VOLUME HAPTICS TOOLKIT MANUAL

for VHTK 1.9 (H3D API 2.1)

< 1.x draft >

CONTENTS

CHAPTER 1

INTRODUCTION

This chapter provides an overview of what VHTK can provide to extend the uses of H3D API, beginning with some background on the topic of volume haptics and the context in which VHTK was born. It will be assumed that the reader is familiar with the concepts of haptic interaction, its components tactile and kinesthetic or proprioception and its roll in surgical training, virtual prototyping and other applications of virtual reality.

1.1 Volume Haptics

The typical modes of haptic interaction used in the various applications of virtual reality is based on the assumed presence of surfaces. The feedback producing algorithms aim to simulate the interaction with real objects that humans are so familiar with. Volume haptics, on the other hand, is concerned with the haptic interaction with volumetric data, that is data without explicit surface representations.

Some consider all haptic algorithms that use volumetric data as a type of volume haptics, even if the data is representing surface data and the feedback produce a surface impression. Examples can be seen in several methods for 6 degrees-of-freedom feedback in rapid prototyping and bone dissection simulators. Others limit themselves to consider the topic to cover only methods for handling haptic interaction with scientific volumetric data, such as medical data from Computer Tomography scanners (CT) and Magnetic Resonance scanners (MR), and parameter spaces in or results from computational simulations. While surfaces can be represented only in one unique way, just more or less accurately and convincingly, volumetric data can be represented in a large variety of forms, each conveying different properties of the data and guiding the haptic instrument to find and follow different shapes, features and directions contained in the data.

1.2 The Volume Haptics Toolkit

The Volume Haptics Toolkit, VHTK for short, was developed during a research project aiming at bringing haptics into volume data exploration interfaces and the volume data understanding process. During this project the algorithms needed for both simple and effective volume haptics were designed and the primary interface for VHTK was formed. The toolkit extends H3D API by introducing the scene-graph nodes necessary for loading volumetric data, handling and processing the data and for using the data to produce both visual and haptic feedback.

VHTK has been designed ground up to make full use of the features provided by H3D API. Thus, the toolkit can be use in all three design levels of H3D API — structural design using X3D, interactive dynamics using Python scripting and low-level setup and extensions in C++. Also the event handling system is highly integrated in the functionality of VHTK. Changes in fields controling, for example, filters propagates an event to every node involved, forming a conceptual multi-modal data pipeline, similar to those of *Visualization Toolkit*, VTK, and *AVS/Express*.

1.3 VHTK Technologies

VHTK was developed to provide volume haptics for both researchers and for high end volume exploration applications. It was designed to provide the same functionality as previous methods developed for volume interaction and to provide a powerful, yet easy to use, application programming interface (API).

To be able to provide both the haptic effects commonly used in volume haptics and a comprehensible and powerful programming interface, a new hierarchical approach to volume haptics programming has been introduced. A application programmer or a novice in volume haptics can use pre-fabricated haptic schemes, while a researcher or advanced programmer may set up their own schemes, from the provided low-level components. The achitecture is designed in a way so that the programmer needs never see the most low-level workings of the system, however not even the haptic solver is hidden from the programmer.

1.3.1 Haptic Modes

The haptic mode is the highest level of haptic interaction provided by VHTK. It is, in effect, a connection between a volumetric dataset and its haptic representation. Volumetric data can, unlike surfaces, be represented in many different ways, so VHTK provides a number of haptic modes, each giving a unique haptic representation of the volumetric data.

The system used in VHTK allows combination of haptic modes into more advanced interaction schemes. In interaction with multi-modal data, for example both blood flow and morphological data from MR in medicine, one haptic mode for each data modality can be used to provide a comprehensive haptic representation of the data at hand. Two or more modes can also be used to provide simultaneous complementary information.

Building an application from the pre-implemented haptic modes of VHTK requires no C++ programming. Using only X3D, a static haptic exploration of advanced scientific data can be implemented. User interfaces to load data and to select and configure both haptic and graphic rendering of can be implemented using Python-scripts.

