HAMLAT: A HAML-based Authoring Tool for Haptic Application Development

Mohamad Eid[1], Sheldon Andrews[2], Atif Alamri[1], and Abdulmotaleb El Saddik[2]

Multimedia Communications Research Laboratory (MCRLab)
University of Ottawa, Canada
Email: [1] sandr071@site.uottawa.ca, [2] {eid, atif, abed}@mcrlab.uottawa.ca

Abstract. Haptic applications have received enormous attention in the last decade. Nonetheless, the diversity of haptic interfaces, virtual environment modeling, and rendering algorithms have made the development of hapto-visual applications a tedious and time consuming task that requires significant programming skills. To tackle this problem, we present a HAML-based Authoring Tool (HAMLAT), an authoring tool based on the HAML description language that allows users to render the graphics and haptics of a virtual environment with no programming skills. The modeler is able to export the developed application in a standard HAML description format. The proposed system comprises three components: the authoring tool (HAMLAT), the HAML engine, and the HAML player. The tool is implemented by extending the 3D Blender modeling platform to support haptic interaction. The demonstrated tool proves the simplicity and efficiency of prototyping haptic applications for non-programmer developers or artists.

Keywords: Haptic authoring tools, HAML, 3D modeling

1 Introduction

The rapid adoption of haptic interfaces in human-computer interaction paradigms has led to a huge demand for new tools and systems that enable novice users to author, edit, and share haptic applications [1]. Nonetheless, the haptic application development process remains a time consuming experience that requires programming expertise. Additionally, assigning material properties (such as the stiffness, static friction, and dynamic friction) is a tedious and non-intuitive task since it requires the developers to possess technical knowledge about haptic rendering and interfaces.

The haptic and graphical rendering pipelines should remain synchronized to exhibit realistic and stable simulation. 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. In view of these considerations, there is a clear need for an authoring tool that can build hapto-visual applications while hiding programming details from the application modeler (such as API, device, or virtual model). This is achieved using standard XML-based descriptions that make these components self-described.

The idea of having a framework that facilitates the development of haptic applications has found significant interest from both the research and industry communities. One research prototype is Unison [2], a viable and extensible framework to standardize the development process of hapto-visual applications. It's main limitation is that the interface must be hard coded into a plug-in before a component becomes usable by the framework. The Haptik Library [3] proposes a hardware abstraction layer to provide uniform access to haptic devices. However, the library does not support higher level behavior (such as collision detection and response) and thus significant programming effort is still required. CHAI 3D is a open source set of C++ libraries for developing real time, interactive haptic and visual simulations [4], but it requires significant programming knowledge and skills. Other research efforts towards building haptic authoring tools can be found in [5][6].

Commercially, HANDSHAKE VR Inc. [7] introduced the ProSENSE toolbox which provides rapid creation of simulation content and includes tele-haptic capabilities. Reachin Technologies [8] introduced an object-oriented development platform for haptic applications that supports graphic and haptic rendering. However, the platform does not have a graphic/haptic editor to build the graphic and haptic scenes. SensAble introduced Claytools [9] and FreeForm systems [10] to incorporate haptics in the process of creating and modifying 3D objects. Though no programming is necessary, the workflow for these tools is complex and often requires additional modeling tools.

The goal of the HAMLAT project is to produce a software application that 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 that allows developers to generate realistic 3D hapto-visual environments. HAMLAT is based on the Haptic Applications Meta Language (HAML) [11] to describe the 3D scene, dynamic characteristics, haptic interface, network configurations, etc. The application can be exported in HAML format so that other users can reload the application to view, touch, and manipulate the populating objects.

The remainder of the paper is organized as follows: in Section 2, we introduce the authoring tool architecture and discuss its comprising components and their respective responsibilities. Section 3 presents the implementation details and the evaluation of the current state of the tool. Finally, in Section 4 we highlight known issues and possible future research avenues.

