different visual outputs, this component separates the scene graph into two sets of responsibilities - nodes for representing the volume data, and nodes for rendering that volume data in different ways. In this way, the same rendering process may be used for different sets of volume data, or varying rendering styles may be used to highlight different structures within the one volume.

Many rendering techniques map the volume data to a visual representation through the use of another texture known as a Transfer function. This secondary texture defines the colours to use, acting as a form of lookup table. Transfer functions can be defined in 1, 2 or 3 dimensions. As X3D does not define a 1-dimensional texture capability, this can be simulated through the use of a 2D texture that is only 1 pixel wide.

2.3 Interaction with Other Nodes and Components

2.3.1 Overview

Volumetric rendering requires a completely different implementation path to traditional polygonal rendering. The data represents not only surface information, but also colour and potentially lighting. As such, volume rendering occupies the space in the renderable scene graph that is a *X3DShapeNode* rather than as individual geometry or appearance information.

2.3.2 Lighting

Volumetric rendering is not required to follow the standard lighting equations for this specification. Many techniques will include the ability to self-light and self-shadow using information from the parent scene graph (light scoping etc).

The volume data is rendered using one or more rendering styles. Each style defines its own lighting equation. Many of these involve non-photorealistic effects. Each style will present it's own lighting equation on how to get from the underlying voxel representation to the contributed output colour. The following are some common terms that will be found in the lighting equations:

- O_V: The initial opacity of the object prior to the use of this style. This opacity may include a previously-calculated transfer function evaluation.
- O_g The output opacity of the object resulting from evaluating this style.
- C_V: The initial colour of the object prior to the use of this style. This opacity may include a previously-calculated transfer function evaluation.
- C_g The output colour of the object resulting from evaluating this style.
- Δf: The normalised value gradient of the voxel. This is the rate of change of the value relative to the values in neighbouring voxels.
- **V**: The vector from the viewer's position to the voxel being evaluated, in the local coordinate space of the volume data
- **n**: The local surface normal. This may be provided by the user through another 3D texture that contains a surface normal for each voxel or internally calculated through algorithmic means.
- $\mathbf{L}_i$ Light direction vector from light source *i*. Typically involves a summation over all light sources affecting the volume.

2.3.3 Geometry

The volumetric rendering nodes are a leaf node in the renderable tree. Volumetric nodes may exist as part of a shared scene graph with DEF/USE, though it is expected to be very rare to see this in practice.

2.4 Conformance

2.4.1 Node Support

The minimum required voxel dimensions that shall be supported are 256x256x256.

2.4.2 Hardware requirements

There is no specific requirements for hardware accelaration of this component. In addition, this component does not define the specific implementation strategy to be used by a given rendering style. It is equally valid to implement the code using simple multi-pass rendering as it is to use hardware shaders.

2.4.3 Scene Graph Interaction

Sensor nodes that require interaction with the geometry (eg TouchSensor) shall provide intersection information based on the volume's bounds for minimum conformance. An implementation may optionally provide real intersection information based on performing ray casting into the volume space and reporting the first non-transparent voxel hit.

Navigation and collision detection shall also require a minimal conformance requirement of using the bounds of the volume. In addtion, the implementation may allow greater precision with non-opaque voxels, in a similar manner to the sensor interactions.

3 Abstract Types

3.1 *X3DVolumeNode*

```
X3DVolumeNode : 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
}
```

This abstract node type is the base type for all node types that describe volumetric data to be rendered. It sits at the same level as the polygonal *X3DShapeNode* (see ISO/IEC 19774-1 12.3.4 X3DShapeNode) within the scene graph structure, but defines volumetric data rather polygons.

The *dimensions* field specifies the dimensions of this geometry in the local coordinate space using standard X3D units. It is assumed the volume is centered around the local origin. If the bounding box size is set, it will typically be the same size as the dimensions.

3.2 *X3DVolumeRenderStyleNode*

```
X3DVolumeRenderStyleNode : X3DNode {
  SFBool [in,out] enabled TRUE
}
```

This abstract node type is the base type for all node types which specify a specific visual rendering style to be used.

The *enabled* field defines whether this rendering style should be currently applied to the volume data. If the field is set to FALSE, then the rendering shall not be applied at all. The render shall act as though no volume data is rendered when set to FALSE. Effectively, this allows the end user to turn on and off volume rendering of specific parts of the volume without needing to add or remove style definitions from the volume data node.