1.3.2 Haptic Primitives

Haptic modes are in VHTK implemented by controlling parameters of *haptic primitives* as functions of the volumetric data that the haptic mode is representing. There are currently four primitives available the VHTK, one constraint for each dimensionality — point (3D), line (2D) and plane (1D) primitives — and one force primitive. Each primitive has a strength parameter, which specifies the strength of the feedback from the primitive. The force, line and plane primitives also have a direction parameter, which can be used to represent some vector feature in a haptic mode.

A haptic mode may use one or several haptic primitives to generate its specific haptic effect. The lowest-level system of VHTK prompts the haptic modes for the instant set of haptic primitives to momentarily represent the haptic feedback, at a rate of about 1 kHz. The haptic mode then reads off the local data properties and sets up one or more primitives to reflect these properties. These primitives are then returned to the system, which use the primitives to calculate the haptic feedback for that time-frame.

1.3.3 Haptic Solver

The innermost workings of VHTK is controlled by one or more numerical or analyhical solvers that converts the haptic primitives into a single force feedback. There are two types of solvers: general solvers and special case solvers. A special case solver first checks if the current primitives and primitive properties fulfill special prerequisites that allow the solver to more easily find the correct feedback. When all special case solvers have failed, the general solver is entrusted to find the solution for any primitives configuration.

While the solvers provided by VHTK should be enough for any application, a researcher may want to find an alternative solver to fit a application specific case and thereby find the solution more effectively.

1.3.4 Putting It Together

The primitives defined by the haptic modes are collected, from the haptics thread at approximately 1 kHz, by the heart of VHTK — the **VolumeHaptics** node. This node makes the necessary setup of variables for the haptic modes and post-processing of the resulting primitives, and feeds the primitives into the available solvers. It is the **VolumeHaptics** node that takes care of the transform interpolation, for dynamic transforms, and supports volume data animation. Thus, for the haptic modes, the data handling and primitives definition are performed on a static dataset in a static frame of reference, making it more easy to implement new haptic modes with full functionality.

There is also a possibility to modify the intermediate results from the solver using a “haptic shader”. Haptic shaders extends the **VHTKHapticShaderNode** node and implement one or several of three functions to modify the primitives, proxy position or the generated force, respectively. There is only one shader implemented at this moment and that is the **VisualHapticPrimitives** node, that provides a visual representation of the current haptic primitives and their configurations. It does not modify the configuration but only synchronizes the parameters between the haptic and the graphic thread.

1.4 Three Types of VHTK Nodes

The user of VHTK will find that there are three different, more or less distinctly separated, types of nodes, reflecting the different functions of the toolkit. These are volume data handling and value processing, visualization and haptic functionality. This section provides an overview of the most important features of the nodes of these areas.

1.4.1 Haptic Functionality

The toolkit encapsulates the steps forming the haptic behaviour into scenegraph nodes, thereby hiding the low-level processing and haptic primitives. The haptic nodes thus form a palette of modes that can be freely selected and combined to generate a wide array of different haptic schemes, allowing a developer to tailor the task specific haptic scheme of an application. Each node also provides an X3D interface to mode specific data and parameters. Analogous to visual models in visual scenegraphs, the transforms above the node affect the position and orientation of the haptic representation of the node's data source. Letting haptic nodes and visualization nodes share parent transform and data source thus provides co-located haptics and graphics.

Currently eight pre-implemented haptic modes are provided for representing features in both scalar and vector data. For some application areas and selected tasks the available predefined haptic modes, or combinations thereof, may not suffice to represent the most interesting features. If so, a fundamentally new haptic mode is needed. The low-level abstraction layer constituted by the haptic primitives is made available for the implementation of new haptic modes. New modes can easily be integrated into the toolkit framework by extending the abstract haptic node type and implementing the new node to provide haptic primitives describing the desired effect.

1.4.2 Volume Data Handling

Data sources and data filters are also implemented as scenegraph nodes. They provide a general data extraction interface for subsequent nodes to use and the filters differ from the other sources, such as readers, only in that their X3D interface allows the assignment of a source to read data from. The data handling structure allows for both analytical and sampled volume data and defines interfaces for extracting the basic features from scalar and vector data: scalar value, scalar gradient vector, vector value, vector curl vector and vector divergence. Among the filters provided by the toolkit are support for conversion between data types and extraction of the magnitude of vector features, such as vector curl, as scalar data for visual volume rendering or haptic feedback. By changing the filtering of the data used by a haptic mode its possible uses can be widely expanded. For example, in a related project a classification algorithm is used as filtering to enhance the haptic feedback.

