I-TOUCH: a framework for computer haptics

Aurélien Pocheville* and Abderrahmane Kheddar*†
*Laboratoire Systmes Complexes
Université d'Evry-Val d'Essonne, CNRS
40, rue du Pelvoux, 91020 Evry, France
Email: aurelien.pocheville@haptique.com
†AIST-CNRS Joint Robotics Laboratory, JRL
IS, National Institute of AIST
Tsukuba Central 2, Umezono 1-1-1, 305-8568, Japan
Email: kheddar@iup.univ-evry.fr

***Abstract*— This paper addresses the ongoing developments of a haptic (in fact multi-modal) framework called I-TOUCH, which serve two purposes. The first purpose is academic and concerns the conception of a generic framework that is able to allow researchers in haptics to prototype quickly computer haptic algorithms and to do quantitative and qualitative evaluations of their concepts. Nevertheless, the foundations of I-TOUCH are radically different from existing commercially available haptics libraries. Indeed, no haptic graph-scene is defined and haptics is directly derived from the dynamic simulation engine. We are providing a discussion on the pros and cons related to this choice. The second purpose is applicative and concerns a priori virtual prototyping with haptic feedback in industry (automotive or aerospace). Secondary we show that I-TOUCH flexibility allows creating new application with haptic feedback in a simple manner. Moreover, I-TOUCH is haptic device independent and this will be demonstrated by connecting simply various active haptic devices and passive ones. I-TOUCH is also made flexible to imply other modalities rendering devices, such as, obviously vision, but also 3D sound and tactile including thermal feedbacks. This extension is made for the purpose of psychophysics research investigations.**

I. INTRODUCTION

In the human-machine interaction and interface field, the emerging of virtual reality techniques brings into light the human-centered-design concept, which subsequently highlights the major importance of the human sensory capabilities other than the visual one. Providing that the vision modality allows understanding the essential parts of physical and science phenomena, the second important sense in any physical manipulation is indubitably the haptic sense which includes all the complexity of the kinesthetic and tactile modalities.

The haptic interfacing general problem arises at two levels:

1) the identification and the understanding of the human haptic sense,
2) its integration to other sensory modalities for an optimal system use.

The difficulty of haptic perception and interaction comes from the very fact that this modality is *active* since it comes from a direct physical interaction with the environment (through contact and taction). Indeed, the haptic perception and interaction is extremely associated with the human motor functions. This is different from, for instance, vision for which the information sampling does not alter its physical support. The haptic perception and interaction make use of a complex and yet not totally understood flow and effort exchange phenomena of different physical nature between the human and the touched parts of the surrounding environment.

This fact poses a serious dilemma, since the haptic device need to be active, it consequently needs to be motorized in order to be able to constraint human motions. Although some works demonstrated that haptic information could probably be displayed in a passive way [1]. Yet, almost all existing concepts shape haptic displays as compact robotic arms having various kinematics. Their interfacing to a simulation requires to deal with stability of the device and the transparency (i.e. the fidelity) of the feedback which seem to be antagonistic, as in force reflecting teleoperation [2].

Another problem occurs in the way the haptic information is computed in the simulation engine and the way the device is linked to the simulation. As we will see in the next section, researchers tried also several schemes. The very truth is that comparing to 3D sound feedback and computer graphics, there seems to be no real standard to perform the matter. Computer haptics research is still active in this direction but we noticed that there is no tools that allows one to make concrete evaluation of different "bricks" proposed here and there. Indeed, as in computer graphics and 3D sound, the feedback requires different computation of different, somehow independent, modules and their link.

Our aim is design computer haptics and control models that can be implemented in the scope of multi-modal and synergistic human-machine interfaces. This work concerns more specifically the development of a computer haptics framework, called I-TOUCH, where different approaches of computer haptics (and other fields) can be experienced and evaluated. The target applications are rigid bodies prototyping for industry and design companies.

This paper starts with a discussion on actual computer haptics libraries and present our speculative point-of-view on this. It is followed by a description of the I-TOUCH framework and our orientation in computer haptics. Because of haptics is to be used concurrently with other modalities such as 3D sound and vision, its integration coherency is also discussed. This is followed by the evaluation tools. Target applications are described next; we will highlight the flexibility of I-TOUCH in creating haptic applications. Some issues and preliminary experimental results given by I-TOUCH are presented. The paper ends with a conclusion and future work.

II. SPECULATIVE ANALYSIS ON COMPUTER HAPTICS

Existing commercialized computer haptics libraries, such as GHOST[1] of Sensable technology or formal MAGMA, now ReachIn API from ReachIn[2], are scene-graph oriented. Indeed, in the case of GHOST, the virtual objects (mesh set) that are involved in force feedback computation need to be specified beforehand. This specification engenders the virtual environment being composed of "haptic polygon" (or triangle) meshes set and a non haptic one. When the virtual probe, point or proxy, manipulated by the human operator via the haptic device, comes into contact with the haptic set, built-in collision detection and response force computation returns, based on a penalty method, the reaction force is displayed to the operator though constrained in appropriate directions the force feedback device. In this method the specified haptic set are somehow blended with the visual triangles through the OpenGL library. The ReachIn computer haptics API differs from GHOST mainly in the fact that the visual graph scene and the haptic one are more independent. The other advantage of ReachIn API is in its independance from the haptic interface; which is not the case for GHOST since it is dedicated to the only Sensable products.

Recent projects such as OpenHL (Open Haptic Library), by analogy to the well known standard graphic library OpenGL, or Chai3D[3] took similar directions with an open source software. The first open source project, OpenHL, seem to be in a *statu quo* stage, the second project, Chai3D, is more ambitious since it is developed in C++ and is specially designed for education and research purposes offering a light platform open for extensions.

Yet the design philosophy of these haptic libraries poses some limitations. First of all, they are all considering point-based-interaction. Indeed it seems to be difficult to use these libraries in the case of a more generic haptic framework. Namely, actual applications requires haptic rendering of complex virtual objects, as it is the case in automotive or aerospace industry prototyping. Other applications require manipulation of deformable objects like in some product design in manufacturing or in interactive surgery simulators. Of course, we are aware that the interaction can be described by a set of "equivalent" points; one need however to program this and make changes that will drastically decrease performance. Just try them in complex scenarios, in fact simple, where an object is manipulated instead of a point.