2 HAMLAT System Architecture

A conceptual overview of HAMLAT is shown in Figure 1. The diagram illustrates the flow of data in the hapto-visual modeling pipeline. A "hapto-visual" application refers to any software that displays a 3D scene both visually and haptically to a user in a virtual setting. The objective is to automate the haptic application development process giving the ability to compose and render hapto-visual applications with no programming efforts. The application artist can export a standard HAML description file and store it in a database. The HAML player, similarly to known audio/video players, recreates the hapto-visual environment by parsing the HAML file.

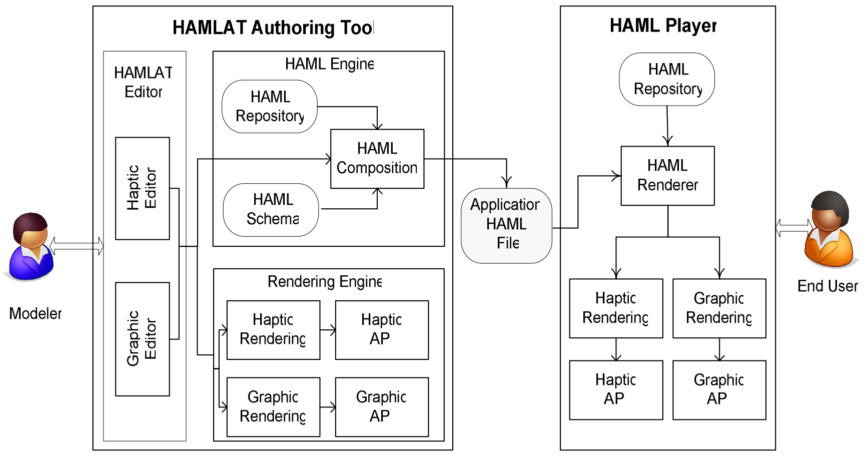


Fig. 1. A conceptual overview of the HAMLAT Authoring Tool

2.1 HAML Description

HAML is designed to provide a technology-neutral description of haptic models [12]. It describes the graphics (including the geometry and scene descriptions), haptic rendering, haptic devices (the hardware requirements), and application information. In other words, HAML is the standard by which haptic application components such as haptic devices, haptic APIs, or graphic models make themselves and their capabilities known.

There have been at least three foreseeable approaches to implementing and utilizing HAML documents: (1) application description that defines description schemes for various haptic application components that, given similar requirements, can be reused to compose similar applications, (2) component description where the HAML file describes the device/API/model via a manual, semi-automatic or automatic extraction, and (3) hapto-visual application authoring and/or composition. The scope of this research is focused on the third approach.

The HAML schema is instantiated for compatibility with MPEG-7 standard through the use of Description Schemes (DS). As explained in [12], the HAML structure is divided into seven description schemes: application description, haptic device description, haptic API description, haptic rendering description, graphic rendering description, quality of experience description, and haptic data description. An excerpt of a HAML document is shown in Figure 2.

2.2 HAMLAT Authoring Tool

The HAMLAT authoring tool is composed of three components: the HAMLAT editor, the HAML engine, and the rendering engine (Figure 1). The HAMLAT editor provides a GUI that enables environment modelers to create and import virtual objects, enable or disable haptic interaction, and assign haptic properties to the selected object(s). The graphic editor enables users to modify graphical properties of objects in the application (such as colors and shading). However, haptic editing is central to the HAMLAT tool. Once the application environment and objects are created, various haptic attributes (such as stiffness, damping, and friction) may be assigned in the same way visual or geometric properties are modified. Also, through the HAMLAT editor, the user is able to specify application parameters, such as the target haptic device and the developer information.

```
<?xml version="1.0" ?>
<HAML>
    <ApplicationDS>...</ApplicationDS>
    <AuthorDS>...</AuthorDS>
    <SystemDS>...</SystemDS>
    <SceneDS>
        <Object>
             <Type>...</Type>
             <Name>...</Name>
             <Location>...</Location>
             <Rotation>...</Rotation>
             <Geometry>
                 <VertexList>
                         <Vertex>...</Vertex>
                 </VertexList>
                 <FaceList>
                         <Face>...</Face>
                 </FaceList>
             </Geometry>
             <Appearance>
                 <Material>...</Material>
             </Appearance>
             <Tactile>
                 <Stiffness>...</Stiffness>
                 <Damping>...</Damping>
                 <SFriction>...</SFriction>
                 <DFriction>...</DFriction>
             </Tactile>
        </Object>
    </SceneDS>
</HAML>
```

