HAVE'2007 - IEEE International Workshop on
Haptic Audio Visual Environments and their Applications
Ottawa, Canada 12-14 October 2007

Extending Blender: Development of a Haptic Authoring Tool

Sheldon Andrews[1], Mohamad Eid[2], Atif Alamri[2], and Abdulmotaleb El Saddik[2]
Multimedia Communications Research Laboratory - MCRLab
School of Information Technology and Engineering - University of Ottawa
Ottawa, Ontario, K1N 6N5, Canada
[1]sandr071@site.uottawa.ca, [2]{eid, atif, abed} @ mcrlab.uottawa.ca

Abstract *–In this paper, we present our work to extend a well known 3D graphic modeler – Blender – to support haptic modeling and rendering. The extension tool is named HAMLAT (Haptic Application Markup Language Authoring Tool). We describe the modifications and additions to the Blender source code which have been used to create HAMLAT. Furthermore, we present and discuss the design decisions used when developing HAMLAT, and also an implementation "road map" which describes the changes to the Blender source code. Finally, we conclude with discussion of our future development and research avenues.*

Keywords *– Haptics, HAML, Graphic Modelers, Blender, Virtual Environments.*

I. INTRODUCTION

A. Motivation

The increasing adoption of haptic modality in human-computer interaction paradigms has led to a huge demand for new tools that help novice users to author and edit haptic applications. Currently, the haptic application development process is a time consuming experience that requires programming expertise. The complexity of haptic applications development rises from the fact that the haptic application components (such as the haptic API, the device, the haptic rendering algorithms, etc.) need to interact with the graphic components in order to achieve synchronicity. Additionally, there is a lack of application portability as the application is tightly coupled to a specific device that necessitates the use of its corresponding API. Therefore, device and API heterogeneity lead to the fragmentation and disorientation of both researchers and developers. In view of all these considerations, there is a clear need for an authoring tool that can build haptic applications while hiding programming details from the application modeler (such as API, device, or virtual model).

This paper describes the technical development of the Haptic Application Markup Language Authoring Tool (HAMLAT). It is intended to explain the design decisions used for developing HAMLAT and also provides an implementation "road map", describing the source code of the project.

B. Blender

HAMLAT is based on the Blender [1] software suite, which is an open-source 3D modeling package with a rich feature set. It has a sophisticated user interface which is noted for its efficiency and flexibility, as well as its supports for multiple file formats, physics engine, modern computer graphic rendering and many other features.

Because of Blender's open architecture and supportive community base, it was selected as the platform of choice for development of HAMLAT. The open-source nature of Blender means HAMLAT can easily leverage its existing functionality and focus on integrating haptic features which make it a complete hapto-visual modeling tool, since developing a 3D modeling platform from scratch requires considerable development time and expertise in order to reach the level of functionality of Blender. Also, we can take advantage of future improvements to Blender by merging changes from its source code into the HAMLAT source tree.

HAMLAT builds on existing Blender components, such as the user-interface and editing tools, by adding new components which focus on the representation, modification, and rendering of haptic properties of objects in a 3D scene. By using Blender as the basis for HAMLAT, we hope to develop a 3D haptic modeling tool which has the maturity and features of Blender combined with the novelty of haptic rendering.

At the time of writing, HAMLAT is based on Blender version 2.43 source code.

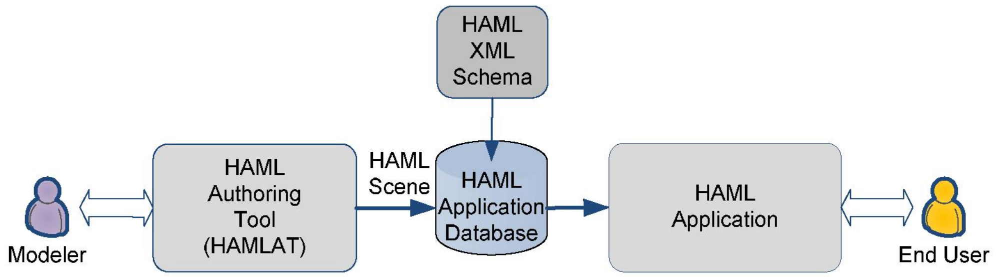


Figure 1: A block diagram illustrating the haptic modeling pipeline

C. Project Goals

As previously stated, the overall goal for the HAMLAT project is to produce a polished software application which combines the features of a modern graphic modeling tool with haptic rendering techniques. HAMLAT has the "look and feel" of a 3D graphical modeling package, but with the addition of features such as haptic rendering and haptic property descriptors. This allows artists, modelers, and developers to generate realistic 3D hapto-visual virtual environments.