Designing computer haptics on the basis of computer graphics pipe analogy [3] rises fundamental physics controversies. First of all, object's mass and inertia can not be directly rendered and "distributed" on the so called *haptic meshes*: friction (static and dynamics ones), impact impulse forces... are simulated on the basis of simplistic models. These very haptic parameters are tuned and conceived as *special haptic effects* that are blended with the contact force computations. Other design such as voxel-based or specific applications based softwares such as VPS[4] show evident limitation when being extended to a more generic framework. The reaction force computation in VPS do not obey any common elementary physics.

If this way was indubitably a necessary path to understand and to promote the haptic technology while spreading it to many applications, it shows now clear limitations to be release as a standard, similarly to computer graphics and 3D sound rendering cases. In our opinion, the difficulties come mainly from two points:

- the difficulty to convolve toward an uniform haptic interface device;
- the difficulty in writing hardware independent computer haptics.

Our approach to computer haptics differs from what is actually considered by some developers such as Sensable, Reachin, VPS, Chai3D and others. Indeed, we think that using a specific haptic scene-graph together or separate from the traditional polygon based graphic libraries is adequate to point-based or polygon-based interaction but is not generic enough. Actually, free motion, inertia, friction, force fields... are implemented as *specific/special haptic effects*. For this reason and for rigor concern, we believe that computer haptics may gain in efficiency if it could be considered as part of the solver i.e. built-in part of the virtual environment dynamic simulator engine. However, if considered so, additional constraints must be taken into account: mainly the real-time and the operator interactivity issues. We noticed that the algorithmic complexity is mainly concerned with two aspects:

- the collision detection computation, and
- the dynamic constraint computation.

Collision detection is a fundamental problem in numerous domains [4] [5]. It is the bottleneck of every physically based simulation and known to be very time consuming. Collision detection methods can be split into two categories: discrete and continuous. Most methods are discrete ones: they sample the objects motions and detect objects inter-penetrations. As a result, these methods may miss collisions (tunneling effect). Moreover, discrete collision detection requires backtracking methods to compute the first contact time, which is necessary in constraint-based analytical dynamics simulations [6]. Depending on the object complexity, however, the computational cost of backtracking may be unpredictably large, mainly because estimating the penetration depth is a difficult problem, for

[1]`http://www.sensable.com/`.
[2]`http://www.reachin.se/`.
[3]`http://www.chai3d.org/`.

[4]`http://www.boeing.com/phantom/vps/`.

example when many triangles have penetrated or if the object is concave or non-convex.

In haptics, the penetration problem is a major cause of instability. As opposed to these methods, continuous methods compute the first time of contact during the collision detection [7]. This computation is inherently part of the algorithm. While more suitable to robust interactive dynamics simulations (to guarantee collision-free motions), continuous methods are usually slower than discrete methods, and are often abandoned for discrete ones. As for collision response computation, which in fact induces forces (some of which will be rendered) there are several methods also reviewed and analyzed in [7].

Since we design computer haptic on the basis of physically based simulation engine, several combination of existing or novel bricks composing the process of the reaction force computation are possible. In this case however, all the scene is haptic and haptic parameters such as mass, inertia, friction coefficient... are already present for the physics engine. To some extent, all the virtual environment is haptic, since motion is related to forces. In order to feedback haptic information to the user, all what one need to specify is which virtual object is being grasped or touched or manipulated: it will be called the proxy. Part of the computed forces, namely those applied on the proxy are fed back to the user operator through the haptic connected interface.

The developed framework will investigate evaluations, in the frame of haptic feedback, of existing or newly developed collision detection algorithms when coupled with existing or newly developed physically based animation methods. As suspected, preliminary coupling reveals some difficulties in achieving the matter.

III. The I-Touch framework

A. Key concepts

We believe that computer haptics will gain in many aspects in being developed dissociated from the interface. We also strongly endeavors toward enrolling computer haptic as part of the simulation run-time engine. Of course the problem, in this case, is that we need a physically-based simulation engine and some applications are not necessarily embedded with it. In these last cases, we recall that whatever the application is, as far as haptic feedback is envisaged, collision detection and force computation are required. Then force feedback quality relay on the sophistication degree of the simulation engine. I-Touch is based on this philosophy. It aims at providing a simple, flexible, modular and easy to benchmark framework, which would provide haptic experience while handling multi-modal interaction.

Our conception brings however an interesting generic issue: in I-Touch, haptic feedback is “disconnected on demand” from the simulation: this is a key concept. Indeed, implementation of such an issue is very challenging and this challenge allows for a generic and a flexible use. Haptic devices are thus completely separated from the simulation engine. However, haptic *information* is the lot of the virtual environment.

Thanks to this framework, we are now able to test different behavior models, along with new collision detection algorithms and haptic paradigms.

B. Design of the I-Touch framework

The I-Touch framework is designed to be modular from its core to its interfaces. Although it is still in the development process, it already allows plug-ins (static linking in program) of different behaviors models and different collision detection algorithms. The framework architecture is given in Figure 1.

The framework is divided in three main modules; each of them is further subdivided in as many submodules as needed:

- *The core system* is responsible for handling the operating system, the configuration, and the basic functionalities of a physically-based simulation. It provides a basic scene graph for managing the various objects that composes the virtual scene. This core system can accept many simulation algorithms along with different input methods. Classical mathematical methods [8] [9] and structures are also provided, for the easy prototyping/evaluation of new/existing algorithms.
- *The input and output system*. While the input system needs to be flexible and needs to manage many different inputs, the output system should ensure high fidelity rendering along with adequate refresh rates according to the addressed modality/output.
- *The simulation system* is composed of the simulation manager, and a set of simulation virtual objects. The simulation system use the core system for standard interaction with the computer and the user, and input/output system for multi-modal interaction. Collision detection algorithms are part of the simulation system.

The I-Touch framework is completely object-oriented. This allows easy part replacement and improvements. It is implemented in a pure standard C++, and apart from the driver libraries, does not use platform dependent code. It can be easily ported to Linux or MacOsX, however it is designed to be best suited to MS Windows OS.