To provide flexible and intuitive control of the scenegraph nodes, the toolkit makes extensive use of transfer functions. Filters for rescaling data use transfer functions to control the input/output conversion and both visual and haptic nodes use transfer functions to control material, colour and size properties.

There are, therefore, several different types of transfer function nodes available, providing different control interfaces. Examples are specification of piecewise linear segment and using the window function common in radiology.

1.4.3 Visualization

The toolkit provides functinality for intuitive visualization of the volumetric data. The main purpose of the toolkit is to provide an interface to advanced volume haptics, so only a few visualization nodes have, so far, been provided, such as volume rendering and stream-ribbons. To make it easy to setup a multi-modal visualization, the visualization nodes have similar functionality and behaviour as the haptic modes described above.

CHAPTER 2

GETTING STARTED

This chapter will help you get started with the toolkit. You will find instructions on how to download and install the package and how to find your way around the different part of the software.

2.1 Setting Up VHTK

Before you get started using VHTK you will first need to get started with the software to which VHTK is a toolkit — H3D API. Since there are no pre-compiled binaries available for VHTK, and you will have to compile it yourself, it is required that you to have installed and configured H3D API with headers and libraries. Once you got H3D API up and running you may download, setup, compile and install VHTK.

2.2 VHTK Command-line Output

VHTK produces a lot of output, primarily for debugging. This behaviour can be modified by changing the debug level (VHTK_DEBUG_LEVEL) to a lower value before compiling the toolkit, or by changing the output level of H3D API. Most messages can be safely ignored, simply informing about the current processes, but other might be important for an application developer.

The messages are divided the following levels:

I	Debug information	can be safely ignored.
W	Runtime warning	this means that there is something that is not configured or used as intended and that some features might not work as anticipated.
E	Runtime error	this message shows that something very wrong has happened that makes it impossible to continue execution. The program will terminate upon this kind of error.

CHAPTER 3

HAPTICS

CHAPTER 4

PYTHON SCRIPTS

The VHTK package comes bundled with some Python scripts to support the fast and easy setup of advanced multi-modal visualization without need for programming or scripting. This chapter describe the scripts and the measures needed to successfully integrate the scripts into a scene-graph.

4.1 ClipPlanes

ClipPlanes allows the user to interactively control clipplanes in a scene. A button click adds a clipplane at the current position with the current orientation. The plane can then be moved and rotated. Previously added planes can be interactively moved and rotated, and removed.

The following must be provided through the **references** field:

1. the group node which should be clipped

The following may also be provided through the **references** field:

1. a identically transformed group for clipplane icons,
2. an identically transformed LocalInfo node,
3. an icon for the clipplanes.

The following must be routed:

1. to the field **button**, the button to control the clipplanes with,
2. to the field **position**, the position to control the clipplanes, and
3. to the field **orientation**, the orientation to control the clipplanes.

4.2 PointsEditor

PointsEditor makes it possible to add, edit and remove points in 3D space, for example for specifying the distribution of glyphs in space. Pushing the button in free space adds a new point, pressing at a point

icon and moving the haptic instrument moves that point, and pressing at a point and then releasing the button removes the point.

The following must be provided through the **references** field:

1. the group node for the point icons.

The following may also be provided through the **references** field:

1. an identically transformed LocalInfo node, and
2. an icon for the points.

The following must be routed:

1. To the **button** field, the button to control the points with
2. To the **position** field, the position to control the points

The resulting points resides in the **point** field.

4.3 ProbeDisplay

ProbeDisplay opens a Tk window and displays information from a specified VolumeProbe.

The following must be provided through the **references** field:

1. the VolumeProbe instance from which to extract the data

This script only reads data from the **VolumeProbe** node and displays it. For the **VolumeProbe** to update the values some probe position must be routed to the **probe** field. See the documentation for **VolumeProbe** for more information.

4.4 PutStreamRibbons

PutStreamRibbons allows the user to interactively release stream ribbons in a specified vector volume.