Fig. 2. An excerpt from a HAML document

The HAML engine is responsible for generating a HAML file that fully describes the environment and through which the same environment can be reconstructed. Therefore, the HAML-formatted document, which holds the default settings of the haptic application, links the HAMLAT tool to the HAML player. Each HAML file generated by HAMLAT represents a stand-alone application that is device and platform independent. The HAML player is responsible for 'playing back' the HAML file generated by the authoring tool. The HAML renderer (refer to HAML Player in Figure 3) parses the HAML file to automatically map the application description with the available resources. Subsequently, the HAML renderer invokes the appropriate haptic and graphic rendering systems and displays through their APIs.

3 HAMLAT Implementation

The basis for our haptic authoring tool is the Blender open source project [13]. Blender includes a full-fledged 3D graphical renderer, an integrated scripting engine, a physics and game engine, and an adaptive user-interface. For these reasons, Blender was chosen as the platform for development of HAMLAT. Figure 3 shows a snapshot of the HAMLAT authoring tool with haptic rendering. The modeler is able to “feel” the physical properties of the rhinoceros as s/he moves the proxy over its surface.

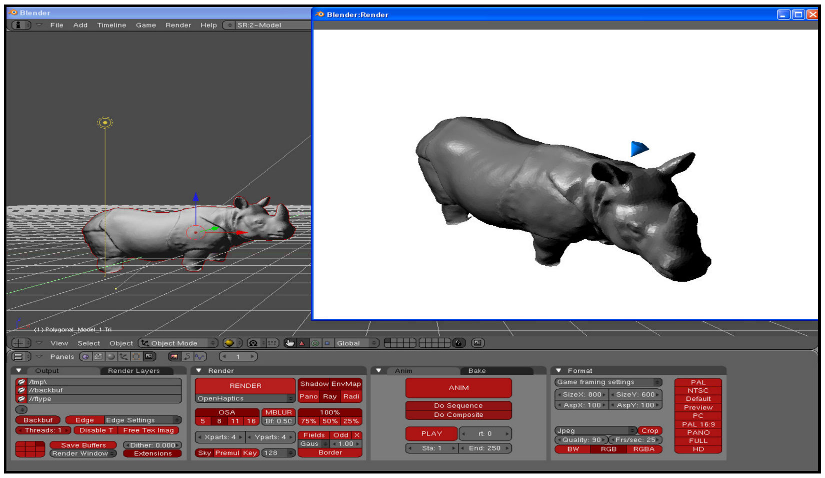


Fig. 3. A snapshot of the Blender-based HAMLAT editor with the haptic renderer

In HAMLAT, the modifications to the Blender framework include:

- extensions to Blender data structures for representing haptic properties
- user interface components for adding and modifying haptic properties
- an external renderer for displaying and previewing haptically enabled scenes
- scripts which allow scenes to be imported / exported in the HAML format

The current implementation is limited to static scenes. In other words, HAMLAT does not support dynamic contents such as animations. This is envisioned as one of our immediate future work. Also, multi-user rendering as well as network capabilities are not supported at the current stage of implementation.

A class diagram outlining the changes to the Blender framework is shown in Figure 4. Components which are pertinent to HAMLAT are shaded in gray. Data structures for representing object geometry and graphical rendering have been augmented to include fields which encompass the tactile and kinesthetic properties necessary for haptic rendering. 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 haptic and graphic rendering must be performed simultaneously and synchronously.