A high-level block diagram of HAMLAT is shown in Figure 1. It illustrates the flow of data in the haptic modeling. HAMLAT assists the modeler, or application developer, in building hapto-visual applications which may be stored in a database for later retrieval by another haptic application. By hapto-visual application we refer to any software which displays a 3D scene both visually and haptically to a user in a virtual setting. An XML file format, called HAML [2], is used to describe the 3D scenes and store the hapto-visual environments built by a modeler for later playback to an end user.

Traditionally, building hapto-visual environments has required a strong technical and programming background. The task of haptically rendering a 3D scene is tedious since haptic properties must be assigned to individual objects in the scene and currently there are few high-level tools for accomplishing this task. HAMLAT bridges this gap by integrating into the HAML framework and delivering a complete solution for development of hapto-visual applications requiring no programming knowledge.

The remainder of the paper is organized as follows: in Section 2, we present the proposed architecture extensions and discuss design constraints. Section 3 describes the implementation details and how haptic properties are added and rendered within the Blender framework. In Section 4 we discuss related issues and future work avenues.

II. SYSTEM OVERVIEW AND ARCHITECTURE

The Blender design philosophy is based on three main tasks: data storage, editing, and visualization. According to the legacy documentation [3], it follows a *data-visualize-edit* development cycle for the 3D modeling pipeline. A 3D scene is represented using data structures within the Blender architecture. The modeler views the scene, makes changes using the editing interface which directly modifies the underlying data structures, and then the cycle repeats.

To better understand this development cycle, consider the representation of a 3D object in Blender. A 3D object may be represented by an array of vertices which have been organized as a polygonal mesh. Users may choose to operate on any subset of this data set. Editing tasks may include operations to rotate, scale, and translate the vertices, or perhaps a re-meshing algorithm to "cleanup" redundant vertices and transform from a quad to a triangle topology. The data is visualized using a graphical 3D renderer which is capable of displaying the object as a wireframe or as a shaded, solid surface. The visualization is necessary in order to see the effects of editing on the data. In a nutshell, this example defines the design philosophy behind Blender's architecture.

In Blender, data is organized as a series of lists and base data types are combined with links between items in each list, creating complex scenes from simple structures. This allows data elements in each list to be reused, thus reducing the overall storage requirements. For example, a mesh may be linked by multiple scene objects, but the position and orientation may change for each object and the topology of the mesh remains the same. A diagram illustrating the organization of data structures and reuse of scene elements is shown in Figure 2. A scene object links to three objects, each of which link to two polygonal meshes. The meshes also share a common material property. The entire scene is rendered on one of several screens, which visualizes the scene.

We adopt the Blender design approach for our authoring tool. The data structures which are used to represent objects in a 3D scene have been augmented to include fields for haptic properties (e.g., stiffness, damping); user interface components (e.g., button panels) which allow the modeler to change object properties have also been updated to include support for modifying the haptic properties of an object. Additionally, an interactive hapto-visual renderer has been implemented to display the 3D scene graphically and haptically, providing the modeler or artist with immediate feedback about the changes they make to the scene.

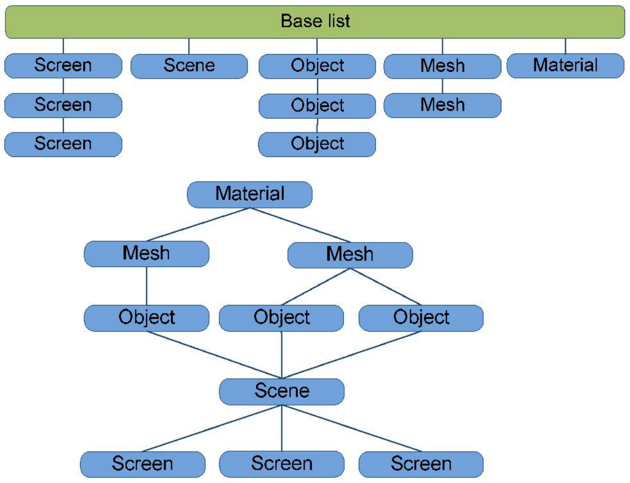


Figure 2: The organization of base data types in Blender. Data structures are stored in linked lists (top) with interconnections between them to create complex 3D scenes (bottom).

