Tagged Volume Rendering of the Heart: A Case Study

Release 1.1

Dan Mueller[1]

July 17, 2007

[1]Queensland University of Technology, Brisbane, Australia

Abstract

This is a companion paper describing the process and parameters for tagged volume rendering of the heart. We firstly review the relevant concepts and consider the problem of visualising the coronary arteries from computed tomography angiography (CTA) images. We then discuss the implementation using the SharpImage prototyping environment. Finally, we list the set of commands capable of reproducing our results using the accompanying dataset.

Keywords: *tagged volume rendering, heart, coronary arteries, ITK, SharpImage*

Contents

1 Introduction

For diagnostic and treatment planning purposes, radiologists and surgeons require means to visualise the coronary arteries from computed tomography angiography (CTA) images. However, this task is non-trivial for various reasons: (a) unwanted structures (such as the thoracic cage) clutter the regions of interest; (b) the contrast agent highlights the coronary arteries (as desired), but also portions of the ventricles, atria, aorta, and pulmonary arteries, and (c) the coronary arteries can exhibit high-intensity calcification artefacts. Clinicians currently have a number of tools at their disposal, including: thin slab maximum intensity projection [1], curved planar reformatting [3], and direct volume rendering (DVR).

In the case of direct volume rendering, the dataset is segmented during the rendering process ('online') via a lookup table ('transfer function'). Various online segmentation algorithms for DVR have been discussed in the literature [8, 4, 12]. Kniss et al. [4] advocated the use of two dimensional transfer functions dependent on pixel value and gradient magnitude (termed 'value-magnitude' transfer functions), in which the user interactively specifies the function using a number of widgets. This method is simple, fast and effective for basic datasets; yet, it is insufficient for delineating objects of interest sharing similar characteristics. Tzeng et al. [12] proposed an approach using an online machine learning algorithm (such as an artificial neural network or support vector machine), with the current pixel value, gradient magnitude, location, and neighbourhood values as inputs. Because of their online nature, both of these methods are constrained by the graphics hardware on which they are implemented.

A number of more recent methods combine both offline and online segmentation. Hadwiger et al. [2] proposed tagged volume rendering which uses a number of *a priori* segmented binary masks ('tags'), each assigned separate transfer functions. Kniss et al. [5] introduced a probabilistic method which decouples classification from colour-mapping, simplifying the transfer function interface. Weber et al. [13] proposed the use of contour-trees for classifying objects based on topology (the trees are computed and simplified offline before being uploaded the the graphics hardware). Of these methods, tagged volume rendering is the most flexible as it does not restrict the choice of segmentation algorithm.

In this paper we apply tagged volume rendering to the problem of visualising the coronary arteries. We utilise Hessian-based line filters for segmenting the coronary arteries [9] and use the Fast Marching active contour method [10] for segmenting the pericardial cavity. A suitable speed function is presented for controlling the expansion of the active contour. The exact set of SharpImage [7] commands for reproducing the results on the accompanying dataset are given. For your reference, a pre-print of the main MICCAI paper can be found here: http://eprints.qut.edu.au/archive/00008421/

2 Implementation

This section details the implementation of the method. It requires built versions of ITK 3.2, ManagedITK 3.2.0.3 [6], and SharpImage 0.9.2 [7]. Before tackling the case study, we must extend the SharpImage environment to handle vessel segmentation and implement a fragment shader for tagged transfer functions. Pre-built versions are available if you wish to skip this step (see below).

2.1 Vessel Segmentation

SharpImage does not have an explicit vessel segmentation command, however it is quite easy to add one. The first step is to wrap the required filters using a ManagedITK external project (see `source/vesselness/CMakeLists.txt`):

```
PROJECT(WrapVesselness)

# Find required packages
FIND_PACKAGE(ITK REQUIRED)
FIND_PACKAGE(ManagedITK REQUIRED)

# Use required packages
INCLUDE(${ITK_USE_FILE})
INCLUDE(${MANAGED_ITK_USE_FILE})

# Wrap the project
BEGIN_MANAGED_WRAP_EXTERNAL_PROJECT("Filtering" "Vesselness")
  SET(MANAGED_WRAPPER_OUTPUT "${CMAKE_BINARY_DIR}")
END_MANAGED_WRAP_EXTERNAL_PROJECT()
```