C. The core system

To facilitate benchmarking and evaluating of new concepts/models, one needs an easy access to the configuration, to the system and to the others components that are not directly involved in the simulation. The core system addresses these requirements. First of all, it provides an easy configuration access for the simulation algorithms. Since parametrization is very important for fast testing, every aspect of the framework is parametrized. Making this, is just as easy as declaring a variable in C++ and making an equivalent variable in a given configuration file. Configuration sets can be put together, thus creating test cases. The configuration is stored in XML-type files which permits easy extraction and manipulation of this data

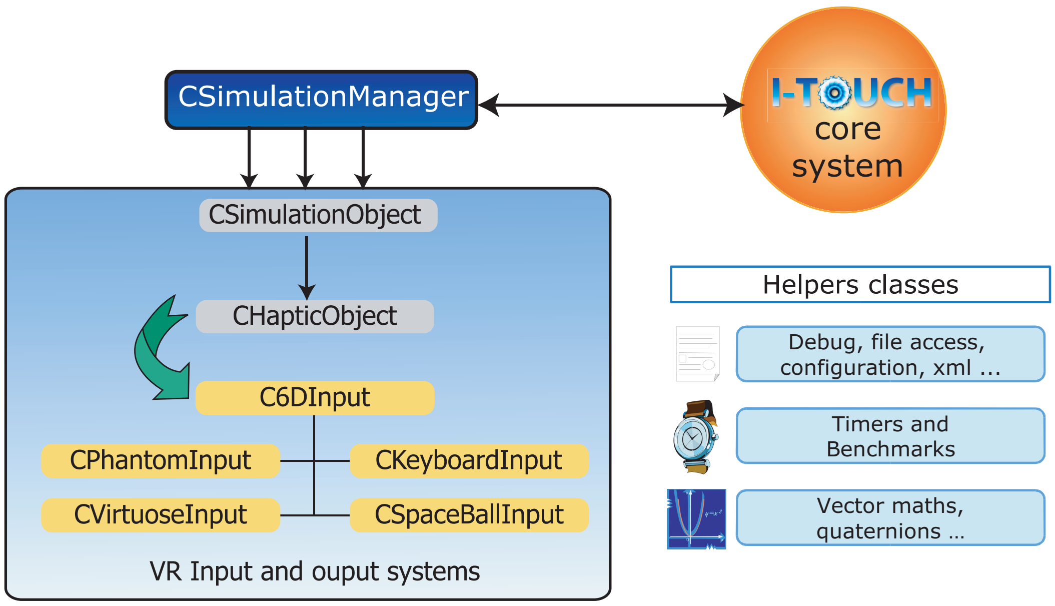
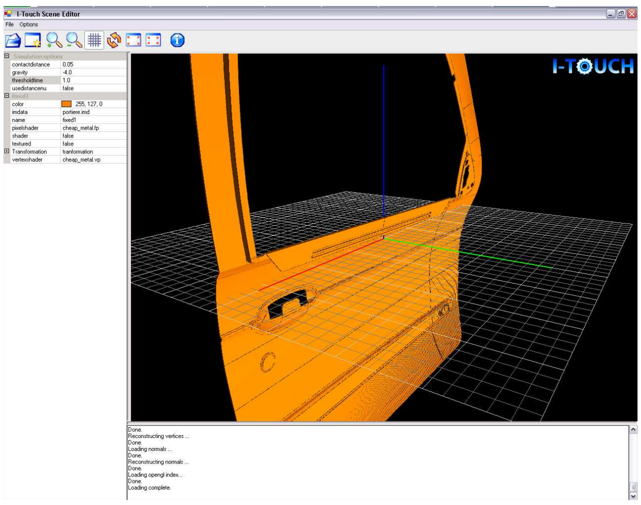


Fig. 1. I-TOUCH framework architecture.

by third party programs. Also, the framework provides an easy way to map XML. Moreover, an off-line simulation viewer/creator is in the development process; it allows fast previewing of scene, and alteration of objects parameters. This viewer is shown in Figure 2.



Simulation options	
contactdistance	0.05
gravity	-4.0
thresholdtime	1.0
usedistancenu	false
fixed1	
color	🟧 255, 127, 0
imdata	portiere.imd
name	fixed1
pixelshader	cheap_metal.fp
shader	false
textured	false
Transformation	tranformation
vertexshader	cheap_metal.vp

```
Done.
Reconstructing vertices ...
Done.
Loading normals ...
Done.
Reconstructing normals ...
Done.
Loading opengl index...
Done.
Loading complete.
```

Fig. 2. The I-TOUCH editor.

In addition to configuration tools, a file format for holding together "geometrical" object properties has been devised. This format is open and flexible, moreover, additional data can be included and be ignored if not necessary to the simulation, even if it is an unknow data.

An importer and exporter have been written for 3DS-Max[5], along with C++ and C# libraries for loading efficiently theses files. Moreover, data channels have name identifiers, making easy to bind them to configuration and/or simulation data.

[5]`http://www4.discreet.com/3dsmax/`.

System abstraction is also provided, to ensure portability. The standard graphic library OpenGL is used for visual rendering, together with its bundle of the latest hardware embedded technologies, such as texture mapping, vertex and pixel shaders to create stunning effects and visually-reality-like environments. 3D viewing with stereo glasses is fully supported. 3D positional audio is also available. A GUI for the visualization of simulation parameters and others properties is in the development stage.

At last, many helpers classes are provided for easy prototyping and debugging of algorithms. Math, debug and input abstraction are available in an convenient way (input used for scene management is not the same that the one that is being used for 6dof inputs). Debug output can be redirected to console, for on-line analysis, or can be saved in text or HTML files for easy off-line management.

D. *The input and output system*

The input and output system takes an important place in this framework for obvious reasons. The framework is very human-centered designed and should be able to handle many different input devices, from simple keyboards to passive haptic devices such as 6dof SpaceMouse to active haptic displays such as the Phantom or the Omni[6], the Virtuose[7], the Delta or the Omega [8], etc. Moreover, some of these inputs also outputs force feedback. It appears that we need to "forward" inputs and force feedback back and forth between the simulation system and the real world.