In the current version of the HAMLAT, the modifications to the Blender framework include:

- data structures for representing haptic properties,
- an editing interface for modifying haptic properties,
- an external renderer for displaying and previewing haptically enabled scenes,
- scripts which allow scenes to be imported/exported in the HAML file format.

A class diagram outlining the changes to the Blender framework is shown in Figure 3. Components which are pertinent to HAMLAT are shaded in gray. HAMLAT builds on existing Blender sub-systems by extending them for haptic modeling purposes. Data structures for representing object geometry and graphical rendering are augmented to include fields which encompass the tactile properties necessary for haptic rendering.

To allow the user to modify haptic properties, GUI components are integrated as part of the Blender editing panels. The operations triggered by these components operate directly on the data structures used for representing haptic cues and may be considered part of the editing step of the Blender design cycle.

Similarly to the built-in graphical renderer, HAMLAT uses a custom renderer for displaying 3D scenes graphically and haptically, and is independent of the Blender renderer. This component is developed independently since haptical and graphical rendering must be performed simultaneously and synchronously. A simulation loop is used to update haptic rendering forces at a rate which maintains stability and quality. A detailed discussion of the implementation of these classes and their connectivity is given in the next section.

III. IMPLEMENTATION

A. Data Structure

A.1 Mesh Data Type

Blender uses many different data structures to represent the various types of objects in a 3D scene— a polygonal mesh contains the location and topology of vertices; a lamp contains colour and intensity values; and a camera object contains intrinsic viewing parameters.

The *Mesh* data structure is used by the Blender framework to describe a polygonal mesh object. It is of particular interest for haptic rendering since many solid objects in a 3D scene may be represented using this type of data structure. The tactile and kinesthetic cues, which are displayed due to interaction with virtual objects, are typically rendered based on the geometry of the mesh. Haptic rendering is performed based primarily on data stored in this data type. Other scene components such as lamps, cameras, or lines are not intuitively rendered using force feedback haptic devices and are therefore not of current interest for haptic rendering.

An augmented version of the Mesh data structure is shown in Figure 4. It contains fields for vertex and face data, plus some special *custom data* fields which allow data to be stored to/retrieved from disk and memory. We have modified this data type to include a pointer to a *MHaptics* data structure, which stores haptic properties such as stiffness, damping, and friction for the mesh elements (Figure 5).

A.2 Edit Mesh Data Type

It should be noted that the Mesh data type has a complimentary data structure, called *EditMesh*, which is used when editing mesh data. It holds a copy of the vertex, edge, and face data for a polygonal mesh. When the user switches to editing mode, the Blender copies the data from a Mesh into an EditMesh and when editing is complete the data is copied back.

Care must be taken to ensure that the haptic property data structure remains intact during the copy sequence. The EditMesh data structure has not been modified to contain a copy of the haptic property data, but this may