We need to wrap two filters, `itkHessianRecursiveGaussianImageFilter` and `itkHessian3DToVesselnessMeasureImageFilter`:

```
# Begin the wrapping
WRAP_CLASS("itk::HessianRecursiveGaussianImageFilter")

  # Wrap the class for INT and REAL types
  WRAP_IMAGE_FILTER_INT(1)
  WRAP_IMAGE_FILTER_REAL(1)

  # Wrap the Sigma property
  BEGIN_MANAGED_PROPERTY("Sigma" SET)
    SET(MANAGED_PROPERTY_SUMMARY   "Set the Gaussian variance (in image spacing).")
    SET(MANAGED_PROPERTY_TYPE      "double")
    SET(MANAGED_PROPERTY_SET_BODY  "m_PointerToNative->SetSigma( value );")
  END_MANAGED_PROPERTY()

  # Wrap the NormalizeAcrossScale property
  BEGIN_MANAGED_PROPERTY("NormalizeAcrossScale" GETSET)
    SET(MANAGED_PROPERTY_SUMMARY   "Get/set whether normalization should occur.")
    SET(MANAGED_PROPERTY_TYPE      "bool")
    SET(MANAGED_PROPERTY_GET_BODY  "return m_PointerToNative->GetNormalizeAcrossScale( );")
    SET(MANAGED_PROPERTY_SET_BODY  "m_PointerToNative->SetNormalizeAcrossScale( value );")
  END_MANAGED_PROPERTY()

# End the wrapping
END_WRAP_CLASS()
```

```
# Begin the wrapping
WRAP_CLASS("itk::Hessian3DToVesselnessMeasureImageFilter")

  # Wrap the template arguments
  WRAP_TEMPLATE("${ITKM_F}" "${ITKT_F}")

  # Wrap the Alpha1 property
  BEGIN_MANAGED_PROPERTY("Alpha1" GETSET)
    SET(MANAGED_PROPERTY_SUMMARY   "Get/set alpha1.")
    SET(MANAGED_PROPERTY_TYPE      "double")
    SET(MANAGED_PROPERTY_GET_BODY  "return m_PointerToNative->GetAlpha1( );")
    SET(MANAGED_PROPERTY_SET_BODY  "m_PointerToNative->SetAlpha1( value );")
  END_MANAGED_PROPERTY()

  # Wrap the Alpha2 property
  BEGIN_MANAGED_PROPERTY("Alpha2" GETSET)
    SET(MANAGED_PROPERTY_SUMMARY   "Get/set alpha2.")
    SET(MANAGED_PROPERTY_TYPE      "double")
    SET(MANAGED_PROPERTY_GET_BODY  "return m_PointerToNative->GetAlpha2( );")
    SET(MANAGED_PROPERTY_SET_BODY  "m_PointerToNative->SetAlpha2( value );")
  END_MANAGED_PROPERTY()

# End the wrapping
END_WRAP_CLASS()
```

We point CMake at this folder, choose a build directory, and configure the project for Visual Studio 8[1]. ManagedITK should create the relevant files, including a `FinishCMake.bat` file which must be run *before* opening the solution file and compiling the project. The output will be a single executable `ManagedITK.Filtering.Vesselness.dll`.

The next step is to create a Python script for use with SharpImage (see `source/vesselness/Vesselness.py`):

```
#=========================================================================
# Module:    Vesselness.py
#=========================================================================

# Import the base script class
import ImageToImageScript
from ImageToImageScript import *

# Add CLR reference and import required libraries
clr.AddReference("ManagedITK.Filtering.Vesselness")
from itk import *

class VesselnessScript(ImageToImageScriptObject):
    # ------------------------------------------------------------------------
    Name = "Vesselness"
    Help = """Implements a Hessian-based line enhancement filter for segmenting\r\
              tubular objects. The input image must be a 3-D image."""
    Sigma = 1.0
    Alpha1 = 0.5
    Alpha2 = 2.0
    NormalizeAcrossScale = False
    # ------------------------------------------------------------------------

    def ThreadedDoWork(self):
        """ Perform the main functions of the script on a background thread. """
        try:
            # Start
```

