A Software Environment for the Responsive Workbench

Michal Koutek and Frits H. Post

Delft University of Technology, Faculty of Information Technology and Systems

email: M.Koutek@cs.tudelft.nl, F.H.Post@cs.tudelft.nl

Keywords: Responsive Workbench, RWB Library, RWB Simulator

Abstract

In this paper we present a software environment for the Responsive Workbench (RWB). We will give a technical information on our RWB system. We will describe the architecture and the usage of the RWB library, interaction tools, and the RWB simulator. Finally we will show some visualization applications on the RWB.

1 Introduction

The Virtual Reality Responsive Workbench (RWB) is a powerful system for 3D visualization and interaction [1]. It intensifies perception of 3D models and data. The RWB is a semi-immersive virtual environment: the user stands in the real world and is looking into a virtual world which is projected on the screen of the Workbench. One of the advantages of the RWB is its tabletop metaphor. It creates for the user an illusion of a laboratory table or a design studio, while the only real element is the wooden construction of the workbench everything else is purely virtual. The RWB also offers a large screen to visualize the 3D models. Combined 2D and 3D interfaces can be used for the user interaction.

The RWB is complementary to the CAVE [3], where the user is almost fully immersed into the projected virtual environment. The usage of the RWB is a bit different than of the CAVE. The RWB benefits from the table metaphor although its field of view is rather limited. In the CAVE, all objects are usually virtual. In automotive industry they put mock-ups, car-seats inside the CAVE to have at least something real with a substance, but they have to face the problems of interference with electro-magnetic tracking and thus wooden or plastic materials have to be used. If we would want to use the CAVE in the same way as the RWB we would have to display a virtual table as well. For some application is the RWB more suitable than the CAVE.

For controlling these types of VR systems VR software and libraries are needed. Two years ago, when the RWB facility was installed at the HPaC Centre at TU Delft, there were not many software options. On one hand there were a few experimental libraries (like Avocado/Avango [2], VR-Lib, MR Toolkit, VrTool) used by VR researchers and the other hand there was a commercial software, like the CAVE library [9] or the WorldToolKit (WTK). Usually, the commercial VR software is not an ultimate option for a VR researcher.

We have chosen a third option. Based on the experiences with the experimental software, we have built our own VR software infrastructure that satisfied our research needs. Our RWB library is based on OpenGL and Iris Performer and has been implemented in C++. A customizing of the CAVE library or of the WTK for our installation of the RWB would be much a harder task.

In this paper we want to give an overview of the framework and the architecture of the Responsive Workbench, the way this VR system works and how to use the RWB library and the RWB simulator.

2 RWB Basics

The Responsive Workbench is based on a stereo projection table system which is combined with an electro-magnetic tracking system, see Figure 1. The stereo images are generated from a powerful graphic station, the SGI ONYX 2 with four CPUs and an Infinite Reality 2 graphic card. We use the display resolution of 1120×840 pixels in 96 Hz in stereo mode.

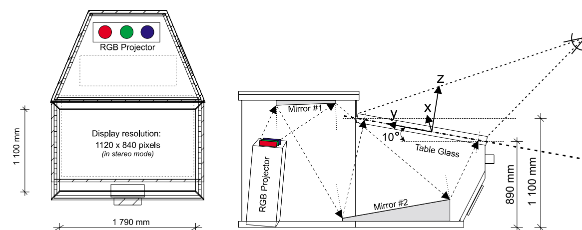

Figure 1: Top and side views on the Workbench

The image from the RGB projector is reflected though two mirrors and has to properly fit on the table glass which is tilted by 10°. To obtain a clear and a sharp image the RGB projector has to be well

calibrated and precisely aligned. Crystal Eyes shutter glasses are used to see the stereo effect.

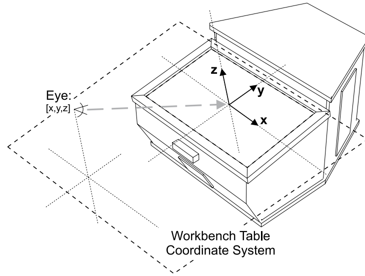

Figure 2: Workbench table coordinate system

The virtual world projected on the screen of the RWB is represented in the workbench table coordinates. The tracked positions and orientations of the user's head and hand must be converted into the same table coordinate system. We are using table-centered and table-aligned coordinate system, see Figure 2.

In the RWB environment, the head-tracker updates user's viewpoint, and the tracking of the stylus pen forms the base for a 3D interaction, see Figure 3.