The following *must* be provided through the **references** field:

1. the group node in which the stream ribbons should be put,
2. an appearance to use, and

3. a template **StreamRibbons** node

The following *must* be routed:

1. to the **button** field, the button to put the stream ribbons with and
2. to the **position** field, the seed position in the same coordinate system as the volume and the group node.

A KeySensor is used to provide the following commands

z undo the last stream tube

Z undo all stream tubes

1–9 the number of stream tubes to apply at once.

4.5 PutStreamTubes

This script is identical to PutStreamRibbons described above, but releases tubes instead of ribbons.

4.6 Rotator

This Python-script provides rotation information extracted from

1. arrow keys,
2. mouse motions, and
3. spaceware devices.

It can and should be used to control 3D scene orientation. This is done by routing the **rotation** field of this script to a **Transform** node.

CHAPTER 5

TUTORIALS

This chapter provides tutorials to show how to build an X3D setup file for the Volume Haptics Toolkit to create visualizations of volumetric data.

Prerequisites These tutorials assumes basic knowledge in H3D programming, especially through X3D, and experience in haptics, scientific visualization and volume visualization.

5.1 Dichloroethane Electro-potential

In this tutorial we use a analytical volume in the rendering of a dichloroethane molecule. The analythical volume is easy to use as a demonstration example, but the principles are similar for sampled volumes. We use the electro-potential field volume node provided by the toolkit to simulate and visualize the electro-potential of a hypothetical molecule, looking very much like a dichloroethane molecule.

Create Molecule Stick-model

We start by creating a simple stick model of the molecule we will explore. Create a top level group node and add a transform, so that we can scale our model to manageable sizes. Now create a shape node and add an indexed lineset node. Add to this node a coordinates node containing the coordinates for your atoms in the **point** field.

Now use the **coordIndex** field of the lineset node to link the atom points together. Also make sure you have the right number of colours for the coordinates. You may select colours to match the atom types or just use gray. The default line width is too slim, so add a line properties node and set the line width scale factor to around five.

Now add shapes for the atoms. Use a transform node to specify the position of the atoms, with coordinates the same as in the lineset node. Colours, specified through a material node in the appearance node for each shape, can indicate the atom type. Use **DEF** and **USE** to lower the level of redundant code, for example for the appearance node for multiple atoms of the same type.

We should now have a nice stick-model of a hypothetical (or real) molecule. Try loading the setup file and rotate the molecule. The size of the molecule will be controlled by the size of the coordinates used for the atom positions and the transform added under the top level group node.

Visualize Electro-potential

Here we will start using the nodes from VHTK, so we will need to load the VHTK library from the X3D file. Add a **ImportLibrary** node and specify the path for the VHTK library file. Depending on your platform the library will either be called 'libVHTK.so' or 'VHTK.dll'.

```
4   <ImportLibrary library="lib/libVHTK.so"/>
5   <ImportLibrary library="lib/VHTK.dll"/>
6   <Inline DEF="DEVICE"url="x3d/device.x3d"/>
```

When we built the stick model, we also specified the position of a number of atoms. By entering these positions into the electro-potential node we get a dataset representing the electro-potential over the molecule. To get the electro-potential also the potential of the points must be entered. Either make up some positive and negative potentials for your atoms in the molecule, or use real values. The potentials should sum up to zero. The values chosen in our demo give the following node setup.

```
148         <ScalarElectroPotential
149             size="7 7 7"
150             point="−0.30   0.37   0.60
151                    +0.30  −0.37  −0.60
152                    +2.07  −0.31  −0.50
153                    −2.07   0.31   0.50
154                    +0.03  −0.10   1.52
155                    +0.03   1.41   0.58
156                    −0.03   0.10  −1.52
157                    −0.03  −1.41  −0.58"
158             potential="+.4
159                        +.4
160                        −2
161                        −2
162                        +.8
163                        +.8
164                        +.8
165                        +.8"/>
```

First we'll visualize this using iso-surfaces and then using volume rendering.