4 Node reference

4.1 BoundaryEnhancementVolumeStyle

```
BoundaryEnhancementVolumeStyle : X3DVolumeRenderStyleNode {
  SFBool      [in,out] enabled          TRUE
  SFFloat     [in,out] contribution     0.2     [0,1]
  SFColorRGBA [in,out] gradientColor    0 0 0 1 [0,1]
  SFNode      [in,out] metadata         NULL    [X3DMetadataObject]
  SFNode      [in,out] surfaceNormals   NULL    [X3DTexture3DNode]
  SFNode      [in,out] transferFunction NULL    [X3DTextureNode]
  SFFloat     [in,out] retainedOpacity  1       [0,1]
  SFFloat     [in,out] boundaryOpacity  0       [0,∞)
  SFFloat     [in,out] opacityFactor    1       [0,∞)
}
```

Provides boundary enhancement for the volume rendering style. In this style the colour rendered is based on the gradient magnitude. Faster changing gradients (surface normals) are darker than slower changing. Areas of different density are made more visible relative to parts that are relatively constant density.

The *gradientColor* field defines the maximum colour that should be applied

at the point of maximum gradient change. Where there is little or no gradient change, the colour is interpolated to transparent. The transfer function, if defined, is evaluated before applying the *gradientColor* calculation.

The *surfaceNormals* field is used to provide pre-calculated surface normal information for each voxel. If provided, this shall be used for all lighting calculations. If not provided, the implementation shall automatically generate surface normals using an implementation-specific method. If a value is provided, it shall be exactly the same voxel dimensions as the base volume data that it represents. If the dimension are not identical then the browser shall generate a warning and automatically generate its own internal normals as though no value was provided for this field.

The output colour for this style is obtained by combining a fraction of the volume's original opacity with an enhancment based on the local boundary strength (magnitude of the gradient between adjacent voxels). Any colour information (RGB components) obtained from the transfer function evaluation is untouched by this style. The function used is

$O_g = O_v (k_{gc} + k_{gs}(|\Delta f|)^{k_{ge}})$

where

- k_{gc} is the amount of initial opacity to mix into the output (*retainedOpacity*)
- k_{gs} is the amount of the gradient enhancement to use (*boundaryOpacity*)
- k_{ge} is a power function to control the slope of the opacity curve to highlight the dataset. (*opacityFactor*)

4.2 CartoonVolumeStyle

```
CartoonVolumeStyle : X3DVolumeRenderStyleNode {
  SFBool      [in,out] enabled          TRUE
  SFNode      [in,out] metadata         NULL    [X3DMetadataObject]
  SFNode      [in,out] surfaceNormals   NULL    [X3DTexture3DNode]
  SFColorRGBA [in,out] parallelColor    0 0 0 1 [0,1]
  SFColorRGBA [in,out] orthogonalColor  1 1 1 1 [0,1]
  SFInt32     [in,out] colorSteps       4       [1,64]
}
```

Uses the cartoon-style nonphotorealistic rendering of the volume. Cartoon rendering uses two colours that are rendered in a series of distinct flat-shaded sections based on the local surface normal's closeness to the average normal, with no gradients in between.

The *surfaceNormals* field contains a 3D texture with at least 3 component values. Each voxel in the texture represents the surface normal direction for the corresponding voxel in the base data source. This texture should be identical in dimensions to the source data. If not, the implementation may interpolate or average between adjacent voxels to determine the average normal at the voxel required. If this field is empty, the implementation shall automatically determine the surface normal using algorithmic means.

The *parallelColor* field specifies the colour to be used for surface normals

that are orthogonal to the viewer's current location (the plane of the surface itself is parallel to the user's view direction).

The *orthogonalColor* field specifies the colour to be used for surface normals that are parallel to the viewer's current location (the plane of the surface itself is orthogonal to the user's view direction). Surfaces that are further than orthgonal to the viewpoint position (ie back facing) are not rendered and shall have no colour calculated for them.

The *colorSteps* field indicates how many distinct colours should be taken from the interpolated colours and used to render the object. If the value is 1, then no colour interpolation takes place, and only the orthogonal colour is used to render the surface with. Any other value and the colours are interploted between *parallelColor* and *orthogonalColor* in HSV colour space for the RGB components, and linearly for the alpha component. From this, determine the number of colours using a midpoint calculation.