The maximum degrees of freedom of an object in the simulation is 6 (an object is put in open space), therefore there is a maximum of 6dof input in the acceleration space, speed space, or position space. This gives a maximum of 18

[6]`http://www.sensable.com/`
[7]`http://www.haption.com/`
[8]`http://www.forcedimension.com/`

input information. For the feedback the sames rules apply, giving 18 feedback channels.

The primary class used for input and haptic output provides access to each of these 36 channels. But this access is not effective unless a derivate class provides actual processing of the requests from the simulation. For example, force feedback sent to a keyboard is not processed. However, actual force put on a space mouse can be retrieved and used in the simulation. To let the simulation engine know dynamically which capabilities an input or an output actually has, a function is available in the base class, and is overridden as needed. At last, the class provides access to unlimited number of device buttons for use in the simulation.

Since the simulation is completely parametrized, it is not possible to foresee which object will be attached to an haptic controller, and what the device capabilities of this haptic controller will be. The mechanism presented here permits "hot" plugging[9] of different haptic devices, and an instant usability in the simulation. A even more interesting approach would be to encapsulate these class in dynamic libraries, that would be loaded at the beginning of the simulation or a given staring points of the simulation. That would provide a way to support additional devices that were not available before, without rewriting nor re-compiling any part of the framework engine.

The visual output system is heavily based on OpenGL and its latests extensions. Basically, objects are linked to material properties (such as colors and textures maps) along with optional vertex and pixel shaders. These shaders can be used to greatly enhance the realism of the visual output.

E. The simulation system

This part of the framework is the most challenging one. It is responsible for the behavior model of the scene. The simulation system is divided into two parts: the simulation manager that deals with calculus and algorithms, and the simulation objects that are placeholders. The simulation manager uses objects properties to drive its computations.

1) The simulation manager: The simulation manager is the central piece of the behavior model. It implements physics simulation laws. It uses the collision detection algorithms and the input systems as an entry. The simulation manager computes the next state of the system. Then, multimodal output systems are used to reflect/project this new state to the operator.

At the time of the writing, two simulation managers have been successfully implemented: one that use constraints for physic calculations, and one that use bounce physics[10]. These simulation managers require proximity queries. While SWIFT++ [10] does provide proximity query, it is only for one point, making it unstable used. Therefore, we are developing a new collision detection algorithm with these new constraints in mind. The simulation operate on flexible frames per second, in order to use maximum capabilities of the hardware. This also means that simulation managers should (and in some extend are) able to cope with low frame rates. The simulation loop somewhat differs from a classical simulation loop, that is:

```
1- Initialization of different objects
foreach time step $\delta_t(t)$ do
    2- Read Haptic Device() (through input classes)
    3- Calculation Of Desired Speed(Haptic Objects)
    4- Non contact forces are applied to objects, but
    their position is not yet affected
    5- Contact points = Proximity Detection()
    5- Compute Contact Forces()
    7- Update Desired Speeds()
    8- Update Position() // integration step
    9- Multimodal Rendering()
end
```

Steps 2, 3, and 9 are part of the haptic proxy concept.

We are using energy conservative integration step in the form of:

$$\vec{\mathbf{p}}\ (t+\Delta t) = \vec{\mathbf{p}}\ (t) + \vec{\mathbf{s}}\ (t+\Delta t)\Delta t - \frac{1}{2}\ \vec{\mathbf{a}}\ (t+\Delta t)\Delta t^2$$

where $\vec{\mathbf{p}}$ is the position, $\vec{\mathbf{s}}$ is the speed and $\vec{\mathbf{a}}$ is the acceleration. This integration step was the most stable yet speed effective for our experiments. Of course, other integration steps can be used and evaluated in a simple and transparent manner.

2) The simulation objects: The simulation objects are conceived as "inert objects", that is to say, they do not make decision by themselves. Instead, the behavior part of the simulation is left to the simulation manager. First, this allows to have independence from data representation. This greatly clarifies the way algorithms work. Then, from the theoretical point-of-view, it ensures that the representation (by representation, we mean visual, haptic and/or any other) is not dependent of the behavior model.

We will end out this section by the following important issue raised by I-TOUCH:

Remark *The way a behavior model handles forces, contact and collision should not affect haptic rendering. In this framework, haptic objects are just standard object attached to an haptic controller. The haptic proxy, then, has to take care of exact representation of the forces. In a similar way, object have a "visual rendering" device that renders objects. The point here is independence, from data representation to behavior model to haptic, visual and 3Dsound rendering.*

IV. MULTIMODAL INTEGRATION

One of the most challenging issues of I-TOUCH is multimodal integration. Visual, auditive and haptic senses have different refresh rates: from as low as 30Hz for visual interaction, up to as high as 10kHz for the 3D sound one. Integrating each of these modalities is not a trivial

[9]In theory, this "hot" plugging would work even if the device was attached while the simulation is running. However, most devices requires initialization that is done at starting.

[10]Bounce physic has been less investigated than constrained physics.

task. Others tentatives tried through parallelization of the computation on different computers. Here, we decided to push the limits on focusing in using one computer, but this unveils some problems as exposed later.

A. Simulation engine flexibility

The fact that the simulation engine is completely flexible and modular allows the integration of different behaviors models with the same multimodal rendering. However, the simulation engine has to provide some information to the output routines. For example, for the real-time 3D sound rendering, contact information (and changes in contact through time) are required. The immediate benefit of this is that we can benchmark how well does a simulation engine behaves with multimodal rendering. For example, bounce models have difficulties in rendering contact information with sound, while they provides excellent rendering of bounce sounds.

B. Sound integration

3D positional audio, while not being as primordial as haptic rendering in most prototyping applications, greatly enhance the immersion of the operator in the simulation. We have two methods for rendering 3D sound: real-time rendering, and semi-real-time rendering.

The real-time rendering uses information directly provided by the simulation, such as changes in the friction map to produce sound. It also uses object properties such as resonance frequencies to computes contact sounds. While this is the correct method for producing friction and bounce sound, it suffers from several drawbacks. First of all, it is very computational-time consuming, and, in a system composed by only one processor, it can become the bottleneck of the simulation (and take the place of the collision detection!). Maybe relocating the sound computations could solve this problem. The other fact is that the sounds generated are, for now, less “realistic” than the ones produced by the second approach.