The iso-surface node is a geometry node, so add it to a shape. Also add an appearance node and specify transparency, to avoid occlusion. Our electro-potential must be sampled for the iso-surface to be extracted, so we put the potential data node in a SampledScalarVolume that is put in the iso-surface node, so that its data is used. Create three iso-surface shapes at one positive iso-value, one negative and one at zero.

```
134     <Shape>
135       <Appearance DEF="SURFAPP">
136         <Material transparency=".7"/>
137       </Appearance>
138       <IsoSurface
139           solid="FALSE"
140           isoValue="-1">
141         <SampledScalarVolume
142             DEF="SVOLUME"
143             width="32"
144             height="32"
145             depth="32"
146             pixelComponentType="RATIONAL"
147             bitsPerPixel="32">
148           <ScalarElectroPotential
149               size="7 7 7"
150               point="-0.30   0.37   0.60
151                      +0.30  -0.37  -0.60
152                      +2.07  -0.31  -0.50
153                      -2.07   0.31   0.50
154                      +0.03  -0.10   1.52
155                      +0.03   1.41   0.58
156                      -0.03   0.10  -1.52
157                      -0.03  -1.41  -0.58"
158               potential="+.4
159                          +.4
160                          -2
161                          -2
162                          +.8
163                          +.8
164                          +.8
165                          +.8"/>
166         </SampledScalarVolume>
167       </IsoSurface>
168     </Shape>
```

Use **DEF** and **USE** to reuse both appearance and electro-potential.

At this point you should have iso-surfaces showing, at some points in space, how the electro-potential field relates to a molecule visualized with sticks and balls. To get a more continuous representation of the electro-potential, we'll use volume rendering. The volume renderer needs data as a 3D texture and needs rescaling. This is done using a **Texture3DVolume** node. Reuse the previously created scalar volume in

a texture volume and put this in a volume renderer node. The texture volume needs a transfer function to translate the electro-potential to the range 0–1. Use a **WindowFunction** with level zero, so that both negative and positive potentials are visualized. The zero potential will then end up at 0.5 in the texture. The **width** parameter of the window function specifies the contrast.

```
191     <VolumeRenderer
192         planes="100">
193       <Texture3DVolume
194           property="SCALAR">
195         <WindowFunction
196             level="0"
197             width="1"/>
198         <SampledScalarVolume
199             USE="SVOLUME"/>
200       </Texture3DVolume>
```

Apart from the texture to visualize, the volume renderer needs colour and opacity transfer functions. Add four transfer function, for example **PiecewiseFunction**, and see to it that they are added to container fields **scalar2red**, **scalar2green**, **scalar2blue** and **scalar2alpha**, respectively. Examples of parameters to use are **continuous** set to true and **segments** set to `"0.0 1.0, 0.5 0.0, 1.0 0.0"`, `"0.0 0.0, 0.5 1.0, 1.0 0.0"`, `"0.0 0.0, 0.5 0.0, 1.0 1.0"` and `"0.0 0.1, 0.5 0.0, 1.0 0.1"`, respectively.

This should give a volume visualization of the potential field. Adjust the number of planes in the volume renderer and the transfer functions to get the wanted result. Observe that both the volume renderer and the texture volume have transfer functions that affect the mapping from original potential data to colour and opacity.