[1] We have not tried Visual Studio Express Edition, so there may be issues if you are using that specific compiler.

```
            self.StartedWork()
            self.WriteInputName()

            # Compute hessian
            filterHessian = itkHessianRecursiveGaussianImageFilter.New( self.Input )
            self.AddEventHandlersToProcessObject( filterHessian )
            filterHessian.SetInput( self.Input )
            filterHessian.Sigma = self.Sigma
            filterHessian.NormalizeAcrossScale = self.NormalizeAcrossScale

            # Compute vesselness
            filterVesselness = itkHessian3DToVesselnessMeasureImageFilter.New( itkPixelType.F )
            self.AddEventHandlersToProcessObject( filterVesselness )
            filterVesselness.SetInput( filterHessian.GetOutput() )
            filterVesselness.Alpha1 = self.Alpha1
            filterVesselness.Alpha2 = self.Alpha2
            filterVesselness.UpdateLargestPossibleRegion( )
            filterVesselness.GetOutput( self.Output )

            # Clean up and finish
            self.DisconnectInputAndOutput()
            self.DisposeOfObject( filterHessian )
            self.DisposeOfObject( filterVesselness )
            self.WriteOutputName()
            self.FinishedWork( True )

        except Exception, ex:
            self.HandleException( ex )
            self.FinishedWork( False )
            self.Finalise( )
```

These files (`ManagedITK.Filtering.Vesselness.dll` and `Vesselness.py`) must be copied to the SharpImage Scripting folder (we recommend creating a subfolder called `SharpImage/Scripting/Custom` or similar). A pre-compiled version of the ManagedITK assembly is provided with this article if you wish to skip this step (see `results/ManagedITK.Filtering.Vesselness.dll`).

2.2 Fragment Shader

We implement a derivation of the tagged volume rendering method described by Hadwiger et al. [2]. In their original method tags represent exact objects, which requires tri-linear interpolation and complex boundary filtering to avoid artefacts and improve resolution. Our application allows us to make a different assumption: a tag is a mask guaranteeing to *include* structures of interest, but not *exactly* delineate them. We can therefore use nearest neighbour interpolation and a stack of value-magnitude transfer functions to refine the structures inside the tags (ie. a 3-D texture, similar to Svakhine et al. [11, Sec. 3]). The tagged transfer function is implemented using OpenGL Shading Language (GLSL) (see `source/fragment-shader/shader-vmt-phong.frag`):

```
//==============================================================================
// Module:    shader-vmt-phong.frag
//==============================================================================
uniform sampler3D sam3Tex0;     // Sampler for transfer function
uniform sampler3D sam3Tex1;     // Sampler for value image
uniform sampler3D sam3Tex2;     // Sampler for gradient image
uniform sampler3D sam3Tex3;     // Sampler for tags image
varying vec3 pos3Tex1;          // Current coord of value image
```

```
varying vec3 pos3Tex2;          // Current coord of gradient image
varying vec3 pos3Tex3;          // Current coord of tags image

void main()
{
    // Interpolate images
    float value = vec4( texture3D(sam3Tex1, pos3Tex1) ).a;
    float gradmag = vec4( texture3D(sam3Tex2, pos3Tex2) ).a;
    float tags = vec4( texture3D(sam3Tex3, pos3Tex3) ).a;

    // Interpolate transfer function
    const float fNumLayers = 4.0;
    vec3 pos3TfAll = vec3( value, 1.0-gradmag, 0.0 );
    vec3 pos3TfTag1 = vec3( value, 1.0-gradmag, 1.0/(fNumLayers-1.0) );
    vec3 pos3TfTag2 = vec3( value, 1.0-gradmag, 2.0/(fNumLayers-1.0) );
    vec4 val4TfAll  = texture3D( sam3Tex0, pos3TfAll );
    vec4 val4TfTag1 = texture3D( sam3Tex0, pos3TfTag1 );
    vec4 val4TfTag2 = texture3D( sam3Tex0, pos3TfTag2 );

    // Adjust alpha for sampling rate
    val4TfAll.a = adjustAlphaForSamplingRate( val4TfAll.a );
    val4TfTag1.a = adjustAlphaForSamplingRate( val4TfTag1.a );
    val4TfTag2.a = adjustAlphaForSamplingRate( val4TfTag2.a );

    // Mix tags
    bool light, enhance;
    vec4 val4Mix;
    if (val4TfAll.a == 0 && tags > 0.5)
    {
        // Tag 2
        val4Mix = val4TfTag2;
        enhance=true; light=true;
        if (val4TfTag2.a <= 0.0) discard;
    }
    else if (val4TfAll.a == 0 && tags > 0.25)
    {
        // Tag 1
        val4Mix = val4TfTag1;
        enhance=true; light=true;
        if (val4TfTag1.a <= 0.0) discard;
    }
    else
    {
        // All
        val4Mix = val4TfAll;
        enhance=true; light=true;
        if (val4TfAll.a <= 0.0) discard;
    }

    // Compute the normal
    vec4 vec4G = gradientFromCentralDifferences( );
    vec4 vec4N = normalize( vec4G );

    // Apply lighting
    if (light)
        gl_FragColor = val4Mix * phong( vec4N, gl_LightSource[0] );
    else
        gl_FragColor = val4Mix;
}
```

