EUROGRAPHICS Workshop IPT & EGVE Workshop (2005)

Colosseum3D – Authoring framework for Virtual Environments

Anders Backman

VRlab/HPC2N, Umeå University, Sweden

Abstract

This paper describes an authoring environment for real time 3D environments, Colosseum3D. The framework makes it possible to easily create rich virtual environments with rigid-body dynamics, 3D rendering using OpenGL Shaders, 3D sound and human avatars. The creative process of building complex simulators is supported by allowing several authoring paths such as a low level C++ API, an expressive high level file format and a scripting layer. To exemplify the use of the framework, an immersive wheelchair simulator application is presented. A natural and intuitive interaction method is implemented using dynamic simulation.

Categories and Subject Descriptors: I.3.4 [Graphics Utilities]: Software support - I.3.6 [Methodology and Techniques]: Interaction techniques

1. Introduction

Authoring a rich multimodal real time 3D environment can be a daunting task. Virtual Reality (VR) is a category of 3D graphics applications, where the user is immersed into the virtual works using display systems such as head mounted displays (HMD), CAVE [CSD93] or other projection systems. A 3D simulator must perform many complex tasks, such as parsing 3D geometry and drawing it efficiently, simulating dynamic objects, handling collision detection, reading sound files and playing them accordingly to the simulation, handling of system events etc.

This work concentrates on the authoring process of virtual environments with both physical as well as visual content. The framework described, Colosseum3D, is a modular system for building large scale simulation environments, based on existing open source software.

To exemplify the use of the framework, a simulator developed for the department of Community Medicine and Rehabilitation, a wheelchair simulator will be presented. As a result of that simulator an intuitive close up interaction method using dynamics simulation has been developed.

The framework can be described with the initial general goals of the project. We needed a framework that:

- supported a creative process in terms of prototyping
- could simulate a rich dynamic environment
- could support an intuitive and natural interaction method

By defining a general extendable file format, rich scripting capability and exporters from commonly used 3D modeling software we can aid the creative process of building dynamic simulations.

The details of the interaction between objects are handled by the simulation. Also, by using dynamic simulation, the interaction between the immersed user and nearby objects can be made to mimic the way we interact in daily life.

Dynamic behavior in a VE can be created using

- Scripts – triggered by user inputs or collision detection
- Controllers – continuous response of the users input
- Simulation – e.g. dynamics

In practice, a simulation environment is built up using all of the above methods.

A vast majority of the Virtual Environments (VEs) built in the past have been based on static graphics and at most animations triggered by user action. Explicitly

creating an environment, where every single action has to be specified, is a time consuming task. It is a methodology that doesn't scale well when building larger environments because all pair wise interactions have to be specified and it results in strong limitations to realism and interactivity. Implicit design, on the other hand, is when only the initial state is defined; and the subsequent states are calculated using generic interaction rules described in the core system. This method allows users to add objects to the environment without having to describe what will happen to them after the simulation has started. This doesn't suit all purposes as sometimes discreet events such as playing sound, starting motors etc., have to be explicitly designed. However it is still a method that brings life to a simulation without much labor. In our framework the environment is described by objects that have visual and physical attributes, just as in the real world. A physics simulation engine is used to simulate motion.

The result is a framework, written in C++, which is a component based tool making it possible to prototype complex simulations in a very short period of time. To support the paradigm of scripting functionality, a Lua scripting module handler has been implemented. This means that the user can create a simulation with rigid-body dynamics, spatial sound, and natural interaction using a descriptive file format, the Lua scripting language or a C++ API.

Our framework has proven to be intuitive when building real-time applications. For instance it has been used by more than 200 students at Umeå University creating simulations with real-time rigid-body dynamics. We have also implemented an intuitive interaction method that works well with two handed or multi-user interaction. The framework is also used as a research platform for new rendering and simulation methods. Examples are point splatting [ZPvBG01] and fluid simulation using the smoothed particle hydrodynamics method [LL03].

The rest of this document is organized as follows: section 2 describes related work. Section 3 gives a brief description of our architecture. Section 4 is a description of the authoring process. In section 5, we describe the method of interaction that we implemented. This is followed by sample application using our framework in section 6.

Finally, section 7 summarizes the results and describes future work.

2. Previous work

Systems using a static description of rigid-body dynamics simulations have been presented before.