To determine the colours to be used, the angles for the surface normal relative to the view position are used. Divide the range π/2 by *colourSteps*.(The two ends of the spectrum are not interpolated in this way and shall us the specified field values). For each of the ranges , other than the two ends, find the midpoint angle and determine the interpolated colour at that point.

For example, using the default field values, the colour ranges would be used:

- 1,1,1,1 for angles $[0 - \pi/8)$
- 0.625,0.625,0.625,1 for angles $[\pi/8 - \pi/4)$,
- 0.375,0.375,0.375,1 for angles $[\pi/4 - 3\pi/8)$,
- 0,0,0,1 for angles $[3\pi/8 - \pi/2]$

4.3 CompositeVolumeStyle

```
CompositeVolumeStyle : X3DVolumeRenderStyleNode {
  SFBool [in,out] enabled  TRUE
  SFNode [in,out] metadata NULL  [X3DMetadataObject]
  SFBool [in,out] ordered  FALSE
  MFNode [in,out] styles   []    [X3DVolumeRenderStyleNode]
}
```

A rendering style node that allows compositing multiple styles together into a single rendering pass. This is used, for example to render a simple image with both edge and silhouette styles.

The *styles* field contains a list of contributing style node references that can be applied to the object. Whether the styles should be strictly rendered in order or not is dependent on the *ordered* field value. If this field value is FALSE, then the implementation may apply the various styles in any order (or even in parallel if the underlying implementation supports it). If the value is TRUE, then the implementation shall apply each style strictly in the order declared, starting at index 0.

4.4 EdgeEnhancementVolumeStyle

```
EdgeEnhancementVolumeStyle : X3DVolumeRenderStyleNode {
```

```
  SFFloat     [in,out] contribution      0.2     [0,1]
  SFColorRGBA [in,out] edgeColor         0 0 0 1 [0,1]
  SFBool      [in,out] enabled           TRUE
  SFFloat     [in,out] gradientThreshold 0.4     [0,0 - π/2]
  SFNode      [in,out] metadata          NULL    [X3DMetadataObject]
  SFNode      [in,out] surfaceNormals    NULL    [X3DTexture3DNode]
}
```

Provides edge enhancement for the volume rendering style. Enhancement of the basic volume is provided by darkening voxels based on their orientation relative to the view direction. Perpendicular voxels are coloured according to the *edgeColor* while voxels parallel are not changed at all. A threshold can be set where the proportion of how close to parallel the direction needs to be before no colour changes are made.

The *gradientThreshold* field defines the minimum angle (in radians) away from the view direction vector that the surface normal needs to be before any enhancement is applied.

The *edgeColor* field defines the colour to be used to highlight the edges.

The *surfaceNormals* field contains a 3D texture with at least 3 component values. Each voxel in the texture represents the surface normal direction for the corresponding voxel in the base data source. This texture should be identical in dimensions to the source data. If not, the implementation may interpolate or average between adjacent voxels to determine the average normal at the voxel required. If this field is empty, the implementation shall automatically determine the surface normal using algorithmic means.

4.5 ISOSurfaceVolumeStyle

```
ISOSurfaceVolumeStyle : X3DVolumeRenderStyleNode {
  SFBool [in,out] enabled        TRUE
  SFBool [in,out] lighting       FALSE
  SFNode [in,out] metadata       NULL [X3DMetadataObject]
  SFNode [in,out] surfaceNormals NULL [X3DTexture3DNode]
  SFBool [in,out] shadows        FALSE
}
```

Renders the volume using ISO-surface colouring.

The *lighting* field controls whether the rendering should calculate and apply shading effects to the visual output. When shading is applied, the value of the *surfaceNormals* field can be used to provide pre-generated surface normals for lighting calculations.

The *shadows* field controls whether the rendering should calculate and apply shadows to the visual output. A value of FALSE requires that no shadowing be applied. A value of TRUE requires that shadows be applied to the object. If the *lighting* field is set to FALSE, this field shall be ignore and no shadows generated.

The *surfaceNormals* field contains a 3D texture with at least 3 component values. Each voxel in the texture represents the surface normal direction for the corresponding voxel in the base data source. This texture should be identical in dimensions to the source data. If not, the implementation may interpolate or average between adjacent voxels to determine the average normal at the voxel required. If this field is empty, the implementation shall automatically determine the surface normal using algorithmic means.