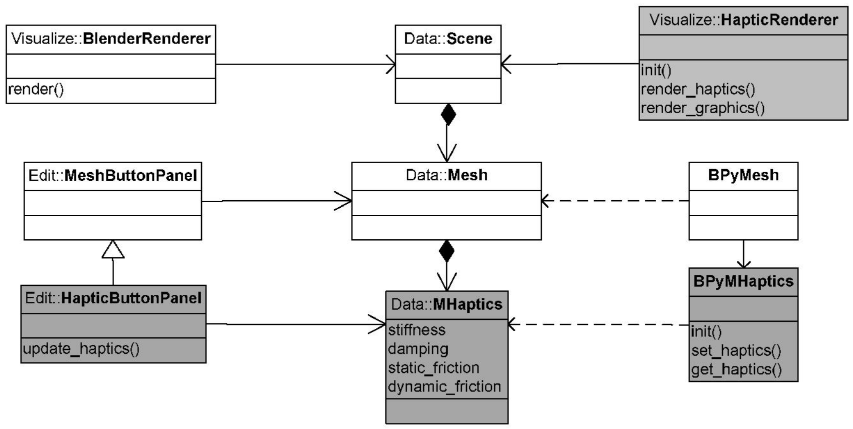


Figure 3: Class diagram of modifications to the Blender framework (Components added for HAMLAT are grayed)

```
    typedef struct Mesh {
        MFace *face;
        MVert *vert;
        CustomData vdata, fdata, hdata;
        MHaptics *haptics;
        …
    } Mesh;
```

Figure 4: Augmented Mesh data structure

```
    typedef struct MHaptics {
        float stiffness;
        float damping;
        float st_friction;
        float dy_friction;
     } MHaptics;
```

Figure 5: The haptic property data structure.

change in future versions (if modification of haptic properties in edit mode is required). The editing mode is mainly used to modify mesh topology and geometry, not the haptic and graphical rendering characteristics.

A.3 Haptic Properties

In this section we'll briefly discuss the haptic properties which may currently be modeled using HAMLAT. It is important for the modeler to understand these properties and their basis for use in haptic rendering.

The stiffness of an object defines how resistant it is to deformation by some applied force. Hard objects, such as a rock or table, have very high stiffness; soft objects, such as rubber ball, have low stiffness. The hardness-softness of an object is typically rendered using the spring-force equation:

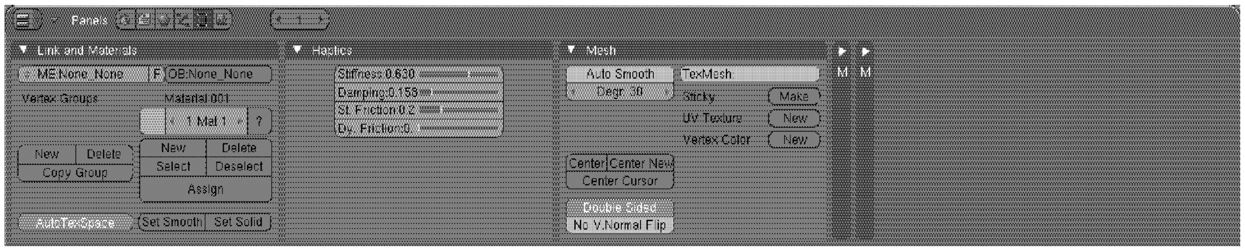
$$\vec{f} = -k_s \vec{x} \qquad (1)$$

Where the force feedback vector *f* which is displayed to the user is computed using k_s the stiffness coefficient (variable name *stiffness*) for the object and *x* the penetration depth (displacement) of the haptic proxy into an object. The stiffness coefficient has a range of *[0,1]*, where 0 represents no resistance to deformation and 1 represents the maximum stiffness which may be rendered by the haptic device. The damping of an object defines its resistance to the rate of deformation due to some applied force. It is typically rendered using the force equation:

$$\vec{f} = -k_d \frac{d\vec{x}}{dt} \qquad (2)$$

Where k_d is the damping coefficient (variable name *damping*) and *dx/dt* is the velocity of the haptic proxy as it penetrates an object. The damping coefficient also has a range of *[0,1]* and may be used to model viscous behaviour of a material. It also increases the stability of the haptic rendering loop for stiff materials.

The static friction (variable name *st_friction*) and dynamic friction (variable name *dy_friction*) coefficient are used to model the frictional forces experienced while exploring the surface of a 3D object. Static friction is experienced when the proxy is not moving over the object's surface, and an initial force must be used to overcome static friction. Dynamic friction is felt when the proxy moves across the surface, rubbing against it. Frictional coefficients also have a range of *[0,1]*, with a value of 0 making the surface of a 3D object feel "slippery" and a value of 1 making the object feel very rough. Frictional forces are typically rendered in a direction tangential to the collision point of the haptic proxy at an object's surface.