The semi-real-time sound rendering approach uses offline recorded sounds of different materials in contact. These different sound are stored in a database according to some material properties. They are used by the simulation as they are and the only amplitude and/or frequency modulation (pitch, volume...) are processed.

C. Visual integration

Relatively to the sound and the haptic rendering, the visual one is the easiest. We can use the same geometry as the one used for physics calculations, or a higher level, smoother one for better rendering. Objects are linked to rendering information, such as geometry, material and alpha information, and pixel and vertex shaders. This allows almost any rendering of the objects, from standard Gouraud-shaded plastic look, to advanced Phong-shaded semi-reflecting materials with bump mapping. Dynamic lighting is also supported. The visual rendering is completely configuration controlled, so there is a great flexibility in the rendering process.

D. Haptic integration

Classic haptic rendering usually involve a specific behavior model. Often, the haptic device data is trusted, in the way that the position of the haptic controller in the real world is believed to be the position of the haptic controller in the virtual world (scale effect taken into account). Haptic feedback used to be simple spring mass system linking virtual and real positions[11]. Some works, such as Barraf’s [11], use also spring mass system to obtain the position.

Our approach differs from the previous ones. Indeed we are conceptually considering that *the haptic devices (interfaces) interact with the simulation and not the reverse i.e. the haptic device does not drive the simulation*. Obviously, haptic devices can induce a change in the course of the simulation but they cannot compromise its integrity. To be more clear, the simulation does not take as granted what is needed from the input device and, in extreme cases, these particular inputs are ignored. In fact, these enhances considerably the stability of the interaction. For example, when the operator actions are toward violating a given non-penetration constraints, they are not considered integrally (as is the case for classical computer haptics methods).

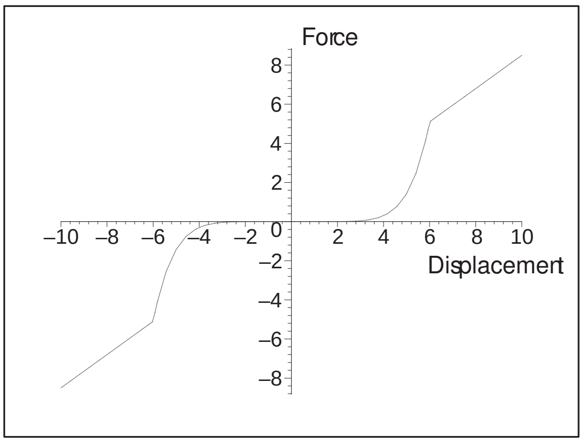


Fig. 3. The new ramp used in place of plain spring

From the simulation part, the operator is *requesting* that the user controlled virtual object moves to a place in the simulation. The haptic proxy processes then computes what would be the object’s speed if it would make the complete move. In the case of free movement, no change is made to this speed, so when the integration step is done, there is resynchronization between virtual and real positions. However, if the object is in contact, the simulation will issue a force to balance the speed. At the integration step, the speed will be reduced, and the two positions will become shifted. In fact, most simulations try to match real and virtual positions of the haptic controller. With this method, this is not required. We can have unsynchronized positions without losing stability. Of course, since the human perception of motion is relative and not absolute, the operator will be completely blurred and will not feel

[11]In addition, most simulation add a damping term, for stability purpose. In our case, that was not even necessary.

the shift between the two positions if any. The formula used to compute speed is the inverse of an the integration step, that is:

$$\vec{\mathbf{v}} = \frac{\vec{\mathbf{p}}_{new} - \vec{\mathbf{p}}_{old}}{\Delta t}$$

For the feedback part, we use a special devised spring system, that reduce vibration noise induced by frequency differences between the haptic and the physic simulation loops. The haptic loop runs at 1kHz, whereas the physic loop runs at 100Hz (even less sometimes, depending on the complexity of the virtual scene and the simulation scenario). The haptic proxy, through feedback, has in charge to reduce the effect of position shifts between two simulation steps. This is done by the spring curve given in the figure 3.

This function profile allows adapting to the frame-per-second rate in free motion, preserving the haptic interaction when contact occurs. Experiments with users showed that this type of haptic feedback did not induce any change in the human-behavior or the performance when using the haptic device.

The fact that haptic integration is not considered as a special rendering allows new synergies between rendering to be investigated. One example of this is the recently implemented *haptic bump*. As for visual bump mapping, we can simulate rough haptic surfaces through haptic bump[12].

We tried two different approaches: height based forces, and normal based forces. The basic principle is the same: the force computed by the simulation engine is slightly modulated by a term, which depends either on the height or the normal. In our actual implementation, haptic bump does only work with one contact point, but we are working on extension to multiple points. As far as " bump sensation" is concerned, the normal based force give superior results. With even a very slight modulation of the kinesthetic force, the effect is surprisingly very present and gives the feeling of a rough surface. Moreover, the bump map used for haptic bump is exactly the same as the one used in the visual bump, thus the two modalities match perfectly and the rendering is coherent. The operator experienced an enhanced quality of multimodal interaction. In the near future, we will try to make haptic bump mapping computation to take part in the hardware.

V. Evaluation tools

The testing of research projects is made easy with I-Touch, however such a testing requires to analyze data from the simulation. In I-Touch, every simulation variable can be "tagged" from recording, this allows after-run simulation analysis through a special tool (show in Figure 4).