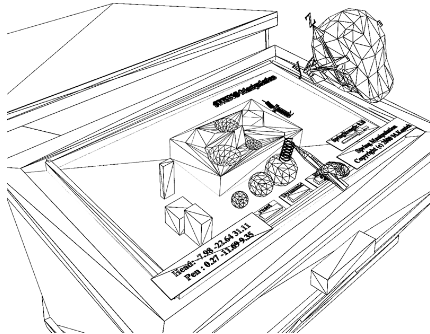

Figure 3: 3D user interaction on the RWB

2.1 Projection and Viewing

In common 3D rendering systems the user's eye is positioned on the axis of perspective projection, so called on-axis-perspective. This is usually the case for a user watching the monitor on which a 3D geometry is displayed. Usually we put the viewpoint on the +Z-axis and the viewing frustum is oriented into -Z-axis. Of course, it is an ideal case when the user has his/her head on the central axis of the screen. For monoscopic images it is not a problem.

But on the Workbench, we cannot assume that user's eye is on the Z-axis. Therefore we have to use an off-axis-perspective. Construction of the RWB perspective frustum is shown in Figure 4.

We have to set up the perspective frustum from the user's eye position (in table coordinates) pointing down, perpendicular to the the Workbench ground plane. After the perspective transformation we have to perform a 2D shift in viewport coordinates to fixate the viewport origin with the origin of the RWB. This is equivalent to the assumption that the user is always looking at the center of the Workbench. By this we have constantly fixed the ground plane of the projected VR world to the screen of the Workbench. This has to be done for the left and the right eye to set the correct stereo perspective.

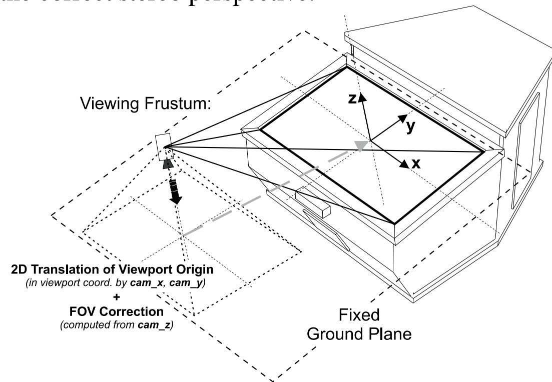

Figure 4: Construction of the viewing frustum

It's important to mention that Iris Performer uses a different notation of viewing direction than usual. Performer's viewing direction is the +Y-axis, and OpenGL uses the -Z-axis.

2.2 Tracking

The electro-magnetic tracking system, in our case the Polhemus Fastrak, measures the position (xyz) and the orientation (AER: azimuth, elevation, roll) of two sensors: the head and the stylus. For a later use we must convert orientations to the Performer's angle notation (HPR: head, pitch, roll).

The tracker daemon process reads periodically (50Hz) the data from the tracking system, converts them to the workbench table coordinate system and stores them in a shared memory. The tracker daemon also offers functions to access and read the data for any running process. More about the tracking system in section 2.3.

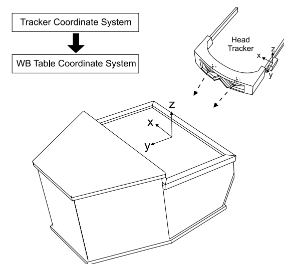

Figure 5: Tracker to table coordinate transformation

The update-view function of the RWB library reads the new position and orientation from the tracker daemon's shared memory. This information is first stored in the tracker-coordinate-system in the form of 4x4 matrix where the position is placed in the translational part of the matrix and the HPR rotation is stored in the rotational 3x3 sub-matrix. This matrix forms the frame of transformation. But originally, this frame is defined with respect to the tracker-coordinate-system, thus it has to be transformed to the workbench-table-coordinate-system, see Figure 5.

Therefore a tracker-to-table transformation 4x4 matrix is used. The original head-tracker frame is transformed by this matrix. The resulting frame has to be aligned so that in the neutral orientation of the head tracker, resp. stylus, is its Z-direction pointing upwards in the virtual world coordinates. This is done by multiplying the frame with pre- and post- rotational matrices. The final frame together with the information on the offsets of the eyes is used to position the viewpoints of the left and the right eyes. The viewing direction is always pointing towards the center of the RWB table, see Figures 5 and 6.