A system using XML as the description language to specify interactions between objects in a VE was described previously [LRR03]. The same authors also presented an example using rigid-body dynamics to drive the core simulation. Glencross et al. [GM99],[GHP01] describes their Iota framework for building physically simulated environments using the Perl language. SiLVIA [LSW99] is a library for building VR applications with collision detection and a constraint based approach for simulating rigid bodies. DBView developed at DaimlerChrysler Research, is a software platform which incorporates collision detection and real-time multi body dynamics [SS98]. Either these systems are for restricted use or are not mature enough for building a complete simulator.

The relation between interaction and presence is studied in [SMND99] and [Sher92]. One of the conclusions is that interactivity can be viewed as one of the key factors in facilitating the feeling of presence. Poupyrev et al. [PIW98] performed a study of several interaction techniques. In their paper they state that the interaction technique defines the "look-and-feel" of VEs. The interaction task in our simulator was rather restricted; the user should only be able to interact directly with the objects in the most natural way, not via menus, buttons or any other GUI widget. This meant that most of the grasping and manipulating methods developed in the last few years such as aperture [FHZ96], world-in-miniature [SCP95], image plane, [PFC*97], and go-go [PBWI96] did not suit our needs. Frölich et al. [FTBB00] describe a system, where the user can interact with physically simulated objects using a spring-based approach. This approach also works well with multi-user/multi-hand environments which is a great benefit. Zachmann [ZR01] presents a number of algorithms to handle collision, dynamics and grasping of objects for assembly process simulations. Many of the existing interaction techniques do not use the precision of the human hand and fingers. In [HH03] Hirota et al. describe a system where they track multiple fingers and use a collision response computation method from real-time haptic rendering to calculate the forces used to manipulate dynamically simulated objects.

Abstracting the actual code of the application from the final devices that will be used in run-time is an area of active research. In [KBH00] presented a framework for building VEs independent of the target interaction and projection hardware. VRjuggler [BJH*01] is a similar approach where the application can be written independently of the target hardware. Neither of these projects targets the process of content generation of VEs.

3. Architecture

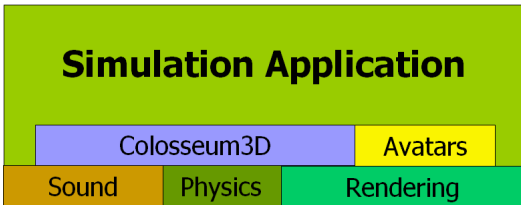

Figure 1: Components of Colosseum3D

Building a multimodal simulator is a large task, and we realized early when designing the framework that most of the work is already done in other projects and toolkits. By using the best available libraries already existing, we could benefit from the excellent work of others. Although the integration of different toolkits still demands a lot of work as there are several different representations of data. For example the rendering toolkit uses an internal representation of geometries that is separated from the geometry representation of the rigid-body toolkit.

3.1 Rendering

We use OpenSceneGraph (OSG) [Osg05] as the rendering toolkit. OSG is an open source scenegraph written in C++ that is portable between most platforms.

3.2 Rigid body dynamics

Vortex [Vor05] from CM-Labs is a fully featured rigid-body dynamics simulation toolkit with a C API. As we have implemented a C++ layer for the dynamic simulation, we have the possibility of switching dynamics toolkit if we decide to. We are also working on the integration of Open Dynamics Engine (ODE) [Ode05] which would make all the ingoing parts of the framework open source based.

3.3 3D Sound

Currently we are using OpenAL [Ope05] to handle 3D sound in the framework. OpenAL is a portable API for spatial sound with a programming interface similar to OpenGL. On top of OpenAL we have developed OpenAL++ [Häm02] which is an object oriented abstraction layer written in C++. An OpenAL++ sound source can be attached to any rendering node in the scene graph to make the source follow the node as it is moved around.

One limitation of using hardware rendered sound is that there is usually an upper bound of the number of sound-sources that simultaneously can be in use. Under Windows this number is usually around 32. To support the idea of contact sound, where hundreds of sound events can be generated in a short time period, we implemented a SoundManager. The sound manager is a broker that handles the requests for sound events in a first-in-first out order. Looping sound such as ambient sounds and music is handled by allocating a sound source for exclusive access. A sound can either be a sampled sound stored to disk in uncompressed wave pcm, the compressed Ogg Vorbis [Ogg05] format or a streamed sound over the internet.

3.4 Avatars

Cal3D [Cal05] is a skeleton based 3D character animation library written in C++. Our framework extends the toolkit with features such as queuing and blending of animations and ground following in a module named ReplicantBody [Sun02]

3.5 Scripting

For scripting of actions and events and even creation of objects we are using Lua [Lua05]. Lua is well suited for embedding into real time applications such as games and simulators due to its efficiency and small memory footprint. To expose all C++ classes to Lua we are using Tolua++ [Tol05] which is a small utility library written in Lua to parse C++ headers and create integration code that enables the use of C++ code from within Lua.