4.6 OctTree

```
OctTree : X3DChildNode, X3DBoundedObject {
  MFNode  [in,out] highRes      NULL  [X3DChildNode]
  SFNode  [in,out] lowRes       NULL  [X3DGroupingNode, X3DShapeNode, X3DVolumeShapeNode]
  SFNode  [in,out] metadata     NULL  [X3DMetadataObject]
  SFBool  [out]    lowResActive
  SFVec3f []       center       0 0 0 (-∞,∞)
  SFFloat []       range        20    [0,∞)
}
```

Allows for the definition of multiresolution data sets that resolve using octants of volume. This node is not restricted to only having volume data as its children - all other geometry types are also valid structures.

The level of detail is switched depending upon whether the user is closer or further than *range* metres from the coordinate *center*.

The *lowRes* field holds the low resolution object instance to be rendered when the viewer is outside *range* metres. The *highRes* field is used to hold the geometry to be viewed when the inside *range* metres. An OctTree renders up to 8 children sub graphs as defined by the *highRes* field. If this field contains more than 8 children, only the first 8 shall be rendered. If less than 8 children are defined, all shall be rendered. It is up to the user to spatially located the geometry for each of the children subgraphs.

4.7 OpacityMapVolumeStyle

```
OpacityMapVolumeStyle : X3DVolumeRenderStyleNode {
  SFBool [in,out] enabled          TRUE
  SFNode [in,out] metadata         NULL [X3DMetadataObject]
  SFNode [in,out] transferFunction NULL [X3DTextureNode]
}
```

Renders the volume using the opacity mapped to a transfer function texture. This is the default rendering style if none is defined for the volume data.

The *transferFunction* field holds a single texture representation in either two or three dimensions that map the voxel data values to a specific colour output. If no value is supplied for this field, the default implementation shall generate a 256x1 greyscale alpha-only image that blends from completely opaque at pixel 0 to fully transparent at pixel 255.

4.8 SegmentedVolumeData

```
SegmentedVolumeData : X3DVolumeNode {
  SFVec3f [in,out] dimensions       1 1 1     (0,∞)
  SFNode  [in,out] metadata         NULL      [X3DMetadataObject]
  MFNode  [in,out] renderStyle      []        [X3DVolumeRenderStyleNode]
  MFBool  [in,out] segmentEnabled   []
  SFNode  [in,out] segmentedVoxels  []        [X3DTexture3DNode]
  SFVec3f []       bboxCenter       0 0 0     (-∞,∞)
  SFVec3f []       bboxSize         -1 -1 -1  [0,∞) or -1 -1 -1
}
```

Defines a segmented volume data set that allows for representation of different rendering styles for each segment identifier.

The *renderStyle* field optionally describes a particular rendering style to be used. If this field has a non-zero number of values, then the defined rendering style is to be applied to the object. If the object is segmented, then the index of the segment shall look up the rendering style at the given

index in this array of values and apply that style to data described by that segment ID. If the field value is not specified by the user, the implementation shall use a OpacityMapVolumeStyle node with default values.

The *segmentedVoxels* field holds a 3D texture with the segment IDs for each voxel. The texture supplied must have either 2 or 4 colour components. The alpha component of the texture defines the segment identifier at each voxel. If a 3-component texture is supplied, the red and green components shall define 2D data at that voxel, and the blue component shall contain the segment identifier. Providing a source texture with 1 component shall result in no rendering of this data.

The *segmentEnabled* field allows for controlling whether a segment should be rendered or not. The indices of this array corresponds to the segment ID. A value at index *i* of FALSE marks any data with the corresponding segment ID to be not rendered. If a segment ID is used that is greater than the length of the array, the value is assumed to be TRUE.

4.9 SilhouetteEnhancementVolumeStyle

```
SilhouetteEnhancementVolumeStyle : X3DVolumeRenderStyleNode {
  SFBool      [in,out] enabled           TRUE
  SFNode      [in,out] metadata          NULL    [X3DMetadataObject]
  SFNode      [in,out] surfaceNormals    NULL    [X3DTexture3DNode]
  SFFloat     [in,out] silhouetteFactor  0.5     [0,∞)
}
```

Provides silhouette enhancement for the volume rendering style. Enhancement of the basic volume is provided by darkening voxels based on their orientation relative to the view direction. Perpendicular voxels are coloured according to the *edgeColor* while voxels parallel are not changed at all. A threshold can be set where the proportion of how close to parallel the direction needs to be before no colour changes are made.