ME:None_None
OB:None_None
Material.001
1 Mat 1
Stiffness:0.630
Damping:0.158
St. Friction:0.2
Dy. Friction:0
Degr: 30

Figure 6: Blender's button space, including the haptic property editing panel

B. Editing

Blender uses a set of non-overlapping windows called *spaces* to modify various aspects of the 3D scene and its objects. Each space is divided into a set of areas and panels which are context aware. That is, they provide functionality based on the selected object type. For example, if a camera is selected the panel will display components which allow the modeler to change the focal length and viewing angle of the camera, but these components will not appear if an object of another type is selected.

Figure 6 shows a screen shot of the button space which is used to edit properties for a haptic mesh. It includes user-interface panels which allow a modeler to change the graphical shading properties of the mesh, perform simple re-meshing operations, and to modify the haptic properties of the selected mesh.

HAMLAT follows the context-sensitive behavior of Blender by only displaying the haptic editing panel when a polygonal mesh object is selected. In the future, this panel may be duplicated to support haptic modeling for other object types, such as NURB surfaces.

The Blender framework offers many user-interface components (e.g., buttons, sliders, pop-up menus) which may be used to edit the underlying data structures. The haptic properties for mesh objects are editable using sliders or by entering a float value into a text box located adjacent to the slider. When the value of the slider/text box is changed, it triggers an event in the Blender windowing sub-system. A unique identifier indicates that the event is for the haptic property panel and the HAMLAT code should be called to update haptic properties for the currently selected mesh.

C. Hapto-Visual Rendering

Blender currently supports graphical rendering of scenes using an internal renderer or an external renderer (e.g., [4]). In this spirit, the haptic renderer used by HAMLAT has been developed as an external renderer. It uses the OpenGL and OpenHaptics toolkit [5] to perform graphic and haptic rendering, respectively.

The 3D scene which is being modeled is rendered using two passes: the first pass renders the scene graphically, and the second pass renders it haptically. The second pass is required because the OpenHaptics toolkit intercepts commands send to the OpenGL pipeline and uses them to display the scene using haptic rendering techniques. In this pass, the haptic properties of each mesh object are used much in the same way color and lighting are used by graphical rendering—they define the type of material for each object. To save CPU cycles, the lighting and graphical material properties are excluded from the haptic rendering pass.

Figure 7 shows source code which is used to apply the material properties during the haptic rendering pass. The haptic renderer is independent from the Blender framework in that it exists outside the original source code. However, it is still heavily dependent on Blender data structures and types.

D. Scripting

The Blender Python (BPy) wrapper exposes many of the internal data structures, giving the internal Python scripting engine may access them. Similar to the data structures used for representing mesh objects in the native Blender framework, wrappers allow user defined scripts to access and modify the elements in a 3D scene.

```
hlMaterialf(HL_FRONT_AND_BACK,
    HL_STIFFNESS,
    haptics->stiffness);
hlMaterialf(HL_FRONT_AND_BACK,
    HL_DAMPING,
    haptics->damping);
hlMaterialf(HL_FRONT_AND_BACK,
    HL_STATIC_FRICTION,
    haptics->st_friction);
hlMaterialf(HL_FRONT_AND_BACK,
    HL_DYNAMIC_FRICTION,
    haptics->dy_friction);
```

Figure 7: Source code for applying haptic properties of a mesh using the OpenHaptics toolkit

The haptic properties of a mesh object may be accessed through the Mesh wrapper classes. A haptics attribute has been added to each of these classes and allows the haptic properties of a mesh object to be accessed through the Python scripting system. Figure 8 shows Python code to read the haptic properties from a mesh object and export to a file. Similar code is used to import/export HAML scenes from/to files.

An import script allows 3D scenes to be read from a HAML file and reproduced in the HAMLAT application; an export script allows 3D scenes to be written to a HAML file, including haptic properties, and used in other HAML applications.

```
def exportHaptics(filename,scene):
  file = open(filename,'w');
  obs = scene.getChildren();
  for ob in obs:
    na = ob.name;
    me = ob.data;
    ha = me.haptics;
    st = ha.stiffness;
    da = ha.damping;
    file.write(na+'%d,%d'%(st,da));
 file.close();
```

Figure 8: Example of an export script which uses the BPy wrappers to access haptic properties of mesh objects.