3.6 Execution

The execution of a simulation in Colosseum3D can be divided into three steps:

- Parsing the description file and generating a database with descriptions.
- Accessing the DB creating the declared objects, materials and constraints.
- Time stepping the simulation in the overall execution loop of the system.

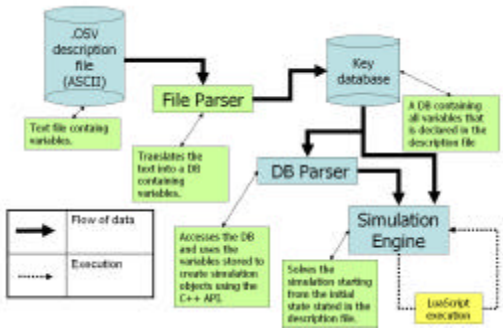

Figure 2: Overall execution of a simulation

3.7 Multithreading

The time take to step the rigid body system by a given time interval depends on many things: such as the number of interacting objects, object complexity, the number of degree of freedoms in the constraints and the hardware the simulation runs on. Running the dynamics solver in serial with the rendering system can cause the simulation to appear jerky and could also result in an unnecessarily low frame rate. In a system with many interacting objects where each object is simple to render, the dynamic solver would be the bottleneck for the throughput of the system. A high and consistent frame rate is desirable in a HMD based display system. Ideally the scene should be updated at the refresh rate of the display, usually between 60 and 85 Hz. Therefore we are running the dynamics and the rendering loop asynchronously in different threads. Synchronization only occurs in between two rendering frames. It is also possible to run the dynamics and rendering processes in the same thread, thus making it a serial application. We have found that it is not always optimal to have the rendering and dynamics process in separate threads. This is especially true on a single processor system. The synchronization of data and the context switches introduce overhead which degrades the performance compared to serial execution. Multithreading works when there is a clear balance between rendering and the dynamics simulation. Therefore multithreading is an option left for the developer of the simulation. We hope to implement a dynamic load balancing in the future.

A good interactive application should respond quickly and deterministically to user input and should ideally be a hard real-time system with an update frequency of at last 15 frames per second as reported by Möller et al [AH02].

We have implemented a best-effort real-time system—that is, a system that tries to meet timing constraints but cannot guarantee this a priori. A simulation is said to run in real time when a simulation step Dt_s takes the same amount of clock time t_c to execute as it steps the simulation forward. As the time to solve a simulation step is not deterministic we have put a controller on the dynamics thread. This means that the thread itself is responsible for stepping the system forward in real-time.

If $Dt_s < t_c$ then the simulation runs slower than real-time and a slow motion effect is achieved and if $Dt_s > t_c$ is true, we have to wait (sleep) for $Dt_s - t_c$ seconds to achieve real-time. Instead of sleep, we could also choose to take several small steps to obtain better accuracy. To obtain real-time when $Dt_s < t_c$ some LOD scheme for the simulation have to be implemented. Another way to obtaining this would be to allow the length of Dt_s to vary within a certain reasonable bound

4. Authoring

When writing a simulator the data should be separated from the operations operating on it. There are many reasons for this; one is that changing the precision of the data should be easily done without changing the functionality of the system. For example, the number of polygons a graphical rendering system can handle increases for each new generation of graphics cards. This also enables the author of the simulator to concentrate on the structural content of the simulation, and leaving the details for later.

In Colosseum3D a simulation is created by declaring a *state*. A state can be seen as a snapshot of the simulation at time t_0. When the simulation starts, successive states are calculated by the simulation engine. Each state describes objects with their visual and their physical attributes at a given time. The visual attribute can be as simple as referencing an existing 3D model in a file format supported by the rendering tool. The physical attribute describes the attributes necessary to simulate the object in a rigid body simulation. This includes mass, material, geometry, etc. An object does not have to have both visual and physical attributes. A pure physical object can be created simply by omitting the visual attributes. Part of the physical description of the system is also joints. These places a constraint on any attached object. A joint can for example describe a hinge, where an object is free to rotate around a given axis.

4.1 Descriptive file format

An alternative for storing a simulation state would be to use XML [Xml05]. The biggest drawbacks with using XML as an authoring format is the overhead and complexity of writing XML code by hand. One of the

most important requirements we had was that it should be easy and intuitive to create simulations using the descriptive file format. We choose to develop a simple but expressive file format with enough generality to support the features we needed.