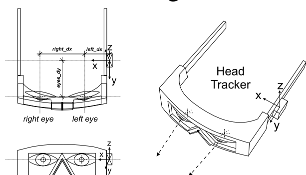

Figure 6: Head tracking and eyes positions

2.3 Calibration

The calibration of the tracking system is a very important process, see Figure 7. It consists of two stages.

First, the tracker-to-table transformation has to be defined. The user clicks with the stylus pen on the 3 points $\oplus$: the origin and points on the X and Y axis of which we know the exact position on the screen (in pixels) and their exact position with respect to the center of the glass-plane (in cm). We measure the positions in tracker coordinates. From this we can obtain the scaling factors, the axis-vectors XY and the Z-axis which is the cross product of X and Y, and the position of the origin with the respect to the tracker-coordinate-system. From these values the ortho-normal frame base (4x4 matrix) is constructed and its inversion gives the tracker-to-table transformation matrix:

$$M_{TT} = \left[\begin{pmatrix} X_i & X_j & X_k & 0 \\ Y_i & Y_j & Y_k & 0 \\ Z_i & Z_j & Z_k & 0 \\ o_i & o_j & o_k & 1 \end{pmatrix} \star M_{scale} \right]^{-1}$$

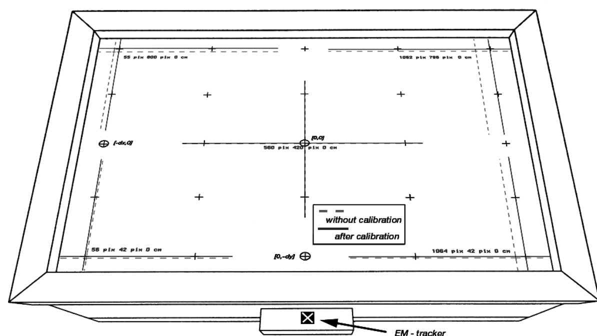

Figure 7: Grid calibration of the tracker data

Next, the grid calibration is performed by measuring the tracking error on the grid. We use a bi-linear interpolation to correct for this error.

In Figure 7, the dashed lines show tracking results in the glass plane without the grid calibration. The tracking error is very annoying especially for 3D interactions with the stylus pen when the virtual cursor is sometimes significantly displaced from the stylus position. For the head-tracking the tracking error is less significant.

3 The RWB Library

Considering the available hardware and our needs we have built the RWB library. We want to use the RWB facility in an optimal way. There is a powerful IR2 graphics in the SGI Onyx 2, which can theoretically render 13 million triangles, and which has a large texture memory. This system is equipped with 4 CPUs and a main memory of $512MB$. These resources should be used effectively by the RWB application.

The typical RWB application needs real time 3D graphics and 3D interaction. Therefore we have chosen for Iris Performer which offers an optimized 3D graphics pipeline based on OpenGL, and also includes an extensive support for multiprocessing and shared memory access in IRIX 6.5 operating system.

The tracker daemon is a separate application which just reads the tracking data from the tracking systems and converts them to the workbench coordinate system. On the side of the RWB library there are functions to read the tracker data from the tracker daemon's shared memory. Advantage of such a solution is that multiple applications/processes can access the shared memory without using HW ports of the tracking system.

Second to solve was to set up the stereo-projection pipeline using the off-axis perspective projection, as discussed above. We have tried several stereo projection schemes including the on-axis perspective which we had proved not to be suitable for the RWB, especially because of the distorted perspective which was a serious problem for 3D interaction with the stylus pen. During this process we had to do many calibrations of our system.

Then we had to define a multiprocessing scheme for a general RWB application, which consists of the main RWB process, the tracker daemon, the keyboard/mouse interaction process and the user application process. The main RWB process consists of application, culling and draw sub-processes as defined in Iris Performer. All the processes can communicate through the shared memory. For this purpose we have created unified shared memory object called "Shared". All the necessary parameters of an RWB application can be accessed from there. It is a responsibility for the application programmer to work carefully with global, local and shared variables. In many cases the problem of a crashing application is hidden in undefined values of variables in forked processes.

Iris Performer together with our RWB library offers many functions to access 3D geometry files or for creating a custom geometry and building the scene graph of the virtual world which is displayed on the RWB.

We have incorporated a generic class, the RWB-obj, which is used to contain the geometrical information as well as interaction abilities in the form of interaction, drawing and culling call-backs. This type of object also incorporates collision detection and intersection calculations.