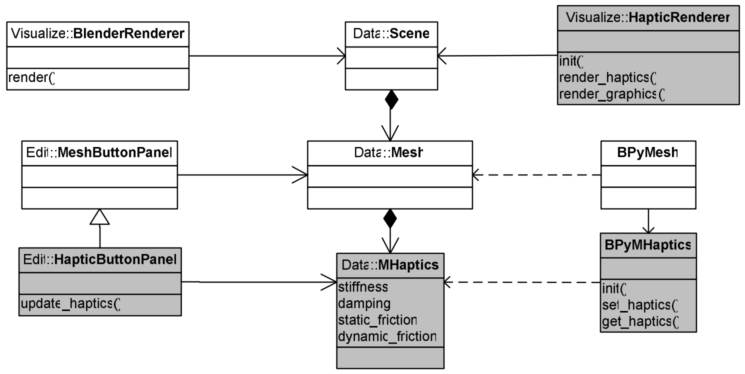


Fig. 4. Class diagram of modifications to the Blender framework (Components added for HAMLAT are in gray)

3.1. Data Structure

In this section, we describe the extensions made to the Blender source code to accommodate haptic modeling and rendering capabilities. Blender applies different data structures to various types of objects in a 3D scene. The Mesh data structure is used to describe a polygonal mesh object. It is of particular interest for haptic rendering since most solid objects in a 3D scene have the same structure. The tactile and kinesthetic cues are typically rendered based on the geometry of the mesh.

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

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

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

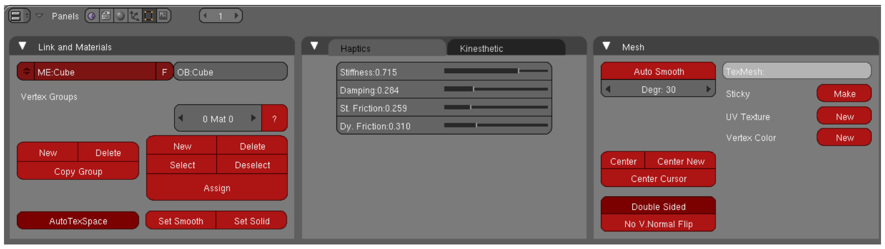
Fig. 5. Augmented Mesh data structure (left), the haptic property data structure (right)

The Mesh data type also has a complimentary data structure, called *EditMesh*, which is used when editing mesh data. It contains copies of the vertex, edge, and face data for a polygonal mesh. When the user switches to editing mode, 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 editing mode is currently used to modify mesh topology and geometry, not the haptic and graphical rendering characteristics. The haptic properties of particular interest are: stiffness, damping, friction, and mass. The hardness-softness of an object is typically rendered using the spring-force equation. The damping of an object defines its resistance to the rate of deformation due to some applied force. The static friction and dynamic friction coefficients are used to model the frictional forces experienced while exploring a surface of a 3D object.

3.2 Editing

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, to perform simple re-meshing operations, and to modify the haptic properties of the selected mesh. The user calibrates the haptic properties (stiffness (N/mm), damping (Kg/s), static and dynamic frictions) and renders the scene until the proper values for these properties are found.

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 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 HAMLAT code is called to update haptic properties for the currently selected mesh.



ME:Cube
OB:Cube
0 Mat 0
Stiffness:0.715
Damping:0.284
St. Friction:0.259
Dy. Friction:0.310
Degr: 30

Fig. 6. Blender's button space, including the haptic property editing panel

3.3 Hapto-Visual Rendering

The 3D scene 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 C code 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.

```
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);
```

Fig. 7. Code for applying haptic properties of a mesh using the OpenHaptics toolkit

3.4 Scripting

The Blender Python (BPy) wrapper exposes many of the C data structures, giving the internal Python scripting engine access to them. For example, the haptic properties of a mesh object may be accessed through the *Mesh* or *NMesh* wrapper classes. The *Mesh* wrapper provides direct access to object data, whereas the *NMesh* class updates changes into the original mesh. Figure 8 shows Python code for reading the haptic properties from a mesh object.