The simulation state can be entered into this descriptive file format. It is an extendable file format which consists of keys, and its associated data. Supported data formats are INT, FLOAT, VECTOR, STRINGVECTOR, STRUCTURE, STRING and FLOAT EXPRESSIONS. These description files can be used to declare any initial data used by the application. All keys and data is parsed and put into a database which can later be accessed by the application at any given time. An example of an object declaration is given in Listing 1. We have scoped out a few important key properties for a descriptive file format for dynamic simulations:

- machine readable/writable
- human readable/writable
- general
- independent of target platform
- expressive
- low learning threshold
- supports a creative process
- guarantees consistency

Machine readable is of course important when we want to have an efficient way of interpreting a description given to the system. The format also need to be machine writable to enable the use of external tools to generate the simulation description, also during simulation it is important that one can store the current simulation state for playback purposes; *human readable/writable* is important for letting users write their own state descriptions, read other users descriptions, find bugs and merge the work with others; the format should support *general* data with few or no limitations to what it can describe; the descriptive format should also be *independent of the target platform* on which it should execute. One example is that, the basic file format should not describe the actual geometry that is going to be rendered, as vertices, edges, polygons etc. The optimal resolution for a geometry used varies with different hardware solutions and generations. Therefore the description format should only reference the actual geometrical description. It is up to the rendering engine to make use of that information the way it prefers; by having an *expressive* description format, the number of lines describing a simulation decreases. A descriptive format also *lowers the learning threshold* for students working with the creation of simulation environment. By having appropriate default values, a simple simulation can be created with very little effort. For the more advanced user, all the parameters of the simulation is available.

By having a variety of parameters and giving the user freedom to do whatever he or she wants while always informing the user of any errors made, we *support the creative process*. The file interpreter *guarantees a consistent* and correct use of the internal API for instantiating the various objects thus reducing the risk of errors.

```
s 10.0 // a scalar value
Object { // Description of a new object
  ID “a sphere object”
  Position [10.2 34.2 s] // uses value of s
  Size [randInterval(s/2, s)] // Random number
  VisualAttributes {
     Geometry {
       Primitive “sphere”
    }
  }
  PhysicalAttributes {
    Mass 1.1
    Material 2
    Geometry {
      Primitive “sphere”
    }
 }
```

Listing 1: Example of object declaration

4.2 Plugin modules

The creative process of authoring a simulation should not be hampered by the tools in use. We tried to implement several paths of creating a simulation using the framework. Currently, an author of a simulator using the framework can describe the simulation state using the descriptive simulation file format, the Lua scripting interface and through a C++ API. The Lua scripting language is an established interpretating language suitable for embedded applications. Through the Lua interface, the author can access the exported parts of the C++ API. This supports a fast prototyping process whereas the application does not have to be recompiled due to code changes. The Lua interface is implemented as dynamic library modules which can be loaded during run-time as seen in Listing 2. No all functionality can or even should be exported to Lua, due to performance issues and that there is not a one-to-one match between Lua and C++.

```
requestPlugin(“physics”)
requrestPlugin(“osg”)
requrestPlugin(“osgsim”)
requrestPlugin(“openalpp”)
```

Listing 2: Loading plugin modules from Lua

4.3 OpenGL shader development

OpenGL Shading Language (GLSlang) is a high level language used to partly replace the fixed pipeline of OpenGL. It can be used to access vertex and fragment primitives efficiently on the graphics processor unit (GPU). As OpenSceneGraph supports GLSlang by default, full access to all of the features of GLSlang could be exposed to any Lua script. By adding the required declarations to the descriptive file format, shaders could be declared and used as visual attribute for any object. Listing 3 and Listing 4 are examples of how a shader could be declared and used when rendering an object. The uniform variables in the shader can later be accessed from a Lua script during runtime to change the appearance of the visual object.

```
// toon Shader
VertexShader {
  ID "toon_vertex"
  File "shaders/toon.vert"
}
FragmentShader {
  ID "toon_fragment"
  File "shaders/toon.frag"
}
ShaderProgram {
  ID "toon"
  VertexShader "toon_vertex"
  FragmentShader "toon_fragment"
  Enable 1
}
ShaderState {
  ID “toon”
  ShaderProgram “toon”
  Uniform {
    ID "DiffuseColor"
    Value [0 0.25 1]
  }
  Uniform {
    ID "PhongColor"
    Value [0.75 0.75 1]
  }
  Uniform {
    ID "Phong"
    Value 0.98
  }
}
```