3.1 The Structure of RWB Applications

We have designed the functions of the RWB library to be clear, easy to use, and to minimize the programming effort made by a programmer of an RWB application. The user has to specify just the rwb-objects within the virtual world and to define the special functional/interactional call-backs of the application and the rwb-objects; more on this in section 3.2. The template for an RWB application looks as follows:

```
#include "rwblib.h"

class MySharedData: public rwbRootClass{
                        .. user specified .. };
MySharedData *MyShared;

static void My_UserCodeFunc()
  // this function is called from
  //  the Main_LOOP of the main process
 { .. user specified .. };

static void My_UserCodeAsyncFunc()
  // this function runs as a separate process
 { .. user specified .. };

static void My_KeyPressed_Event(int dev,int key)
 // this function runs in a user-input process
 { .. user specified .. };

static void my_reset_function(void)
 { .. user spec.; function called from the global reset};

static void reset_but(rwbObj *v)
 { .. user specified; button callback function };

static void exit_func(rwbObj *v)
 { .. user specified; button callback function };


int main (int argc, char *argv[])
{
 rwbInit_main(argc,argv);

  pfdInitConverter("myfile1.iv");
  pfdInitConverter("myfile2.obj");
 // Load all geometry-loaders before Performer forks

 rwbInit_scene();
 rwbInit_view();

 MyShared = new MySharedData();
 Shared->user_data=(void *)MyShared;

 Shared->UserKeyPressed_Event =
                   &My_KeyPressed_Event;
 Shared->UserCodeAsyncFunc     =
                   &My_UserCodeAsyncFunc;
 Shared->UserCodeFunc          =
                        &My_UserCodeFunc;

 rwbInit_sim();

 rwb_button *but1 = new rwb_button("exit",
          7.f,-18.5f,2.1f,10.f,3.f,4.f);
    but1->setPickFunc(exit_func);

 rwb_button *but3 = new rwb_button("reset",
          0.f,-18.5f,2.1f,10.f,3.f,4.f);
    but3->setPickFunc(reset_but);

 Make_ApplicationPanel("Test_RWB_Application",
                 "Copyright_(c)_2001_M.Koutek");

 MyShared->myObject = new myObject(Shared->my_world);
 MyShared->myObject->init();

 Shared->reset_func = &my_reset_function;
 my_reset_function(); // call it for the first time

 rwbHideGlobalCoordXYZ(); // or rwbShowGlobalCoordXYZ

 <... load or create geometries
       and build the scene graph ...>

 Scene Root::
 ------------> Shared->App_worldDCS

 Shared->App_worldDCS->addChild(<your pfObject:
                   pfDCS, pfGroup or pfNode>);
       or

     rwbObj *obj;
          // RWB class for anything you need
     obj = new rwbObj(0);
        // 0.. bounding box, 1.. sphere, 2.. cylinder
     obj->attachGRobj(<your performer-object:
                        pfDCS, pfGroup or pfNode>,
                xpos,ypos,zpos,xscale,yscale,zscale,
                rot_h,rot_p,rot_r);

     obj->add_to_SceneGraph(Shared->App_worldDCS);

     obj->DisableCollisions(); // or EnableCollisions
     obj->makeGenericBoundVol();
     obj->HideBoundingVolume(); // or ShowBoundingVol.
     obj->Update_TransfMat();

 rwbForkMain();
}
```

The detailed documentation of RWB library functions can be found at [8]. The most simple working application consists of:

rwbInit_main(arc,argv);
rwbInit_scene(); rwbInit_view(); rwbInit_sim();
rwbShowGlobalCoordXYZ(); rwbForkMain();

It creates an empty world with a grid texture on the ground and XYZ coordinate system in its origin.

3.2 3D Interaction and User Interface

The RWB library offers the user interaction with devices like a keyboard, a mouse, a space-mouse (with 6 degrees of freedom), and a stylus pen (6 DOF). The space-mouse is used for navigation in large environments and to its 9 keys extra functionality can be assigned by the application. For a real 3D interaction the RWB applications use the stylus pen with a tracking sensor and one button.

A widget set is available for building a 3D user interface with buttons, sliders, menus, display, dials and type-ins windows.

At the position of the stylus pen in the virtual environment a virtual cursor is displayed follows motions of the stylus.

3.2.1 Direct Object Manipulation Tools

When the stylus pen is inside an RWB object (colliding with it), the object is selected and changes its color to red.

The user can then invoke its function by clicking on the button, or holding the button and simultaneously performing some motion. There are 4 basic call-back functions of a generic RWB object:

- **touch/untouch:** an object is being touched
- **pick:** at the moment of the first button-click
- **manipulation:** during manipulation
- **release:** at the moment of releasing the button

Each of these can be user specified. For example touch call-back can print object information, or if the user picks a door it start an animation of opening the door.