$O_g = O_v * (1 - |\mathbf{n} . \mathbf{V}|) ^ \text{silhouetteFactor}$

The *surfaceNormals* field contains a 3D texture with at least 3 component values. Each voxel in the texture represents the surface normal direction for the corresponding voxel in the base data source. This texture should be identical in dimensions to the source data. If not, the implementation may interpolate or average between adjacent voxels to determine the average normal at the voxel required. If this field is empty, the implementation shall automatically determine the surface normal using algorithmic means.

4.10 StippleVolumeStyle

```
StippleVolumeStyle : X3DVolumeRenderStyleNode {
  SFFloat [in,out] distanceFactor             1     [0,∞)
  SFBool  [in,out] enabled                    TRUE
  SFFloat [in,out] interiorFactor             1     [0,∞)
  SFFloat [in,out] lightingFactor             1     [0,∞)
  SFNode  [in,out] metadata                   NULL  [X3DMetadataObject]
  SFFloat [in,out] gradientThreshold          0.4   [0,0-π/2]
  SFFloat [in,out] gradientRetainedOpacity    1     [0,1]
  SFFloat [in,out] gradientBoundaryOpacity    0     [0,∞)
  SFFloat [in,out] gradientOpacityFactor      1     [0,∞)
  SFFloat [in,out] silhouetteRetainedOpacity  1     [0,1]
  SFFloat [in,out] silhouetteBoundaryOpacity  0     [0,∞)
  SFFloat [in,out] silhouetteOpacityFactor    1     [0,∞)
  SFFloat [in,out] resolutionFactor           1     [0,∞)
}
```

Renders the volume using stipple patterns making use of the Wang stipple patterns for 3D dimensional data sets. Stipple patterns are automatically generated by the browser internals based on a number of algorithmic hints. It is recommended the approach defined in [STIPPLE] is used.

The general approach of the rendering process is to render a set of points, whose density is defined by a number of factors - edge, boundary silhouette enhancements, lighting and other effects. The renderer determines a absolute maximum density of points in a voxel (N_{max} and then evaluates every voxel to obtain the number of points (N) to be rendered. The distribution of points in the volume of space is an implementation- specific detail. The final calculation of N is determined by the follow set of equations:

The *gradientThreshold* field defines the minimum angle (in radians) away from the view direction vector that the surface normal needs to be before any boundary enhancement is applied.

$N = N_{max} * T_b * T_s * T_d * T_l * T_r * T_i$
$T_b = C_v * (k_{gc} + k_{gs} * (|\Delta f|)^{k_{ge}})$
$T_s = C_v * (k_{sc} + k_{ss} * (1 - (|\Delta f \cdot \mathbf{V})) ^ {k_{se}})$
$T_d = 1 + (z / a) ^ {k_{de}}$
$T_l = 1 - (\mathbf{L}_i \cdot \Delta f) ^ {k_{le}}$
$T_r = ((D_{near} + d_i) / (D_{near} + d_0)) ^ {k_{re}}$
$T_i = ||\Delta f|| ^ {k_{ie}}$

where

- k_{gc} is the amount of initial opacity to mix into the output (*gradientRetainedOpacity*)
- k_{gs} is the amount of the gradient enhancement to use (*gradientBoundaryOpacity*)
- k_{ge} is a power function to control the slope of the opacity curve to highlight the dataset. (*gradientOpacityFactor*)
- k_{sc} is the amount of initial opacity to mix into the output (*silhouetteRetainedOpacity*)
- k_{ss} is the amount of the gradient enhancement to use (*silhouetteBoundaryOpacity*)
- k_{se} is a power function to control the slope of the opacity curve to highlight the dataset. (*silhouetteOpacityFactor*)
- k_{de} is the distance attenuation factor (*distanceFactor*)
- k_{le} is the lighting contributions factor (*lightingFactor*)
- k_{re} is the resolution enhancement contribution factor (*resolutionFactor*)
- D_{near} is the distance from the eye to the near clipping plane.
- d_i is the current distance from the near clipping plane to the volume's center.
- d_0 is the original distance from the near clipping plane to the volume's center at world startup time.
- k_{ie} is the interior enhancement contribution factor (*interiorFactor*).