Setup Haptic Feedback

Interactive Stream-ribbons

Final Code

```
1 <?xml version="1.0"encoding="UTF-8"?>
2 <Group>
3
4   <ImportLibrary library="lib/libVHTK.so"/>
5   <ImportLibrary library="lib/VHTK.dll"/>
6   <Inline DEF="DEVICE"url="x3d/device.x3d"/>
7
8   <IMPORT inlineDEF="DEVICE"exportedDEF="HDEV"AS="HDEV"/>
```

```
 9
10  <Background
11      skyColor="1.0 1.0 1.0"/>
12
13  <VolumeHaptics
14      stiffness="400"/>
15
16  <Transform
17      scale=".05 .05 .05">
18
19    <Group DEF="STREAMS_GROUP"/>
20    <LocalInfo DEF="INFO"/>
21
22
23    <!-- Sticks in the stick molecule model -->
24
25    <Shape>
26      <Appearance>
27        <LineProperties
28            linewidthScaleFactor="5"/>
29      </Appearance>
30      <IndexedLineSet
31          coordIndex="2 1 0 3 -1
32                      0 5 -1
33                      0 4 -1
34                      1 6 -1
35                      1 7 -1">
36        <Color
37            color=".2 .2 .2
38                    .2 .2 .2
39                    .8 .2 .8
40                    .8 .2 .8
41                    .8 .2 .2
42                    .8 .2 .2
43                    .8 .2 .2
44                    .8 .2 .2"/>
45        <Coordinate
46            point="-0.30   0.37   0.60
47                   +0.30  -0.37  -0.60
48                   +2.07  -0.31  -0.50
49                   -2.07   0.31   0.50
50                   +0.03  -0.10   1.52
```

```
51                    +0.03   1.41   0.58
52                    -0.03   0.10  -1.52
53                    -0.03  -1.41  -0.58"/>
54      </IndexedLineSet>
55    </Shape>
56
57
58    <!-- Balls in the stick molecule model -->
59
60    <Transform
61        translation="-0.30   0.37   0.60">
62      <Shape>
63        <Appearance DEF="C">
64          <Material
65              diffuseColor=".2 .2 .2"
66              specularColor=".8 .8 .8"/>
67        </Appearance>
68        <Sphere
69            DEF="SPHERE"
70            radius="0.2"/>
71      </Shape>
72    </Transform>
73    <Transform
74        translation="+0.30  -0.37  -0.60">
75      <Shape>
76        <Appearance USE="C"/>
77        <Sphere USE="SPHERE"/>
78      </Shape>
79    </Transform>
80    <Transform
81        translation="+2.07  -0.31  -0.50">
82      <Shape>
83        <Appearance DEF="Cl">
84          <Material
85              diffuseColor=".8 .2 .8"
86              specularColor=".5 .5 .5"/>
87        </Appearance>
88        <Sphere USE="SPHERE"/>
89      </Shape>
90    </Transform>
91    <Transform
92        translation="-2.07   0.31   0.50">
```

```
93        <Shape>
94          <Appearance USE="Cl"/>
95          <Sphere USE="SPHERE"/>
96        </Shape>
97      </Transform>
98      <Transform
99          translation="+0.03  -0.10   1.52">
100       <Shape>
101         <Appearance DEF="H">
102           <Material
103               diffuseColor=".8 .2 .2"
104               specularColor=".5 .5 .5"/>
105         </Appearance>
106         <Sphere USE="SPHERE"/>
107       </Shape>
108     </Transform>
109     <Transform
110         translation="+0.03   1.41   0.58">
111       <Shape>
112         <Appearance USE="H"/>
113         <Sphere USE="SPHERE"/>
114       </Shape>
115     </Transform>
116     <Transform
117         translation="-0.03   0.10  -1.52">
118       <Shape>
119         <Appearance USE="H"/>
120         <Sphere USE="SPHERE"/>
121       </Shape>
122     </Transform>
123     <Transform
124         translation="-0.03  -1.41  -0.58">
125       <Shape>
126         <Appearance USE="H"/>
127         <Sphere USE="SPHERE"/>
128       </Shape>
129     </Transform>
130
131
132     <!-- Iso surfaces  -->
133
134     <Shape>
```

```
135       <Appearance DEF="SURFAPP">
136         <Material transparency=".7"/>
137       </Appearance>
138       <IsoSurface
139           solid="FALSE"
140           isoValue="-1">
141         <SampledScalarVolume
142             DEF="SVOLUME"
143             width="32"
144             height="32"
145             depth="32"
146             pixelComponentType="RATIONAL"
147             bitsPerPixel="32">
148           <ScalarElectroPotential
149               size="7 7 7"
150               point="-0.30   0.37   0.60
151                      +0.30  -0.37  -0.60
152                      +2.07  -0.31  -0.50
153                      -2.07   0.31   0.50
154                      +0.03  -0.10   1.52
155                      +0.03   1.41   0.58
156                      -0.03   0.10  -1.52
157                      -0.03  -1.41  -0.58"
158               potential="+.4
159                          +.4
160                          -2
161                          -2
162                          +.8