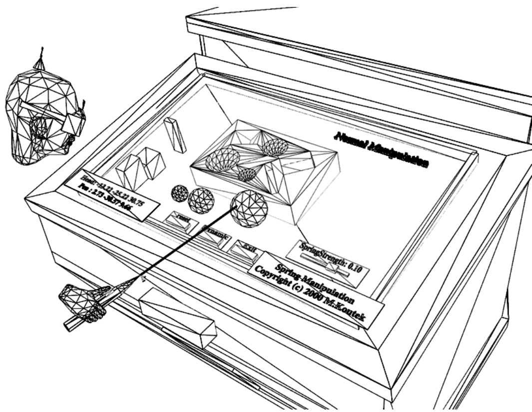

Figure 8: Object manipulation

Objects can be selected and manipulated directly with the stylus, or in the case of distant objects, using a ray-casting technique (see Figure 8).

3.2.2 Object Collisions

For a realistic object behavior in user interaction, collision detection is important to prevent objects from moving through each other.

Detecting object collisions helps to create an illusion that virtual objects have a substance.

We have implemented the following object collision schema into the RWB library: collision between stylus-object, ray-object and object-object.

The stylus and ray intersection/collision is supported by Iris Performer. Collisions between individual objects had to be implemented into our library.

First, bounding volume collisions between objects are evaluated. In the case of a collision of bounding boxes/spheres/cylinders the system performs a precise triangle object collision check. This part of the collision detection can put serious limitations on performance, especially if the triangle collision is not optimized and all triangles of one object are tested against triangles of the other object. We have implemented some optimalizations, but for a general object with more than 200 triangles, the interactivity decreases significantly during manipulation of an object colliding with an other object.

3.2.3 Dynamic Object Manipulation Tools

In the RWB library we have also built a visual force-feedback method to provide a visual interface and to substitute a real force input [4]. We use spring-based tools attached to objects assisting the manipulation, based on the following assumptions:

- a linear relation of force with spring compression / extension is intuitively understood and shown by the spiraling shape of a spring. Thus, even without exerting real force, a user has an intuitive notion of transforming a change of spring length to a force.
- bending and torsion of a shaft is used to show forces and torques exerted on virtual objects
- stability is introduced by friction and damping
- physical contact of objects is intuitively equivalent with geometric intersection

We have introduced a set of spring-based tools for providing the basic manipulation tasks, see Figure 9.

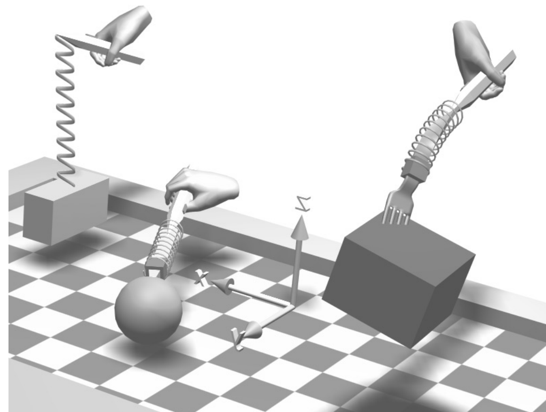

Figure 9: Spring-based manipulation tools

- **spring:** attached to the center of an object. It supports linear motions. The tool has 1 DOF (degree of freedom), the length of the spring, and controls 3 DOF (xyz) of an object.
- **spring-fork:** attached to an object it defines a contact point for transfer of forces and moments to the object. It supports translations and rotations. The tool has 3 DOF (extension, bend, torsion) and controls 6 DOF (xyz+hpr) of an object.

- **spring-probe:** used for probing the material stiffness of an object or pushing an object. The tool has 1 DOF (length) and can control 3 DOF (xyz) or 1 DOF (pressure) of an object.

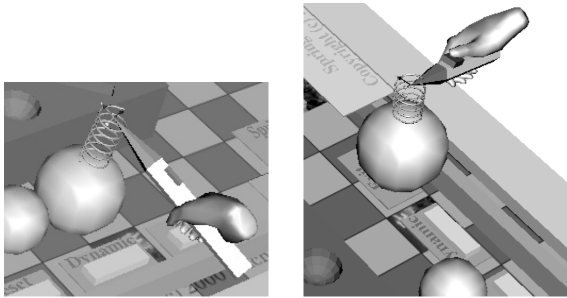

Figure 10: The Spring selection and manipulation

The Spring-tools are used as a link between user's hand and a manipulated object. When the user lifts a heavy object, the spring will extend proportionally to the object's weight and its motion.