3 Commands

This section lists the SharpImage commands required to reproduce our results. You will need a good desktop computer with *at least* 2 GB RAM and a recent graphics card with *at least* 512 MB VRAM (you will receive memory allocation errors otherwise). For this experiment we used an Intel Pentium D, 2 $\times$ 3.0 GHz processors, 2 GB RAM, NVIDIA GeForce 8800 GTX, 768 MB VRAM. We assume the data has been unzipped to `C:/Temp`, along with `source/commands/A-SPHERES.txt`. The full list of commands can be found in `source/commands/commands.txt`.

3.1 Orient and Resize

Because our data came from various sources, we found it desirable to orient the images to the same anatomical position. Much of the processing is undertaken on a subsampled image, so we also resize the image at this point.

```
> Open "C:/Temp/A.mhd#SS3"
> Orient Given="RPI" Desired="ASL"
> Properties Origin=itkPoint(0,0,0)
> Save "C:/Temp/A-ASL.mhd"
> CastToF
> Resize OutputSize=itkSize(256,256,256) Interpolator="Linear"
> Save "C:/Temp/A-ASL-SMALL.mhd"
> Close
```

3.2 Speed Function

We now compute the edge-based speed function:

```
> Open "C:/Temp/A-ASL-SMALL.mhd#F3"
> CurvatureFlow TimeStep=0.1 Iterations=10
> Threshold Lower=-2000 Upper=100 OutsideValue=100
> ConnectedThreshold Lower=60 Upper=100
> Rename "Sternum"
> BinaryPixelMath Operation="Add" Input1="Threshold" Input2="Sternum"
> LevelSetSpeed OutputMinimum=0.0
> Save "C:/Temp/A-ASL-SMALL-SPEED.mhd"
> Close
```

The seed for the region growing can be anywhere in the sternum (we used [50, 90, 100]) and the parameters for the speed function were as follows:

```
Gaussian Normalize=False
Gaussian Sigma=0.5
Sigmoid Alpha=-10.0
Sigmoid Beta=10.0
```

```
Threshold Lower=-250.0
Threshold Upper=355.0
Rescale OutputMinimum=000.000
Rescale OutputMaximum=001.000
```

3.3 Initial Contour

The initial contour could be computed from a set of points or — as we prefer — from a set of spheres generated using a 3-D interface:

```
> Open "C:/Temp/A-ASL-SMALL.mhd#SS3"
> BinaryThreshold Lower=-250 Upper=2000
> CastToUC
> GenerateMaskFromFile "C:/Temp/A-SPHERES.txt"
> BinaryPixelMath Operation="Mask" Input1="Cast" Input2="Mask"
> MorphologicalOpen Operation="Binary" KernelRadius=itkSize(4,4,4)
> Save "C:/Temp/A-ASL-SMALL-INITIAL.mhd"
> Close
```

Notice the morphological opening which removes unwanted pulmonary vessels. The list of spheres is defined in `source/commands/A-SPHERES.txt`:

```
Sphere: Center=119,062,100 Radius=48
Sphere: Center=095,085,076 Radius=45
Sphere: Center=068,106,055 Radius=25
Sphere: Center=112,018,126 Radius=54
Sphere: Center=102,075,126 Radius=50
```

3.4 Fast Marching

We now evolve the active contour, extract the contour at a suitable arrival time, and return the tag to full size:

```
> Open "C:/Temp/A-ASL-SMALL.mhd#SS3"
> Open "C:/Temp/A-ASL-SMALL-INITIAL.mhd#UC3"
> Open "C:/Temp/A-ASL-SMALL-SPEED.mhd#F3"
> FastMarching InitialImage="Initial" StoppingValue=100.0
> MorphologicalErode KernelRadius=itkSize(2,2,2)
> BinaryThreshold Lower=0 Upper=14
> CastToUC
> Resize OutputSize=itkSize(512,373,512) Interpolator="Nearest"
> Save "C:/Temp/A-ASL-HEART.mhd"
> Close
```