For example, FPS data, or time taken to compute a frame (much more speaking than FPS in regard to performance)

[12]A specific device is being developed to render surface tactile and roughness informations [12]

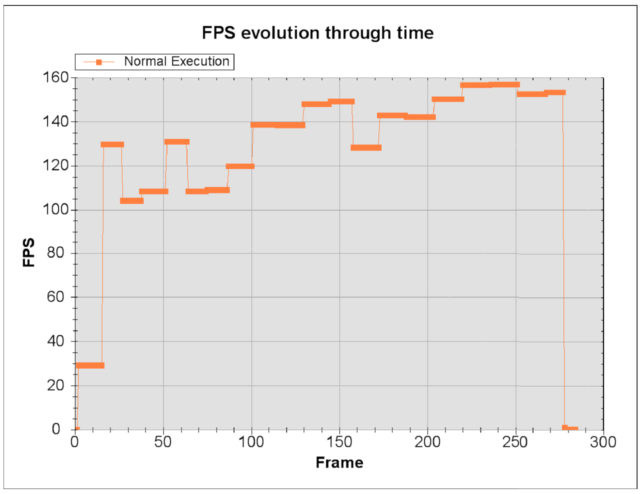


Fig. 4. Example of frame rate monitoring data.

evolution can be viewed easily. Also, the debugging facilities and text functions in I-Touch make it easy to dump data. However, we want to go further, and new real-time tools are in development. Such tools will render in real time evolution of variables, in numerous manners (time graphs, bars, standard text, etc.). Specific analysis tools to simulation should also be included, such as automatic reporting of number of contacts, physics calculation time, time spent in rendering or in other tasks, etc. This will create a complete and easy-to-use evaluation tool, in order to accelerate further the development of test cases.

VI. Applications examples

To prove the extensibility and flexibility of I-Touch, we made practical test cases to show actual implementations of the proposed concepts and algorithms.

A. Virtual scribble

The first application is the so called *virtual scribble*. The purpose of this demonstration is to demonstrate how easy it is to create and derive entire applications from I-Touch. In virtual scribble, the haptic device (in our case, a Phantom device) is hold like a pen. In the virtual world, a sheet of paper standing on a desk is shown to the user. The user can then use the virtual pen to write virtually (of course, the Phantom is in fact in the open space). The following steps were required to make this sample application:

1) Imports 3D models of a pen and a desk thanks to 3Dsmax and then use the exporter to create `imdata` files.
2) Create two objects in the configuration file, either with a text editor or with the offline scene explorer. One of the object is the desk, and is marked as not moving (infinite mass). Use the offline scene explorer to check placement and object properties. Set textures to the objects.
3) In the configuration file, bind the pen object to a haptic controller (Phantom support is built-in).
4) In I-Touch, use a simple height-test to handle collision (contact detection), or use a more complex

algorithm (height test was used in our case). Select the default behavior model (constraint based) to handle contact resolution.

5) Add the code to handle collision between the pen and the desk. In our code, contacts points are saved in a list and then rendered to screen. One possible extension would to add scribble sound to the simulation.
6) Run the simulation, and let children enjoy writing practice!

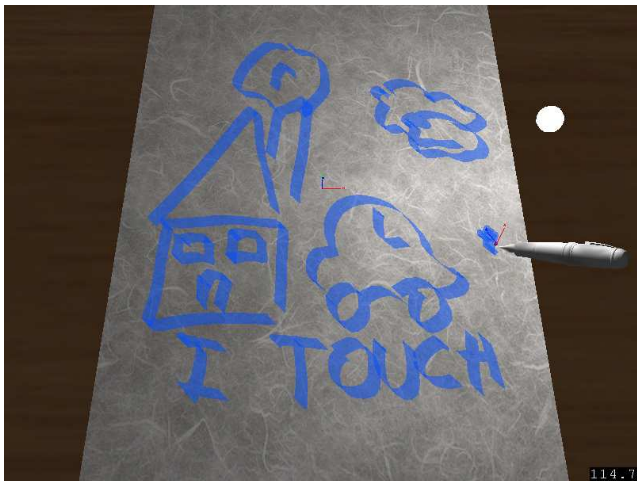


Fig. 5. Virtual Scribble sample application.

A snapshot screen of virtual scribble application is illustrated in the figure 5. As we can see, the implementation of this simulation does not require a great amount of I-TOUCH internals. The fact that code can be added in response to events will be in the future separated from the main program and will be available as dynamic libraries. This will allow customization of I-TOUCH without changing code.

B. *Virtual prototyping*

One of the main aims of I-TOUCH is virtual prototyping (VP). VP is to be seen as a complementary tool to CADM software techniques. It is the front end of a product life management process taking on board constraints related to manufacturing, utilization, and maintenance. To fulfill human-centered designs, the VP architecture should allow "digital mock-up" to be interactively explored, manipulated, and tested in various usage scenarios. VP implementation is not an easy task. It involves the successful integration of multimodal rendering with physically realistic behavior model, at high refresh rates. In industry, model precision is of prime importance.

We are currently developing a virtual prototyping case, which uses our built-in collision detection, behavior model and haptic proxies. Steps required to create such a program are the following:

- Identify the virtual prototyping tasks and involved objects.
- Import/Export 3D models of these objects from industry internal format (CADM software).
- Configure objects.
- Bind a haptic interface (PHANToM, Virtuose, etc.) to the manipulated virtual object.
- Use default or specify algorithms for physics and collision detection (the choice option is still under development).
- Perform VP tasks within I-TOUCH.
- Measure what ever must benchmark (not yet envisaged).

This case does not differs really from the previous one, however it has three big differences. The first one is that we are treating a more complex scenario, which requires more processing power. The robustness of the algorithms (in regard to the number of polygons and contact points) is of vital importance. Secondly, while in Virtual Scribble physics/collision detection are not very important, here we must have realistic devices. And, at last but not least, we have many contacts point instead of only one. Currently, very few haptic software handle multiple contact points, and they are often sacrificing in others parts.

An illustration of this case is given in the figure 6. The VP scenario consists in mounting/dismounting of a window-winder motor in/out of a car door (the 3D models are kindly provided from RENAULT car industry and the CEA (French nuclear authority)). The operator can test if the window-winder does really fit, and if it is possible to put it in place, accounting for the shape of the door car. What is gained here is the intuitiveness of the operation. The CADM engineer disposes a powerful tool that allows him quick changes of CAD models. Operation timing can also be monitored as well as forecasting maintenance operation procedure and eventual requested tools.

C. *In development progress*

We are currently investigating, through the use of I-TOUCH, new haptic paradigms and interfaces. One of these issue concerns thermal feedback interfacing. Thermal feedback would provide thermal exchange properties produced by different materials when using bare hand interaction: for example metal feels "colder" than the wood, which in turn if often felt warmer than plastic. Through the interfacing of thermal devices in I-TOUCH, we are trying to test different thermal rendering algorithms. Here again, I-TOUCH our work is mostly focus on mathematic models and coupling than in software adaptation and change. This demonstrates the modularity and the flexibility of I-TOUCH.

