VDesktop: Event Management and Physically Based Behaviour in Tabletop Displays

Aitor Moreno[1], Jan Heukamp[1], Jorge Posada[1], Alejandro García-Alonso[2]

[1] VICOMTech
Paseo Mikeletegi, 57
20009, San Sebastian, Spain

{amoreno, jheukamp, jposada}
@vicomtech.org

[2] Euskal Herriko Unibertsitatea
Lardizabal, 1
20018, San Sebastian, Spain

alex.galonso@ehu.es

ABSTRACT

Applications for digital tabletops have some notable differences when compared to traditional desktop applications, the principal difference being that the user input method is not the traditional combination of mouse and keyboard.

This work addresses the difficulties and characteristics of event-handling management, when the applications are extended from 2D to 3D in the context of tabletop displays and where the interactions between the user and the represented 3D objects are more complex. This complexity increases when physically based behaviour of the objects is considered.

More specifically, the scope of this work is oriented to the support of physic-based simulation events in a tabletop display with an implementation of a system based on the OGRE graphical library and the ODE physical library, that are able to handle such elements.

Keywords

Tabletop display, Virtual Table, Simulation, Physics, Event Management, 3D widgets

1. INTRODUCTION

The digital tabletops displays allow the user to apply novel interaction paradigms, offering them a more usable and intuitive way to interact with a computerised system. One of the key characteristics of digital tabletops is that they provide an alternative to the traditional input mechanisms (mouse and keyboard). Instead, tabletop displays have a combination of input devices that translate the user's gestures into application input, the most widely found input methods are touch screens [Forl06] and gesture recognition using a tracking system.

Digital tabletop systems have some advantages compared to the traditional screen, keyboard and mouse combination. Firstly, the interaction is more natural and intuitive. Secondly, it provides a better social interaction with the system, since it allows multiple users to meet near the display, regardless of whether it is a horizontal or vertical tabletop. As a consequence, collaborative work can be enhanced using a digital tabletop.

A typical demonstration application that almost all the digital tabletop systems usually support is multimedia viewing and management (Figure 1), where the users can move, zoom and pan a set of photographs, maps or videos just by using their fingers as input devices.

Figure 1. PerceptivePixel wall [Han06]. A vertical digital tabletop.

On the other hand, there are other sets of applications that use physical simulation to give an even more realistic behaviour to the objects represented in the tabletop. The BumpTop Desktop [Agar06] improves

the desktop metaphor, extending it to the 3D world and adding physical behaviour, providing a more realistic representation of a real desktop.

However, there is a lack of a generic methodology to handle events from tabletop displays combining physically based behaviour and interaction with 3D objects.

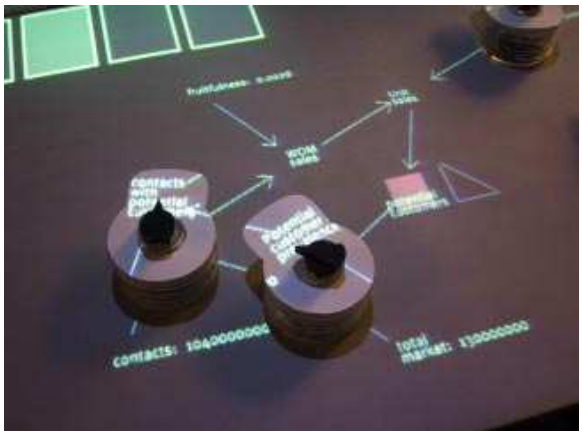

Figure 2. Sensetable [Patt01] tracked elements used as user interface controls.

In applications where this methodology is employed, the objects interactively process all the events generated by the users. Also, internal events will be triggered when collisions or other customised events occur. In 2D and classical desktop applications, all of this management is commonly addressed in a middleware layer between the applications, the users and I/O devices, and it is typically provided by GUI toolkits and frameworks tied to the host operating system.