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. The BPy wrappers also expose the Blender windowing system that allows the user to specify meta-data about the application.

Using the Blender's Python scripting engine, we have added import and export plug-ins for HAML files as part of the authoring tool. Modelers may export scenes from the authoring tool, complete with 3D geometry and haptic properties. The HAMLAT interface provides a HAML export function to generate the HAML file.

```
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();
```

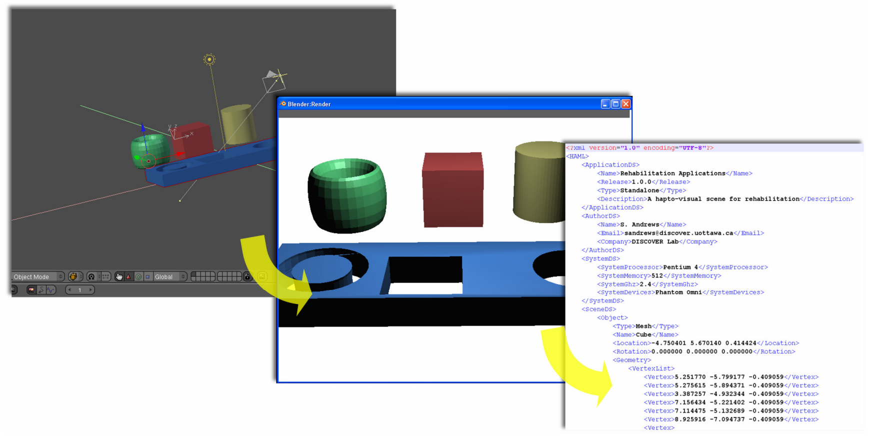
Fig. 8. Export script which uses the BPy wrappers to access haptic properties of mesh objects

4 Application Development

This section provides a brief outline for the development of a simple hapto-visual application using HAMLAT and the HAML framework. Figure 9 shows the three phases of development: design, rendering and testing, and exporting to a HAML file.

For the first step, the author creates the geometry and location of objects in the 3D scene. This includes specifying the orientation and scale of mesh objects, as well as the position of the camera and scene lights. The author edits the visual and haptic properties for each object in the scene by selecting them individually and using the buttons and sliders. The rich set of modeling and editing tools available to the author via the Blender-based interface means that scenes such as the one represented in Figure 9 may be developed quickly and easily.

The modeler can choose to render their "in-progress" scene using the interactive haptic renderer. This allows them to experience how the scene will be displayed to the end user. Evaluating the haptic and visual rendering of a scene is often a necessary step in the modeling pipeline since the author may be unaware of particular aspects of the scene until rendering is performed. Therefore, having an interactive hapto-visual renderer integrated as part of the modeling environment is a powerful feature.



```
<?xml version="1.0" encoding="UTF-8"?>
<HAML>
    <ApplicationDS>
        <Name>Rehabilitation Applications</Name>
        <Release>1.0.0</Release>
        <Type>Standalone</Type>
        <Description>A hapto-visual scene for rehabilitation</Description>
    </ApplicationDS>
    <AuthorDS>
        <Name>S. Andrews</Name>
        <Email>sandrews@discover.uottawa.ca</Email>
        <Company>DISCOVER Lab</Company>
    </AuthorDS>
    <SystemDS>
        <SystemProcessor>Pentium 4</SystemProcessor>
        <SystemMemory>512</SystemMemory>
        <SystemGhz>2.4</SystemGhz>
        <SystemDevices>Phantom Omni</SystemDevices>
    </SystemDS>
    <SceneDS>
        <Object>
            <Type>Mesh</Type>
            <Name>Cube</Name>
            <Location>-4.750401 5.670140 0.414424</Location>
            <Rotation>0.000000 0.000000 0.000000</Rotation>
            <Geometry>
                <VertexList>
                    <Vertex>5.251770 -5.799177 -0.409059</Vertex>
                    <Vertex>5.275615 -5.894371 -0.409059</Vertex>
                    <Vertex>3.387257 -4.932344 -0.409059</Vertex>
                    <Vertex>7.156434 -5.221402 -0.409059</Vertex>
                    <Vertex>7.114475 -5.132689 -0.409059</Vertex>
                    <Vertex>8.925916 -7.094737 -0.409059</Vertex>
                    <Vertex>
```