Following the haptic bump paradigm, we are also trying to interface I-TOUCH with new haptic devices, which would feed bumped surface in the real world [12]. Of course, the coupling of haptic bump and visual bump will remain unchanged. The haptic device abstraction will make it easy to integrate the new device, which will produce "actual" effect in place of the "simulated" haptic bump through kinesthetic device.

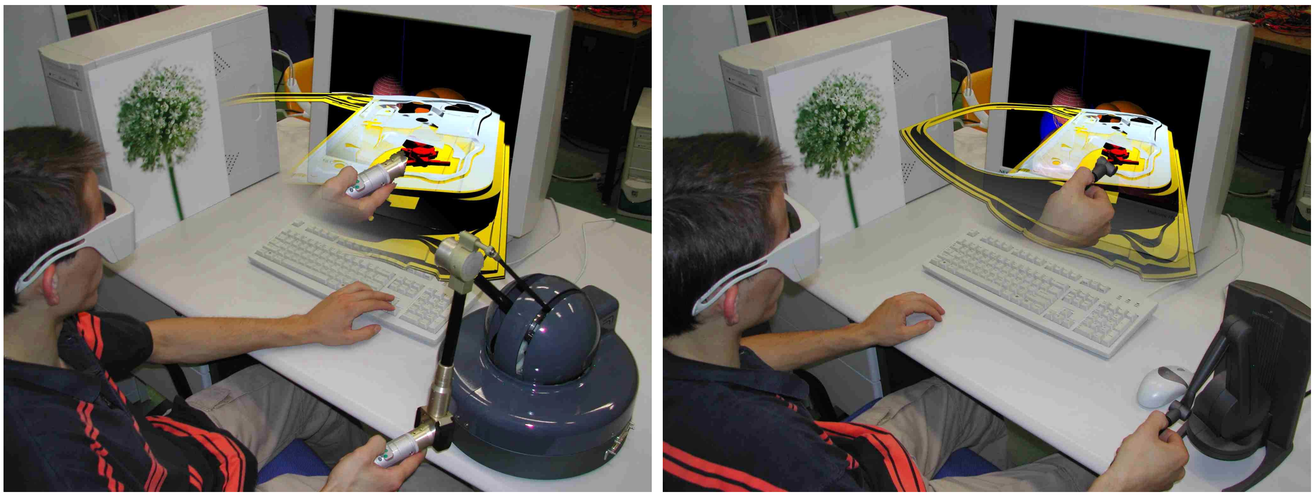


Fig. 6. Virtual prototyping application, the car door is a courtesy of RENAULT©. Left image show the scenario of a unmounting feasibility check with haptic feedback using the Virtuose haptic device. The same image is illustrated right with the PHANToM as a haptic device.

VII. Issues raised by I-Touch

A. On data models

In most simulations, objects are modeled by their surfaces that are meshed into a set of triangles. Most collision detection algorithms make often use of convexity properties, and assumes that virtual objects can be seen as a union of convex sub-objects. This assumption, from our experience, leads to more problems than it actually solves. In the real world, very few objects are convex. This leads, almost every time, to a decomposition into convex objects [13]. This decomposition has inherent problems at jointures, because special treatment has to be made to handle "false surfaces" that did not exist in the initial object. Furthermore, one of the main arguments in favor of convex objects is that they permit only one contact point between two convex objects. The trivial example of a cube resting on plane shows that a simple plane/plane contact exhibits more than one contact point. In the case of constraint physics, more than one point for plane/plane contact is mandatory. We believe that making a distinction between convex and non-convex models is not viable in the near future. Instead, we focus on methods that work on arbitrary models [14].

B. On behavior models

One of the issue raised by haptic rendering is real-time constraints. Because most of the haptic loops runs at 1kHz, there is a need for fast simulation. Even with a haptic proxy, we noticed that the simulation frame-per-second dropped below 50Hz for common scenarios. It was extremely difficult to efficiently use the haptic rendering. Many simulation adapt to this haptic real-time requirements by decreasing the precision of the physics, and often by acting directly in the behavior model. One of the most well known behavior model is the penalty method. This model is flawed at the basis, because not only inter-penetration (of course inter-penetration does not happen in the real world, so it is discarded in virtual prototyping and in most realistic physical simulation) is permitted, but it is *required* for response. Many examples of invalid force computations exists in the penalty realm. As a result, the blue object does not move at all. One can argue this situation happen because of heavy inter-penetration, but since this one is required, there is a chance that situations like the one showed here will eventually happen, even if the time step is small.

For rigor concern we focused on constrained based methods. Constrained methods do not require inter-penetration to compute reaction forces, they prevent it instead. The fact that objects can not move into each other is translated explicitly into unilateral constraints conditions on relative speed, and absolute positions and forces. One of these model is the one introduced by Sauer and Schömer, which has been adapted in one of our scenario. This model translates the non penetration problem into a LCP[13] formulation, that can be solved by many different algorithms. In this model, a foresee step is made, which parametrizes the next positions and speeds of objects by the forces applied to contact points. Then it uses the fact that for a contact, one of the contact force or the relative contact speed is null, while the other is not. The final complementary conditions of this model is:

$$\mathbf{N}^T\mathbf{J}^T\mathbf{M}^{-1}\mathbf{J}\mathbf{N}\Delta t\mathbf{f}+\mathbf{N}^T\mathbf{J}^T(\mathbf{u}^t+\Delta t\mathbf{M}^{-1}\mathbf{f}_{ext}) \geq 0 \perp \mathbf{f} \geq 0$$

where $\mathbf{N}, \mathbf{J}$ are used to transpose mass and inertia matrix $\mathbf{M}$ to contact points, $\mathbf{f}$ and $\mathbf{f}_{ext}$ are respectively contact forces and external forces, and $\mathbf{u}$ is the speed vector at instant t. We solved this LCP by using the Lemke algorithm provided in [15].