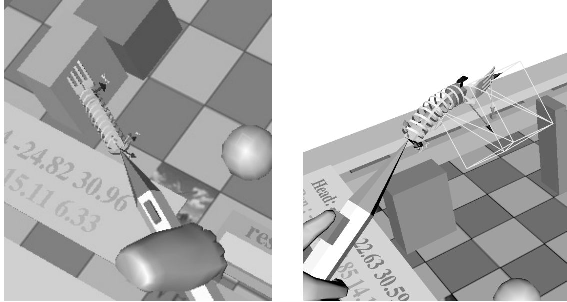

Figure 11: The Spring-fork selection / manipulation

The fork metaphor seems to be very intuitive. For object selection the fork has to be inserted into an object. The user can fix the position and the orientation of the fork inside the object. Then the spring part (handle) of the fork gives a visual dynamic feedback during the manipulation of the object. The user controls one end of the fork and the other end is influenced by the object. The fork can bend, extend (compress), or twist according to the laws of mechanics.

If virtual forces and moments are applied to virtual objects using the tools, they will show appropriate inertial effects according to the object's mass and moments of inertia.

3.3 Workbench In Workbench

Besides already mentioned functions and features of the RWB library including the collision and the intersection functions, the 3D user interface and the dynamic object manipulation tools, there are also some special functions and features of the RWB library.

For an improvement of the user's orientation in the virtual world projected on the RWB we have built in the library the Workbench In Workbench function, see Figures 12 and 13.

A small copy of the Workbench is projected onto the RWB table top. It contains the whole virtual world with all its objects as well as the user's head and the stylus. The user can orient in a large world by looking onto the Workbench miniature and seeing which part of the world is displayed on the RWB. The WIW function also helps to locate and manipulate objects

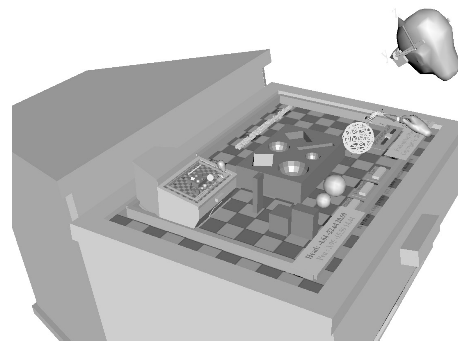

Figure 12: WIW: mini preview

which are not projected onto the RWB table top because they are out of the field of view. These objects are visible in the RWB miniature.

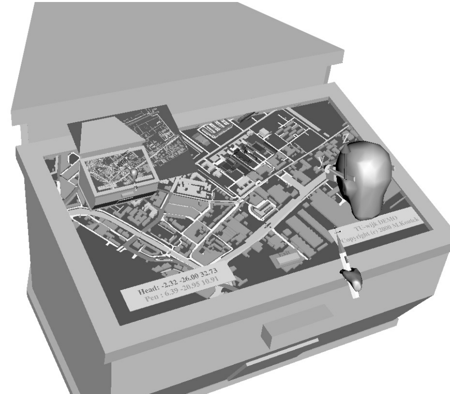

Figure 13: WIW: navigation assistence

3.4 Monitoring of User Interaction

A very important aspect of the work with the RWB is to be able to monitor and debug the RWB application in a distributed VR environment. Therefore we have implemented the monitoring function of the user interaction. The principle is quite simple. During the runtime of the application the tracker data are written into a file (or sent through a network) for immediate or later use, such as animated replay of a session.

Currently, we are using the file with the tracker data for the RWB Simulator.

4 The RWB Simulator

It is not always convenient and effective to debug or monitor an RWB application on the RWB itself. Sometimes the user performs application-specific tasks, and it is difficult to see if the task or the underlying algorithm works properly, when the user is just standing at the Workbench and wearing the shutter glasses. Usually, many program variables will be written onto the screen and analyzed, but on the real Workbench you cannot pause the application. This is even more complex if we consider the multiprocessing nature of the RWB application.

The Workbench Simulator is in fact the same RWB application that is compiled in a simulator mode.

It means that the tracker data are not read from the tracker daemon but from the tracker data file. The application then runs in the same way as on the real Workbench. What is different is the role of the user, and of course the type of the 3D projection.