Listing 3: Shader declaration

```
Object {
  ID "cow"
  Position [1  0  2.7]
  Orientation [0 0 0]
  Dynamic 1
  VisualAttributes {
    Shadower 1
    Geometry {
      File "cow.osg"
      Scale [0.1 0.1 0.1]
    }
    ShaderState "toon"
  }
  PhysicalAttributes {
    Material 1
    Mass 2
    Geometry {
     ParseMethod “TriangleMesh"
    }
  }
}
```

Listing 4: Use of shader declaration

4.4 Visual programming.

As a large simulation can contain hundreds or even, thousands of objects, the process of entering each object into a descriptive file format can quickly become time consuming and prone to errors. Therefore a more efficient way of creating objects must also be supported.

Currently we are using a 3D modeling tool, Creator from Multigen Paradigm, to create larger scenes. The physical attributes are entered as text into the comments field for a node. Later on when that part of the scene is read, it is parsed and searched for attributes. Any physical attribute is found and the associated object is created. Figure 3 **s**hows a part of a scene where both the visual representations as well as the physical representations of static objects are shown. This makes it easier to orientate and place physical object as it supports the WYSIWYG paradigm of creating scenes.

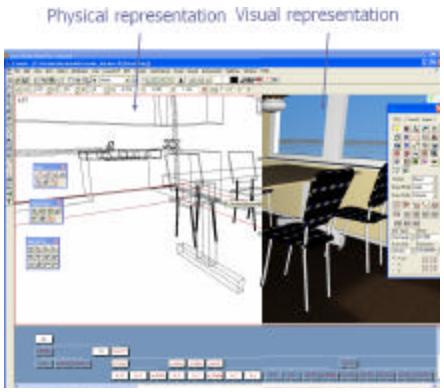

Figure 3: Using Multigen Paradigm Creator for building physical/visual environments

5. Natural interaction in a VE

The hardware setup for the interaction task is the following:

Ascention MotionStar™ using 3 sensors

Fakespace PinchGlove™

Virtual Research V8 HMD

Dual AMD Athlon MP 2000+

NVIDIA FX5900 Ultra

5.1 Direct manipulation

The interaction tasks in our simulator are designed to be in direct correspondence to the way we interact in our daily life. To achieve this we use a direct manipulation method using a "Classical" virtual hand that according to Poupyrev et al. [PIW98] is classified as an egocentric metaphor for manipulation. As all objects in the environment are simulated with rigid body dynamics, we also wanted the user's virtual hands to be simulated in the same manner. We make use of a constraint in Vortex called Relative Position Relative Orientation (RPRO). It is a spherical constraint (ball and socket) which provides full kinematic control of the constraint. This means that we can control the constraint via a 6 DOF motion sensor.

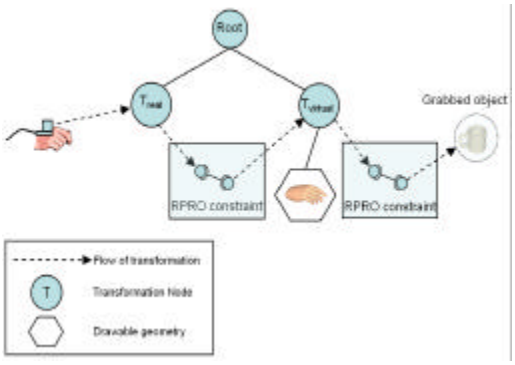

Figure 4: Schematic view of the interaction structure

The interaction we have implemented can be divided into several parts:

- Indication/Grasping
- Manipulation
- Real-virtual discrepancy
- Navigation

5.2 Indication/Grasping

Indication is a two way process. Either the system informs the user that a hand is touching an object that can be manipulated, this is done with an opened hand (Figure 5b). Or the user indicates using the PinchGlove™, that the hand will grasp any object touching the hand (Figure 5c).

Figure 5: Indication of interaction modes. a) Hand in neutral mode, no object in contact with the hand. b) Hand in contact mode, an object in contact with the hand. c) Hand in grasp mode, an object is attached to the hand.

A very important benefit of using a constraint based method for interacting with objects is that an object can also be grasped using both hands, or by several users. The dynamics solver will try to stabilize the system, which can be illustrated as there are several springs connected to the object, and they each strive to get to its rest position. We have chosen not to distinguish be-

tween different types of grasping as Zachmann[ZR01]. This means that some of the precision of the human hand is lost, but to the benefit of being easy to implement.