163                          +.8
164                          +.8
165                          +.8"/>
166         </SampledScalarVolume>
167       </IsoSurface>
168     </Shape>
169
170     <Shape>
171       <Appearance USE="SURFAPP"/>
172       <IsoSurface
173           solid="FALSE"
174           isoValue="+1">
175         <SampledScalarVolume USE="SVOLUME"/>
176       </IsoSurface>
```

```
177     </Shape>
178
179     <Shape>
180       <Appearance USE="SURFAPP"/>
181       <IsoSurface
182           solid="FALSE"
183           isoValue="0">
184         <SampledScalarVolume USE="SVOLUME"/>
185       </IsoSurface>
186     </Shape>
187
188
189     <!-- Volume Rendering -->
190
191     <VolumeRenderer
192         planes="100">
193       <Texture3DVolume
194           property="SCALAR">
195         <WindowFunction
196             level="0"
197             width="1"/>
198         <SampledScalarVolume
199             USE="SVOLUME"/>
200       </Texture3DVolume>
201       <PiecewiseFunction
202           containerField="scalar2red"
203           continuous="TRUE"
204           segments="0.00  1.0
205                     0.50  0.0
206                     1.00  0.0"/>
207       <PiecewiseFunction
208           containerField="scalar2green"
209           continuous="TRUE"
210           segments="0.00  0.0
211                     0.50  1.0
212                     1.00  0.0"/>
213       <PiecewiseFunction
214           containerField="scalar2blue"
215           continuous="TRUE"
216           segments="0.00  0.0
217                     0.50  0.0
218                     1.00  1.0"/>
```

```
219       <PiecewiseFunction
220           containerField="scalar2alpha"
221           continuous="TRUE"
222           segments="0.00  0.10
223                     0.50  0.00
224                     1.00  0.10"/>
225     </VolumeRenderer>
226
227
228     <!-- Haptics -->
229
230     <VectorFollowMode
231         DEF="FOLLOW_MODE"
232         active="true">
233       <VectorElectroPotential
234           DEF="VVOLUME"
235           point="-0.30   0.37   0.60
236                  +0.30  -0.37  -0.60
237                  +2.07  -0.31  -0.50
238                  -2.07   0.31   0.50
239                  +0.03  -0.10   1.52
240                  +0.03   1.41   0.58
241                  -0.03   0.10  -1.52
242                  -0.03  -1.41  -0.58"
243           potential="+.4
244                      +.4
245                      -2
246                      -2
247                      +.8
248                      +.8
249                      +.8
250                      +.8"/>
251       <PiecewiseFunction
252           containerField="strength"
253           DEF="STRENGTH"
254           continuous="TRUE"
255           segments="0.00  0.0
256                     0.30  1.0
257                     1.00  2.0
258                     1000  5.0"/>
259     </VectorFollowMode>
260
```

```
261     <VectorFrontShapeMode
262         DEF="FRONT_SHAPE_MODE"
263         active="false"
264         twoSided="true"
265         snapdrag="true">
266       <VectorElectroPotential
267           USE="VVOLUME"/>
268       <PiecewiseFunction
269           containerField="strength"
270           USE="STRENGTH"/>
271     </VectorFrontShapeMode>
272
273   </Transform>
274
275
276   <PythonScript DEF="PUT"url="python/PutStreamTubes.py">
277
278     <Group USE="STREAMS_GROUP"containerField="references"/>
279
280     <Appearance
281         containerField="references">
282       <Material
283           ambientIntensity="1"/>
284     </Appearance>
285
286     <StreamTubes
287         containerField="references"
288         useEuler="FALSE"
289         step="0.1"
290         maxLength="5.0">
291       <VHTKVectorDataNode USE="VVOLUME"/>
292       <MagnitudeVolume>
293         <VHTKVectorDataNode USE="VVOLUME"/>
294         <WindowFunction
295             level="1"
296             width="2"/>
297       </MagnitudeVolume>
298       <MagnitudeVolume
299           containerField="radiusVolume">
300         <VHTKVectorDataNode USE="VVOLUME"/>
301         <WindowFunction
302             level="1"
```

```
303                 width="2"
304                 roof=".2"/>
305             </MagnitudeVolume>
306         </StreamTubes>
307
308     </PythonScript>
309     <ROUTE
310         fromNode="HDEV"fromField="mainButton"
311         toNode="PUT"toField="putStreamTubes"/>
312     <ROUTE
313         fromNode="HDEV"fromField="trackerPosition"
314         toNode="INFO"toField="position"/>
315     <ROUTE
316         fromNode="INFO"fromField="position"
317         toNode="PUT"toField="putStreamTubes"/>
318
319 </Group>
```