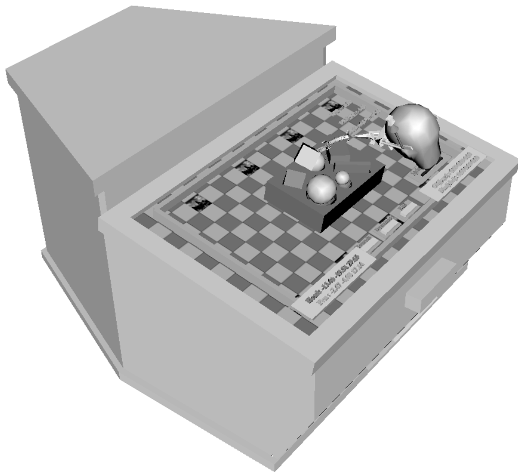

Figure 14: The RWB Simulator: fork manipulation

On the real Workbench the user performs the interaction with the RWB and with the running application. At any moment the user can start the writing of the tracker data to the file. Then within the RWB Simulator the user sitting at a common workstation can watch what was happening on the real RWB. The application world is displayed on the model of the Workbench. The user can observe the run of the RWB application, how the user performed with it and the simulator user can navigate around the RWB model with the mouse via a trackball metaphor (see Figure 15). The keyboard can be used to steer the simulator (e.g. pause, trace back/forward or reset simulation, reposition the user's head or the stylus).

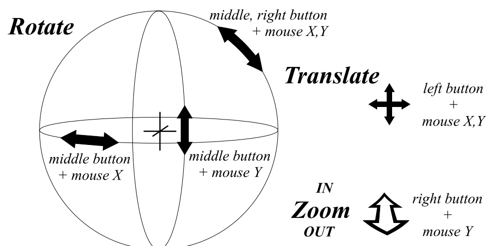

Figure 15: Using the mouse: trackball navigation

Another aspect of VR research is a demonstration & presentation of results. It is not possible to take stereo/immersive pictures of a user working with a RWB application. Usually, we switch the projection to a monoscopic mode and then we do "some adjustments" with the perspective to align the user with the virtual world. The RWB Simulator is very convenient for making pictures/animations of the RWB application, see Figures 10-17.

A big advantage of the RWB Simulator lies in its portability. The RWB applications can be implemented and developed on common graphic workstations with Iris Performer and the RWB library. Currently the library works exclusively on SGI workstations. With the availability of the Performer for Linux the RWB library will also be available for PC's.

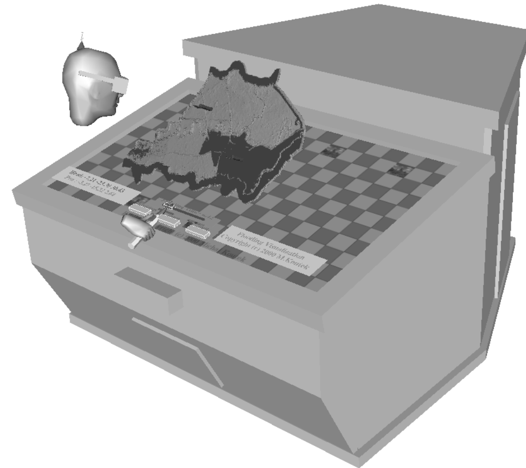

Figure 16: Delft WL|Hydraulics: visualization of the flooding simulation

Using our RWB library and simulator, the process of application development runs as follows.

First, the user prepares a main part of an application: the scene graph of the virtual world, basic functions and callbacks, the user interface. During this preparation stage the user compiles the application with the simulator option.

In the next step, the user runs the application on the real RWB, and performs some tests and adjustments. The user's can save sample tracker data for having some user's interaction data, and switches back to the simulator mode. This process repeats until the implementation is finished.

After the final test of the application on the real Workbench the RWB Simulator can produce images and animations for presentations.

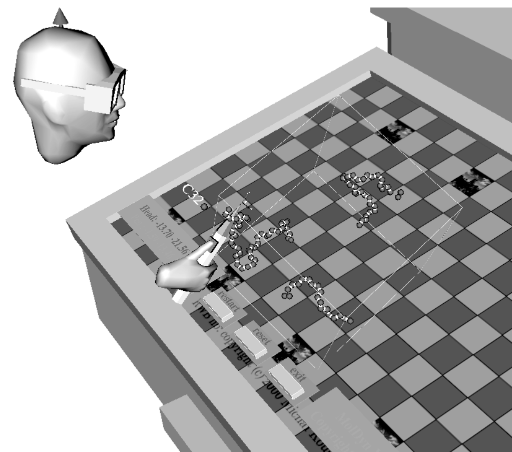

Figure 17: The RWB Simulator: molecular dynamics

5 Examples of RWB Applications

There is a wide range of applications running on the Responsive Workbench.