Figure 6: Two handed interaction.

5.3 Manipulation.

When an object is grasped, the object follows the hand as it moves. Both the hand and the grasped object are still physically simulated which means that they can both be used to interact with other objects.

5.4 Real-virtual discrepancy.

The virtual hand is physically simulated, i.e. it responds to the constraints added by the physics module when contacts are created. This means that the virtual hand can get stuck while the real hand continues to move. Imagine that the user places the virtual hand under a table, the virtual hand will stop, but the real hand with the 6 DOF sensor will continue through. Zachmann tries to solve this problem with a "ghost" object, an object which marks the position where the object would really be if there was no collision. The "real" object, i.e. the object we are manipulating, can then be attached to the ghost in at least three ways:

- The rubber band metaphor: the object is connected via a rubber band to the ghost.
- Rubber band and spiral spring. This resembles a method of Frölich et al. [FTBB00] uses.
- Incremental motion: The object will incrementally move according to the ghost unless there is a collision. If there is a collision, the simulation will determine the new direction.

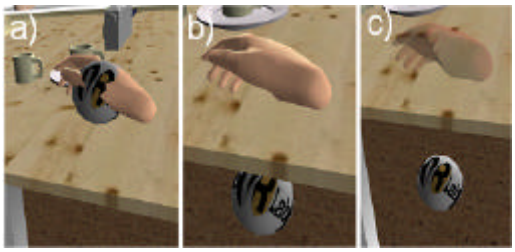

Figure 7: Textured ball indicates the position of the real hand. a) Real and virtual hand are coincident. b) Real hand has "penetrated" the virtual table. c) Virtual hand is drawn in transparent mode.

We choose a slightly different approach. When the distance between the real and the virtual hand is larger than a certain threshold *deviationDistance* (we used 0.3 meter in our experiments) we disable the collision calculations for the hand. This makes the virtual hand snap back to its correct place, namely where the user is holding his hand. When the distance is below certain threshold (empirically derived) (0.1**deviationDistance*) we enable the collision again. To give the user additional information we also choose to render the hand transparently and on top of all other objects when the distance is in the interval of [0.1**deviationDistance*, 0.7**DeviationDistance*]. This keeps the user informed of where the hands are at any given time provided they are looking in the direction of the hand. The reason for using the upper limit of 0.7*DeviationDistance is that we want to avoid an unsteady state, where the hand would flicker due to lag in the physical constraint used for positioning the dynamic object representing the hand. Figure 7c shows the situation just before the collision calculation of the virtual hand is disabled which makes the hand move through the table and snap back to its appropriate position.

5.5 Navigation

The task of navigation in a VE is defined as the control of user viewpoint motion [BOW97]. In our simulator the navigation context was already clear from the beginning of the project; the user would sit in a physical wheelchair and navigate using the wheels. Therefore we use a rig where a real wheelchair is elevated with the wheels being able to rotate freely. On each wheel we use a counter to measure the movement of the wheel. In the simulation we have modeled a virtual wheelchair with the same physical attributes as the real one using Colosseum3D script. All moving parts such as the front wheels with their hinges and the rear wheels are all simulated. The rear wheels are specified as motorized hinges, which, in turn are controlled by the counters on the real wheelchair. By calculating the velocity of the

real wheels and setting the desired velocity of the hinges driving the back wheels in the simulation, any movement of the real wheel is correctly transferred to the virtual one.

6. Sample application

The Stroke Simulator [MBH*04] is a research project at the department of Geriatrics, Umeå University where we investigate how an interactive and immersive VE can induce empathy among relatives, family and health care staff about the everyday life experiences of a stroke patient. The user of the simulator is seated in a physical wheelchair wearing PinchGloves™, a HMD and 6DOF motion sensors for navigation and interaction. The VE is an ordinary apartment where the user has to perform everyday tasks such as sweeping the floor, setting a table for breakfast. The environment contains dynamically simulated objects such as beverages, serial packages, broom, dust pans, forks, knives etc. To simulate the effect of a stroke, certain anomalies can be switched on and off, e.g. double vision, motion blur, perspective anomalies, and effects from unilateral neglect that will show the environment from the stroke patient's perspective. The Stroke Simulator is developed using the Colosseum3D framework and make use of the dynamic simulation to describe the environment both visually and physically. The user sits in a physical wheelchair which is also used for navigating the VE.