3.5 Vesselness

We now segment the vessels using the script created earlier:

```
> Open "C:/Temp/A-ASL-SMALL.mhd#F3"
> Vesselness Sigma=0.7
> ImageRange Minimum=0.0 Maximum=100.0
> ConnectedThreshold Lower=40 Upper=1000
> CastToUC
> Resize OutputSize=itkSize(512,373,512) Interpolator="Nearest"
> MorphologicalDilate Operation="Binary" KernelRadius=itkSize(2,2,2)
> Save "C:/Temp/A-ASL-VESSELS.mhd"
> Close
```

The `ImageRange` script is required to increase the contrast for selecting the seeds. We specified two seeds for the region growing: one in the left coronary artery (LCA) [95, 122, 169], and one in the right coronary artery (RCA) [126, 86, 96].

3.6 Combine Tags

We are now ready to combine our two tags into a single image:

```
> Open "C:/Temp/A-ASL-VESSELS.mhd#UC3"
> Open "C:/Temp/A-ASL-HEART.mhd#UC3"
> ShiftScale Shift=-150
> BinaryPixelMath Operation="Max" Input1="Heart_Shift" Input2="Vessels"
> Save "C:/Temp/A-ASL-TAGS.mhd"
> Close
```

3.7 Histogram

This step is purely optional, but it is often desirable to have a histogram to aid transfer function specification:

```
> Open "C:/Temp/A-ASL-SMALL.mhd#F3"
> MorphologicalGradientMagnitude KernelRadius=itkSize(1,1,1)
> ValueEdgeHistogram Value="Small" Edge="Magnitude" NumberOfBins=[512,128]
> Save "C:/Temp/A-ASL-HISTOGRAM.png"
> Close
```