Fig. 9. Development of a HAML application. Left-to-right: design, render, and export

Finally, once the modeler is satisfied with the environment, the entire application can be exported (including geometry, graphical, and haptic properties) to an XML file for distribution in a HAML repository or playback as a standalone HAML application.

Workflow: 1. User creates geometry for the scene, 2. assigns visual and haptic material properties, 3. exports to HAML file format, 4. HAML player loads the scene from a repository and renders it to the end user.

5 Conclusion and Future work

The current paper presents a HAML-based authoring tool, known as HAMLAT for hapto-visual application development that requires no programming efforts from the modeler. The artist creates or imports graphical models in the HAMLAT editor and assigns haptic properties to them. HAMLAT can render both the graphics and haptics of the created environment. Finally, the modeler can export the environment in a HAML format that can be distributed and rendered using the HAML player.

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 [14]. Furthermore, the current version of HAMLAT does not support dynamic scenes simulation. One of our future works is to enable developers to define the environment dynamics and render them haptically. Finally, rendering of multi-user applications (such as user-object-user simulations) will be considered for incorporation in the upcoming version of HAMLAT.

References

1. Eid, M. Orozco, and A. El Saddik, “A Guided Tour in Haptic Audio Visual Environment and Applications”, International Journal of Advanced Media and Communication, vol.1, n.3, 265 – 297, 2007.
2. N.R. El-Far, X. Shen, and N.D. Georganas, “Applying Unison, a Generic Framework for Hapto-Visual Application Development, to an E-Commerce Application”, Proceedings of HAVE, Ottawa, Canada, 2004.
3. The Haptik Library: http://sirslab.dii.unisi.it/haptiklibrary/. Last viewed on June 1, 2007.
4. The CHAI 3D Open Source Project Website: http://www.chai3d.org. Last viewed on September 3, 2007.
5. Rossi, K. Tuer, and D. Wand, “A New Design Paradigm for the Rapid Development of Haptic and Telehaptic Applications”, IEEE Conference on Control Applications, Toronto, Canada, 2005.
6. A. Pocheville, A. Kheddar, and K. Yokoi, “I-TOUCH: A generic multimodal framework for industry virtual prototyping”, Technical Exhibition Based Conference on Robotics and Automation, TExCRA'04, Treasure hunting in technologies, 2004.
7. HANDSHAKE VR Inc. http://www.haptic.ca/section/view/ index.php.
8. Reachin Technologies: http://www.reachin.se. Last viewed on June 1, 2007.
9. J. Alguire, “Claytools System 1.0”, In GameDeveloper magazine, 2005.
10. FreeFlorm Systems from SensAble: http://www.sensable.com/products-freeform-systems.htm. Last viewed on June 1, 2007.
11. F.R. El-Far, M. Eid, M. Orozco, and El Saddik, “Haptic Application Meta-Language”, DS-RT, Malaga, Spain, 2006.
12. M. Eid, A. Alamri, and A. El Saddik, “MPEG-7 Description of Haptic Applications Using HAML”, In Proceedings of the HAVE 2006, Ottawa, Canada, 2006.
13. Blender official Website: http://www.blender.org. Viewed on June 1, 2007.
14. OpenHaptics Toolkit: http://www.sensable.com/products-openhaptics-toolkit.htm. Viewed on June 1, 2007.