4.11 ToneMappedVolumeStyle

```
ToneMappedVolumeStyle : X3DVolumeRenderStyleNode {
  SFColorRGBA [in,out] coolColor      0 0 1 0  [0,1]
  SFBool      [in,out] enabled        TRUE
  SFNode      [in,out] metadata       NULL     [X3DMetadataObject]
  SFNode      [in,out] surfaceNormals NULL     [X3DTexture3DNode]
  SFColorRGBA [in,out] warmColor      1 1 0 0  [0,1]
}
```

Renders the volume using the Gooch shading model of two-toned warm/cool colouring. Two colours are defined, a warm colour and a cool colour and the renderer shades between them based on the orientation of the voxel relative to the user. This is not the same as the basic ISO surface shading and lighting. The following colour formula is used:

$cc = (1 + \mathbf{L}_i \cdot \mathbf{n}) * 0.5$
$C_g = \Sigma\ cc * warmColor + (1 - cc) * coolColor$

The *warmColor* and *coolColor* fields define the two colours to be used at the limits of the spectrum. The *warmColor* field is used for surfaces facing towards the light, while the *coolColor* is used for surfaces facing away from the light direction.

The *surfaceNormals* field contains a 3D texture with at least 3 component values. Each voxel in the texture represents the surface normal direction for the corresponding voxel in the base data source. This texture should be identical in dimensions to the source data. If not, the implementation may interpolate or average between adjacent voxels to determine the average normal at the voxel required. If this field is empty, the implementation shall automatically determine the surface normal using algorithmic means.

4.12 VolumeData

```
VolumeData : X3DVolumeNode {
  SFVec3f [in,out] dimensions       1 1 1     [0,∞)
  SFNode  [in,out] metadata         NULL      [X3DMetadataObject]
  SFNode  [in,out] renderStyle      NULL      [X3DVolumeRenderStyleNode]
  MFNode  [in,out] voxels           []        [X3DTexture3DNode]
  SFVec3f []       bboxCenter       0 0 0     (-∞,∞)
  SFVec3f []       bboxSize         -1 -1 -1  [0,∞) or -1 -1 -1
}
```

Defines the volume information to be used on a simple non-segmented volumetric description that uses a single rendering style node for the complete volume.

The *renderStyle* field allows the user to specify a specific rendering technique to be used on this volumetric object. If the value not specified by the user, the implementation shall use a OpacityMapVolumeStyle node with default values.

The *voxels* field provides the raw voxel information to be used by the specific rendering styles. The value is any *X3DTexture3DNode* type and may have any number of colour components defined. The specific interpretation for the values at each voxel shall be defined by the value of the *renderStyle* field. If more than one node is defined for this field then each node after the first shall be treated as a mipmap level of monotonically decreasing size. Each level should be half the dimensions of the previous level

5 Support Levels

The Volume rendering component defines three levels of support as specified in Table 2.

Table 2 — Volume rendering component support levels

Level	Prequisites	Nodes/Features	Support
Level 1	Core 1 Grouping 1 Shape 1 Rendering 1		
		X3DVolumeRenderStyleNode	n/a
		X3DVolumeShapeNode	n/a
		OctTree	All fields fully supported
		OpacityMapVolumeStyle	Only 2D texture transfer function needs to be supported. All other fields fully supported.
		VolumeData	All fields fully supported
Level 2	Core 1 Grouping 1 Shape 1 Rendering 1		
		BoundaryEnhancementVolumeStyle	All fields fully supported
		CompositeVolumeStyle	ordered field is always treated as FALSE. All other fields fully supported
		EdgeEnhancementVolumeStyle	All fields fully supported

		SegmentedVolumeData	All fields fully supported
		SilhouetteEnhancementVolumeStyle	All fields fully supported
		ToneMappedVolumeStype	All fields fully supported
Level 3	Core 1 Grouping 1 Shape 1 Rendering 1		
		ISOVolumeStyle	All fields fully supported
		CartoonVolumeStyle	All fields fully supported
		CompositeVolumeStyle	All fields fully supported
		StippleVolumeStype	All fields fully supported

[Xj3D Homepage | Xj3D @ Web3d | Screenshots | Dev docs | Dev Releases | Contributors | Getting Started]

Last updated: $Date: 2005/12/13 19:36:32 $