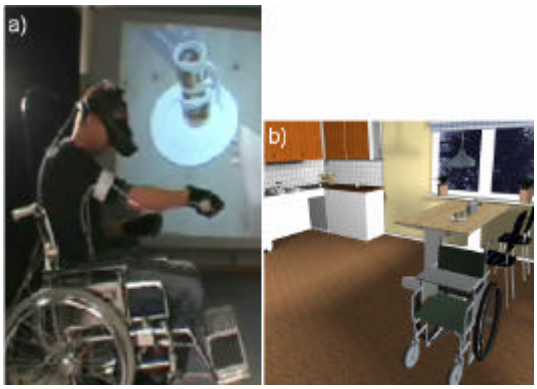

Figure 8: User manipulating objects in the Stroke Simulator. b) Screenshot from the environment.

7. Conclusion and future work

We have described a framework for building interactive real-time simulations with rigid-body dynamics. The framework has successfully been used for implementing a simulator that uses an intuitive and natural interaction and navigation method. The goal of making the framework easy to use have been proven by the fact it have been used by more than 150 students at Umeå University which all successfully managed to build complex simulations rigid-body dynamics and interactive 3D sound.

One problem with the implementation of the wheelchair navigation model is the wheel sensor control. As we measure the velocity of the wheels of the real wheelchair and feed that into the simulation, a wheel on the virtual wheelchair can never rotate unless the real one does. This means that if the virtual wheelchair is placed in a downward slope, it will not start to roll due to the forces of gravity. To achieve this, a force feedback loop with motors on the real wheelchair would be required. We have done some preliminary tests with good results. This is something that has to be developed further. It would also greatly enhance the experience of the simulator as the user would experience any collision of the wheelchair against obstacles and therefore help in the task of navigation.

To better support the authoring process of building immersive virtual environments, we will incorporate support for the Collada file format [Col05] which is a format where several industrial parties have collaborated to define a standard specification that contains both rendering and simulation aspects. By supporting this format, it is possible to author simulations from 3D tools such as Maya and 3DSMax.

A more detailed user study of the usability of the simulator is beyond the scope of this paper and will be presented in future articles.

Acknowledgements

We would like to thank Claude Lacoursière for his invaluable advice regarding dynamic simulations and Peter Sunna and Daniel Sjölie for their great programming effort. The project wouldn't have been possible without the inspiration from Gösta Bucht and Marcus Maxhall. Finally thanks to the OpenSceneGraph community and especially Robert Osfield for great feedback on OSG related questions.

References

[AH02] AKENINE-MÖLLER T, HAINES E., Real-time rendering, Second Edition, *A K Peters Ltd.*, 2002.

[BJH*01] BIERBAUM A, JUST C., HARTLING P., MEINERT K., BAKER A., CRUZ-NIERA C ., VRJuggler: A Virtual Platform for Virtual Reality Application Development, *IEEE Virtual Reality Conference, Yokohama*, March 2001, Japan, pp. 89-96.

[BOW97] BOWMAN D. A., KOLLER D., HODGES L. F., Travel In Immersive Virtual Environments: An Evaluation of Viewpoint Motion Control Techniques, *IEEE Proceedings of VRAIS'97,* pp. 45-52.

[Cal05] CAL3D, http//cal3d.sourceforge.net, 05/05/18

[Col05] COLLADA, http://www.collada.org, 05/05/18

[CSD93] CRUZ-NIERA C., SANDIN D., DEFANTI T., Surround-screen Projection-based Virtual Reality: The Design and Implementation of the CAVE. *In Computer Graphics, Proceedings of SIGGRAPH 93*, August 1993.

[FHZ96] FORSBERG A., HERNDON K., ZELEZNIK R., Aparture based selection for immersive virtual environment, *Proceedings of UIST'96"*, 1996, pp. 95-96.

[FTBB00] FRÖHLICH B., TRAMBEREND H., BEERS A., AGRAWALA M., BARAFF D., Physically-based manipulation on the responsive workbench, *In Proceedings IEEE Virtual Reality 2000*, 2000.

[GHP01] GLENCROSS M., HOWARD T., PETTIFER S., Iota: An Approach to Physically-Based Modelling in Virtual Environments, *IEEE Virtual Reality Conference, Yokohama*, March 2001, Japan, pp. 287-288.

[GM99] GLENCROSS M. , MURTA A., A Virtual Jacob's Ladder, *GraphiCon-99,* Moscow, ISBN: 5-89209-436-7, August 1999, pp.88-94.

[HH03] HIROTA K., HIROSE M., Dexterous Object Manipulation Based on Collision Response, *In Proceedings IEEE Virtual Reality 2003,* 2003.