This model gives good results, if the collision detection step (that would be more a "proximity detection step") provides enough information. However, most current algorithms do not provide such information. The fact is, it is very uncommon (and much more difficult) to ask

[13]**L**inear **C**omplementary **P**roblem

for proximity, opposed to intersection. This is why most software packages reports only collision detection (and it is often in the form of a yes/no answer, the interpenetration depth is not often available). In addition, the packages that report proximity information only report one point, which is is course insufficient in a plane/plane contact. What we need here is a algorithms that reports all “contacts” points, that is a sort of contact topology. This strengthen a lot subsequent algorithms. Due to numerical errors, it is also necessary that such a package account for jitter and use numerical thresholds to determine the contact topology. Such a software package is currently in development at the LSC.

VIII. Conclusion and future work

With this work, we developed a new approach to handle computer haptic simulations. We conceive that haptic virtual objects are to be considered in the simulation like any other objects, with no more and no less rights. This enrolls the haptic rendering computation as part as the physically-based simulation engine which compute contact forces based on a close external force/acceleration/motion loop. The fidelity of the haptic rendering depends on the sophistication of the simulation engine which are built on the basis of different bricks such us the used physical equation formulation, the numerical integration method, the collision detection algorithm, etc. To prototype and evaluate these bricks, we build I-Touch a multimodal algorithm benchmarking framework and we target applications that are known to be complex, like virtual prototyping in industry.

Further work is oriented toward refining I-Touch through its multimodal component to serve also as a psychophysics evaluation tool. Progressively our aim is to evolve it to a complete piece of software that can serve haptic research.

Acknowledgment

This work is partially sponsored by the TOUCH-HapSys (see, `www.touch-hapsys.org`) EU CEC project. Contract No. IST-2001-38040, action line IST-2002-6.1.1 (FET-Presence) under the 5th research program.

References

[1] A. Lécuyer, S. Coquillart, A. Kheddar, P. Richard, and P. Coiffet, “Pseudo-haptic feedback: can isometric input devices simulate force feedback?” in *IEEE International Conference on Virtual Reality*, New Brunswick, 2000, pp. 83–90.

[2] D. A. Lawrence, “Stability and transparency in bilateral teleoperation,” *IEEE Transactions on Robotics and Automation*, vol. 9, no. 5, pp. 624–637, 1993.

[3] G. V. Popescu, “The Rutgers haptic library,” in *IEEE International Conference on Virtual Reality, Haptics Workshop*, New Brunswick, 2000.

[4] M. C. Lin and S. Gottschalk, “Collision detection between geometric models: a survey,” in *IMA Conference on Mathematics of Surfaces*, vol. 1, San Diego (CA), May 1998, pp. 602–608.

[5] P. Meseure, A. Kheddar, and F. Faure, “Détection des collisions et calcul de la réponse,” Action Spécifique DdC du CNRS, Tech. Rep., 2003.

[6] D. Baraff, “Fast contact force computation for nonpenetrating rigid bodies, siggraph,” *ACM SIGGRAPH*, 1994.

[7] S. Redon, “Algorithmes de simulation dynamique interactive d’objets rigides,” Ph.D. dissertation, Université d’Evry, 2002.

[8] L. Dorst and S. Mann, “Geometric algebra: A computational framework for geometrical aplications,” *IEEE Computer Graphics and Applications*, 2002.

[9] J. C. Hart, G. K. Francis, and L. H. Kauffman, “Visualizing quaternion rotation,” *ACM Transactions on Graphics*, vol. 13, no. 3, pp. 256–276, 1994.

[10] A. Gregory, A. Mascarenhas, S. Ehmann, M. Lin, and D. Manocha, *Six Degree-of-Freedom Haptic Display of Polygonal Models*. T. Ertl and B. Hamann and A. Varshney, 2000, pp. 139–146.

[11] P. J. Berkelman, R. L. Hollis, and D. Baraff, “Interaction with a realtime dynamic environment simulation using a magnetic levitation haptic interface device,” *IEEE International Conference on Robotics and Automation*, pp. 3261 – 3266, 1999.

[12] A. Drif, J. Citérin, and A. Kheddar, “A multilevel haptic display design,” in *IEEE/RSJ International Conference on Intelligent Robotic Systems (IROS)*, 2004.

[13] K. T. McDonnell, “Dynamic subdivision-based solid modeling,” 2000.

[14] E. Guendelman, R. Bridson, and R. Fedkiw, “Nonconvex rigid bodies with stacking,” *ACM SIGGRAPH*, 2001.

[15] K. G. Murty, *Linear Complementary Linear And Nonlinear Programming*. Internet Edition, 1997.

[16] C. Lennerz, E. Schömer, and T. Warken, “A framework for collision detection and response,” in *11th European Simulation Symposium, ESS’99*, 1999, pp. 309–314.

[17] J. Sauer and E. Schömer, “A constraint-based approach to rigid body dynamics for virtual reality applications,” *Proc. ACM Symposium on Virtual Reality Software and Technology*, 1998.

[18] D. E. Stewart, *Time-stepping methods and the mathematics of rigid body dynamics*. Birkhuser, 2000, ch. 9.

[19] A. Lécuyer, “Contribution l’étude des retours haptique et pseudo-haptique et de leur impact sur les simulations d’opérations de montage/démontage en aréonautique,” Ph.D. dissertation, Université Paris XI, 2001.

[20] R. Barzel and A. H. Barr, “A modeling system based on dynamic constraints,” *Computer Graphics*, vol. 22, 1988.

[21] D. Baraff and A. Witkin, “Dynamic simulation of non-penetrating flexible bodies,” *Computer Graphics*, vol. 26, no. 2, pp. 303–308, 1992.

[22] J. Schmidt and H. Niemann, “Using Quaternions for Parametrizing 3–D Rotations in Unconstrained Nonlinear Optimization,” in *Vision, Modeling, and Visualization 2001*, T. Ertl, B. Girod, G. Greiner, H. Niemann, and H.-P. Seidel, Eds. Stuttgart, Germany: AKA/IOS Press, Berlin, Amsterdam, 2001, pp. 399–406.

[23] G. Bouyer, “Rendu haptique et sonore 3d en prototypage virtuel,” 2002.