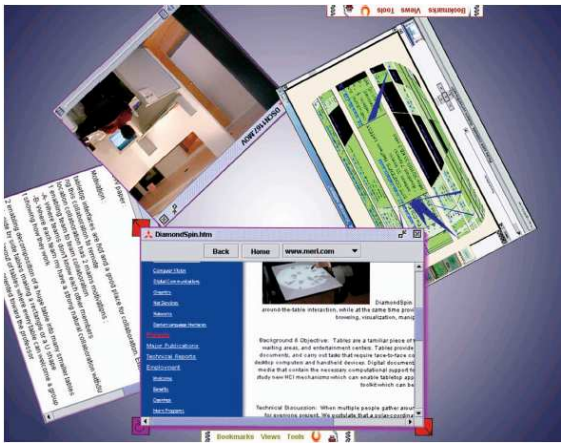

Figure 3. DiamondSpin [Shen04] TableTop Toolkit.

Following this approach, this work presents the definition and the characteristics of a middleware layer able to mediate between the user interactions in a desktop application and the underlying 3D environment, supporting physically based behaviour of 3D objects.

2. STATE OF THE ART

For a comprehensive study of modern tabletop system and methodologies, we refer the reader to [Stac06].

There are several works about tabletops displays, being the more relevant the following ones.

Sensetable [Patt01] (Figure 2) uses tracked physical objects as an input device. The system tracks the positions of multiple objects on a horizontal display surface. Each tracked objects is, in essence, a user interface element with a predefined behaviour.

DiamondSpin [Shen04] (Figure 3), is a library for virtual desk applications development that allows the gesture interaction between people allocated around the desktop and virtual objects.

Figure 4. BumpTop Desktop [Agar06] is a virtual and physic desktop with enhanced file organization.

Perceptive Pixel [Han06] (Figure 1) presents a large and vertical touch display where multiple users can interact with the included graphically and aesthetically rich applications.

The Reactable [Jor06] is a collaborative electronic music instrument with a multi-touch tabletop interface that uses tangible objects ad input controllers. As in the Sensetable, it uses different objects to represent synthesizers, and other electronic musical instruments.

Microsoft Surface [Surface07] is the Microsoft attempt to enter the digital tabletops market. It is essentially a Windows Vista ® PC with a 30 inches touch display and five cameras that track the object interaction. The graphical quality of the Microsoft Surface user interface is its chief strength.

The BumpTop Desktop [Agar06] (Figure 4) application offers a full interactive virtual 3D world

where all the traditional desktop documents are represented as 3D objects with physical behaviour.

From the application point of view, all these tabletop displays technologies are conceived as events triggered by *i)* the user, due to interaction with the display, *ii)* by the virtual objects, due to their own interactions or *iii)* by the system itself, e.g. timers. This separation of the real tabletop display hardware and applications is performed in a middleware layer that acts as an interface.

In the following section, the basic concepts to implement this generic event management are described. After a description of the 3D widgets and their organisation, the basic events and the geometric event management are presented.

3. EVENT MANAGEMENT

The event management connects system and user interactions. The user input events are interpreted and the system reacts accordingly with the implemented behaviour.

In a higher abstraction level, the application event management allows the objects to trigger customised events that the system will deliver to other objects that are listening or waiting for them. Normally, this kind of event management is implemented using observer patterns, where the objects register themselves as observers of the desired event classes.

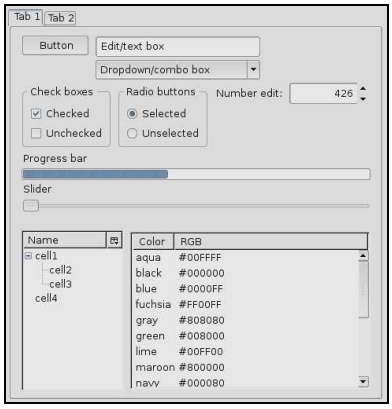

Figure 5. 2D GUI showing some common controls (Wikimedia Commons).

In 3D virtual worlds with physical behaviour, an extended object oriented event management is implemented to handle the user and object interactions, taking into account that it must provide real-time graphics rendering, real-time collision events and the traditional user interaction.

3D WIDGETS

The user interfaces are composed of widget elements whose main characteristics are that they may be drawn and they have specific behaviour. In Figure 5, a classical 2D user interface is shown. An extension of the traditional 2D desktop metaphor was introduced by the Compiz 3D and related projects (Figure 6), giving a sophisticated 3D management of the desktop.