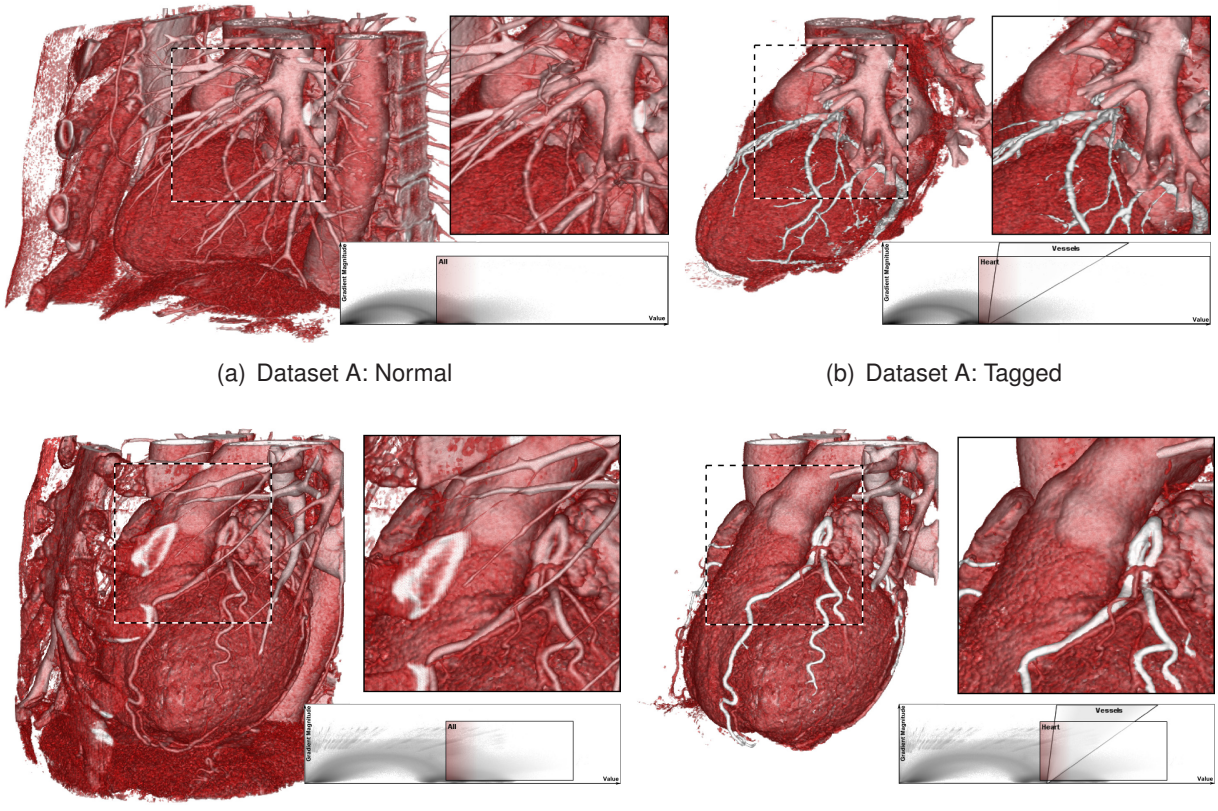
3.8 Volume Render

The preparation is now complete and we are ready to render the volumes:

```
> Open "C:/Temp/A-ASL-HISTOGRAM.png#UC2"
> Open "C:/Temp/A-ASL-TAGS.mhd#UC3"
> Open "C:/Temp/A-ASL.mhd#SS3"
> MorphologicalGradientMagnitude KernelRadius=itkSize(1,1,1)
> RescaleIntensityToUC
> VolumeRender Value="A-ASL.mhd" Gradient="Rescale" Tags="Tags"
        Tf=siTransferFunction(itkSize(512,128),"All","Heart","Vessels")
```

Select the “Renderer Editor” and configure the “FragmentProgramPath”. Select the “Transfer function editor” and right-click on the transfer function to add components similar to those shown in the results section.

4 Results



(a) Dataset A: Normal

(b) Dataset A: Tagged

(c) Dataset C: Normal

(d) Dataset C: Tagged

Figure 1: Resultant images using the normal and tagged volume rendering approaches. Dataset A is provided with this article.

5 Conclusion

We have presented a method for tagged volume rendering of the heart. The method is comprised of two stages: offline segmentation of the coronary arteries and pericardial cavity; followed by the online application of value-magnitude transfer functions to refine the tags. The method was presented using the SharpImage prototyping environment which made the steps somewhat involved, but easily reproduced. Feel free to contact us with any questions or suggestions[2].

[2]Corresponding author: Dan Mueller: d.mueller@qut.edu.au or dan.muel@gmail.com.

References

[1] E. Fishman, D. Ney, D. Heath, F. Corl, K. Horton, and P. Johnson. Volume rendering versus maximum intensity projection in CT angiography: What works best, when, and why. *Radiographics*, 26(3):905–922, 2006. 1

[2] M. Hadwiger, C. Berger, and H. Hauser. High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *Visualization*, pages 301–308. IEEE, 2003. 1, 2.2

[3] A. Kanitsar, R. Wegenkittl, D. Fleischmann, and M. Gröller. Advanced curved planar reformation: flattening of vascular structures. In *IEEE Visualization*, pages 43–50, 2003. 1

[4] J. Kniss, G. Kindlmann, and C. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, 2002. 1

[5] J. Kniss, R. Van Uitert, A. Stephens, G. Li, and T. Tasdizen. Statistically quantitative volume visualization. In *Visualization*, pages 287 – 294. IEEE, 2005. 1

[6] D. Mueller. ManagedITK: .NET wrappers for ITK. *The Insight Journal*, January–June, 2007, Available online: http://insight-journal.org/dspace/handle/1926/501. 2

[7] D. Mueller. SharpImage: An image processing prototyping environment. *The Insight Journal*, July–December, 2007, Available online: http://insight-journal.org/dspace/handle/1926/556. 1, 2

[8] H. Pfister, B. Lorensen, C. Bajaj, G. Kindlmann, W. Schroeder, L. Avila, K. Raghu, R. Machiraju, and J. Lee. The transfer function bake-off. *IEEE Computer Graphics and Applications*, 21(3):16–22, 2001. 1

[9] Y. Sato, S. Nakajima, N. Shiraga, H. Atsumi, S. Yoshida, T. Koller, G. Gerig, and R. Kikinis. Three-dimensional multi-scale line filter for segmentation and visualization of curvilinear structures in medical images. *Medical Image Analysis*, 2(2):143–168, 1998. 1

[10] J. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge Press, 2nd edition, 1999. 1

[11] N. Svakhine, D. Ebert, and D. Stredney. Illustration motifs for effective medical volume illustration. *IEEE Computer Graphics and Applications*, 25(3):31–39, 2005. 2.2

[12] F. Tzeng, E. Lum, and K. Ma. An intelligent system approach to higher-dimensional classification of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 11(3):273–284, 2005. 1

[13] G. Weber, S. Dillard, H. Carr, V. Pascucci, and B. Hamann. Topology-controlled volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):330–341, 2007. 1