[Häm02] HÄMÄLÄ T., OpenAL++ - An Object oriented toolkit for real-time spatial sound, *Master's thesis UMNAD 391,* Department of Computing Science, Umeå University, 2002.

[KBH00] KESSLER G. D., BOWMAN D., A., HODGES L., F., The Simple Virtual Environment Library: An Extensible Framework for building VE Applications, *Presence.* Vol. 9, No. 2. April 2000, pp. 187-208

[LL03] LIU G. R., LIU M. B., Smoothed Particle Hydrodynamics: A Meshfree Particle Method, *World Scientific Pub Co Inc*, 2003

[LRR03] VAN LAERHOVEN T., RAYMAEKERS C., VAN REETH F., Generalized Object Interactions in a Component based Simulation Environment, *Journal of Winter Conference on Computer Graphics 2003*, 2003, pp. 472-479.

[LSW99] LENNERZ C., SCHOEMER E., WARKEN T., A Framework for Collision Detection and Response, *In 11th European Simulation Symposium*, ESS'99, 1999, pp. 309-314.

[Lua05] LUA, http://www.lua.org, 05/05/18

[MBH*04] Maxhall M., Backman A., Holmlund K., Hedman L., Sondell B., Bucht G. Caregiver responses to a stroke training simulator, *Proceedings of ICDVRAT2004 conference*, New College, Oxford, UK, 20-22 September 2004

[Ode05] OPEN DYNAMICS ENGINE ODE, http://opende.sourceforge.net/, 18/5/2005

[Ogg05] OGG VORBIS, http://www.vorbis.com/, 05/05/18

[Ope05] OPENAL, http://www.openal.org, 05/05/18

[Osg05] OPENSCENEGRAPH, http://openscenegraph.sourceforge.net, 05/05/18

[PBWI96] Poupyrev I., Billinghurst M., Weghorst S., Ichikawa T., Go-Go Interaction Technique: Non-Linear Mapping for Direct Manipulation in VR, *Proceedings of UIST'96"*, 1996, pp 79-80.

[PFC*97] Pierce J., Forsberg A., Conway M., Hong S., Zeleznik R., Mine M., Image plane interaction techniques in 3D immersive environments, *Proceedings of Symposium on Interactive 3D Graphics"*, 1997.

[PIW98] POUPYREV I., ICHIKAWA T., WEGHORST S., BILLINGHURST M., Egocentric Object Manipulation in Virtual Environments: Empirical Evaluation of Interaction Techniques, *Computer Graphics Forum*, 17(3), 1998, pp. 41-52.

[SCP95] STOAKLEY R., CONWAY M., PAUSCH R., Virtual reality on a WIM: interactive worlds in miniature, *Proceedings of CHI'95*, 1995, pp. 265-272.

[Sher92] SHERIDAN, T. B., Musings on Telepresence and Virtual Presence, *Presence: Teleoperators and Virtual Environments*, 1(1), 1992, pp. 120-126.

[SMND99] SCHUEMIE M.J. VAN DER MAST C.A.P.G, NIJHOLT A., DONK O., VAN DIJK B., (EDS.),Presence: Interacting in VR?, *Proceedings Twente Workshop on Language Technology 15*, TWLT 15, ISSN 0929-0672, 1999, pp. 213-217.

[SS98] SAUER J., SCHÖMER E., A constraint based approach to rigid body dynamics for virtual reality applications, "*Proceedings of the ACM Symposium on Virtual Reality Software and Technology*", 1998

[Sun02] SUNNA P., "Real-time Character Animation", *Master´s thesis UMNAD 397*, Department of Computing Science, Umeå University, 2002

[Tol05] TOLUA++, http://www.codenix.com/~tolua/, 05/05/18

[ZPvBG01] ZWICKER M., PFISTER H P., VAN BAAR J., GROSS M., Surface splatting. In proceedings of ACM SIGGRAPH 2001, *Computer Graphics Proceedings*, 2001, pp. 371-378

[ZR01] ZACHMANN G., RETTIG A., Natural and Robust Interaction in Virtual Assembly Simulation, *Eighth ISPE International Conference on Concurrent Engineering: Research and Applications (ISPE/CE2001)*. West Coast Anaheim Hotel, California, USA, July 2001.

[Vor05] VORTEX, Real-time simulation toolkit, Cmlabs,

http://www.cm-labs.com/, 18/5/2005

[Xml05] EXTENSIBLE MARKUP LANGUAGE (XML)

http://www.w3.org/XML/, 05/05/18