The BPy wrappers also expose the Blender windowing system. Figure 9 shows a panel which appears when the user exports a 3D scene to the HAML file format. It allows the user to specify supplementary information about the application such as a description, target hardware, and system requirements. These are fields defined by the HAML specification [2] and are included with the authored scene as part of the HAML file format. User-interface components displayed on this panel are easily extended to agree with the future revisions of HAML.

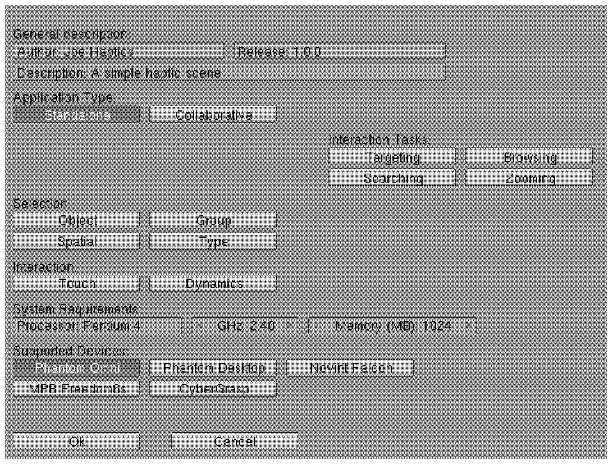

General description:
Author: Joe Haptics
Release: 1.0.0
Description: A simple haptic scene
Application Type: Standalone
System Requirements:
Processor: Pentium 4
GHz: 2.40
Memory (MB): 1024
Supported Devices: Phantom Omni

Figure 9: HAML export panel

IV. CONCLUSION AND FUTURE WORK

The current version of HAMLAT shows that a unified modeling tool for graphics and haptics is possible. Promisingly, the features for modeling haptic properties have been integrated seamlessly into the Blender framework, which indicates it was a good choice as a platform for development of this tool. Blender's modular architecture will make future additions to its framework very straightforward.

Currently, HAMLAT supports basic functionality for modeling and rendering hapto-visual applications. Scenes may be created, edited, previewed, and exported as part of a database for use in by other hapto-visual applications, such as the HAML player [6]. However, there is room for growth and in there are many more ways we can continue leveraging existing Blender functionality.

As per future work, we plan to extend HAMLAT to include support for other haptic platforms and devices. Currently, only the PHANTOM series of devices is supported since the interactive renderer is dependent on the OpenHaptics toolkit [5]. In order to support other devices, a cross-platform library such as Chai3D or Haptik may be used to perform rendering. These libraries support force rendering for a large range of haptic hardware. Fortunately, due to the modularity of our implementation, only the interactive haptic rendering component need be altered for these changes.

In addition to support multiple hardware platforms, a user interface component which allows the selection and configuration of haptic devices will be important. Most likely, this will be added as part of the user preferences panel in Blender.

Adding support for haptic devices as part of editing tasks is also a planned feature. This will allow the modeler to modify the shape, location, and other properties on in-scene objects. For example, the sculpting mode in Blender allows a user to manipulate the geometry of a 3D object using a natural interface, similar to reshaping a piece of clay. HAMLAT will build on this technology by allowing the modeler to manipulate the virtual clay using high DOF haptic interfaces.

REFERENCES

[1] Blender organization, “Blender official website,” http://www.blender.org, September 2007.

[2] F. R. El-Far, M. Eid, M. Orozco, A. El Saddik, “Haptic Application Meta-Language,” DS-RT, Malaga, Spain, 2006.

[3] Blender Organization, “Blender Architecture,” http://www.blender.org/development/architecture, September 2007

[4] YafRay, “Free Raytracing for the Masses,” http://www.yafray.org, September 2007.

[5] SensAble Technologies, Inc. “OpenHaptics Toolkit,” http://www.sensable.com/products-openhaptics-toolkit.htm, September 2007.

[6] M. Eid, M. Mansour, R. Iglesias, A. El Saddik. “A Device Independent Haptic Player,” IEEE Intl. Conference on Virtual Environments, Human-Computer Interfaces, and Measurement Systems (VECIMS 2007), Italy, 2007.

[7] F. Conti, F. Barbagli, D. Morris, C. Sewell, “CHAI: An Open-Source Library for the Rapid Development of Haptic Scenes,” Demo paper at IEEE World Haptics, Pisa, Italy, March 2005.