GIS, architectural, landscape planning/observation applications profit from the large overview, the high level of immersion and the 3D interaction.
In Figure 13, the Workbench In Workbench function assists the navigation in a GIS application. The user is observing a model of the TU-Delft campus.
In Figure 16, the user performs an interactive visualization of the flooding simulation.

The RWB, can also be used for various experimental applications such as shown in Figures 12 and 14, where the dynamic object manipulation with the spring-based tools is shown.

Visualization and simulation applications greatly benefit from the computational power of the SGI's Onyx 2 system. A large scalar / vector data set of a simulation can be interactively visualized on the RWB. Many visualization techniques can be used and combined together to produce the best visualization of given phenomena.

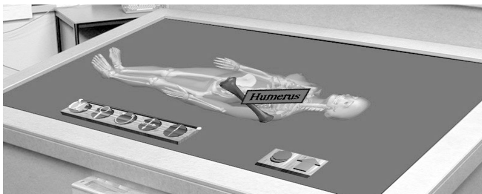

Figure 18: GMD: medical visualization

The RWB provides a convincing impression of a laboratory table application, for example in medical training or instruction on human anatomy, see Figure 18.

Figure 19: TN-HPaC: molecular dynamics

The Workbench environment is also suitable for scientific visualization and simulation. One example is molecular dynamics, see Figures 17 and 19.

The Figures 18 and 19, are real snapshots of applications running on the RWB and were not created using the RWB Simulator. The reader can compare the value of RWB Simulator images with the real ones. It's worth to mention that creating of the real images was a bit more complex task (adjusting light conditions, correcting the perspective, etc.). Making the simulator images was much simpler.

6 Conclusions and Future Work

We have implemented the RWB library which forms the basic implementation environment for the RWB applications. We have tested this system on several case studies. Some of them were mentioned in this paper.

This library, based on Iris Performer, offers not only the optimal usage of the available HW resources for the realtime 3D graphics and interaction but also includes several special features and extra functions which cannot be found (yet) in commercial packages like the CAVE library and the CAVE simulator [9].

Our system is still under development and increasing functionality. Currently, we prepare a version for PC's with Linux and by this make the RWB facility available for students and their VR assignments. We also plan to add an interface for *vtk* (Visualization Toolkit) [10].

References

[1] W. Krüger, B. Fröhlich, C.A. Bohn, H. Schüth, W. Strauss, G. Wesche, *The Responsive Workbench: A Virtual Work Environment*, IEEE Computer, July 1995, pp. 42-48.

[2] P. Dai, G. Eckel, M. Göbel, G. Wesche, *Virtual Space: VR Projection System Technologies and Applications*, Internal report on AVOCADO framework, GMD, 1997.

[3] C. Cruz-Neira, T.A. Sandin, R.V. de Fanti, *Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE*, Proc. of SIGGRAPH, 1993, pp. 135-142.

[4] M. Koutek, F. H. Post, *Dynamics in Interaction on the Responsive Workbench*, Proc. of Eurographics Virtual Environments 2000, Springer, Amsterdam 2000, pp. 43-54

[5] R. van de Pol, W. Ribarsky, L. Hodges, F. Post, *Interaction Techniques on the Virtual Workbench*, Proc. of Eurographics Virtual Environments '99 workshop, Springer, Vienna 1999.

[6] D. Bowman, L. Hodges, *User Interface Constrains for Immersive Virtual Environment Applications*, Proc. of IEEE VRAIS, 1997, pp. 35-38.

[7] S. Bryson, *Approaches to the Successful Design and Implementation of VR Applications*, ACM SIGGRAPH'94, Course Notes, 1994.

[8] *The RWB Library and the RWB Simulator*, http://www.cg.its.tudelft.nl/~michal/RWBlib

[9] *The CAVE Library and the CAVE Simulator*, http://www.ncsa.uiuc.edu/VR/VR/PapeClass/, http://www.ncsa.uiuc.edu/Vis/ImmersaVis/, http://avl.iu.edu/programming/cavelib2.5.6/

[10] *The Visualization Toolkit*, http://www.kitware.com/vtk.html