In 3D environments, the 2D widgets of the user interface must be translated to the 3D world. Thus, a 3D widget is defined as an encapsulation of geometry and behaviour used to control or display information about application objects [Conn92].

WIDGET ORGANISATION

The widgets are organised in a hierarchical tree, where the internal elements act as containers. Each leaf widget has its own associated drawable geometry, collision geometry and the physical behaviour. However, the containers don't have any drawable geometry, their function is to allow tree traversal.

Figure 6. Compiz 3D on Ubuntu GNU/Linux (Wikimedia Commons).

The collision geometry of the containers is the composition of its child's geometry, and it is only recomputed when the widget sub-tree is modified. This geometry composition schema can be overridden by calculating a bounding box or sphere of all the contained children, which is a major efficiency improvement.

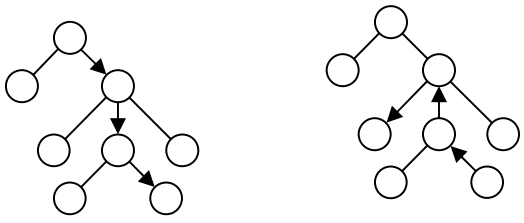

Figure 7. Left: The Root - Leaf communication as a tree traversal. Right: A widget – widget communication as a path.

The input events are sent by the root and they are usually propagated to the leaf widgets using a standard tree traversal (Figure 7, left). This could be called the standard algorithm for event management (Algorithm 1).

The geometric events are usually defined with a 3D position and a direction (mainly, a ray). As the widgets belong to a dynamic 3D world, their parents are responsible to pass the event data transformed to the local system reference of the widget (normally applying an affine transformation). These events must be passed only to those children whose bounding boxes collide with the ray.

Apart from the parent-child communication, it is useful the communication between any pair of tree elements reduces the amount of steps needed to traverse the tree between source and target widgets. The widget to widget communication is done by attaching a path to the event. This path defines a list of widgets that must be followed in order to reach the desired target widget, providing a direct communication between them (Figure 7, right). To calculate the path, an upwards tree traversal is performed until a common widget is found, which will be the pseudo-root of the path.

```
procedure processEvent ( event )
  foreach child in children do
    event.push (collision point);
    event.push (child transform);
    child.processEvent (event);
    event.pop ();
    event.pop ();
  ...
  otherProcessing ();
end;
```

Algorithm 1. The process event pseudo-algorithm is a recursive function that traverses the tree.

It is possible to stop the propagation of any event. When a widget does not want to propagate an event, it consumes that event, and thus, the event will not be propagated toward the target widget.

The widgets must register the events that they want to receive. If any given widget children are not registered to a specific event, the event will not be propagated to the children. This is a major improvement, since the calculation of collisions and bounding box checks are not trivial, and it is better to avoid it if they are not necessary. It would be inefficient to generate thousands of events per seconds, when only a few of the widgets are interested in those events.

The parent widget must have the control over the events that their children receive. In this way, they can filter or modify some of the event if required.

BASIC EVENTS

Although an application can define its own customised events, a common set of necessary events has been defined:

Input Event: The input events are those that are triggered by the users when they interact with the system. With high similarities with the 2D world, the input events are classified in *MoveEvent*, *PressedEvent*, *DragEvent* and *ReleaseEvent*, as a clear reference to the standard mouse events in a 2D GUI.

The ***MoveEvent*** events are generated each time the device pointers are moved over an object.

The ***PressedEvent***, ***DragEvent*** and ***ReleaseEvent*** are generated each time the user clicks, drags or releases the pointer device over an object.

For example, almost all the widgets react to the *DragEvent* modifying their size. Depending on the drag direction, the size of the objects is increased or decreased. If a multi-touch device is used, this functionality can be used to allow the user to resize an object by placing fingers in two corners of the object, sending two *DragEvent* events to the target object with a distinguishable identifier. The object will modify its own size using the information provided by both events.

Collision Events: Collision Events are created by a physical world when two objects collide. The event is sent to both objects, and they usually react to the collision in a physically based behaviour. One of the major problems is that a high number of collision events are generated by the physics engine in every step of the simulation, which leads to efficiency and performance problems that will be addressed in the next topic.

SystemEvents: The system can trigger events when a condition occurs, the most common system events are command actions, time dependant events and I/O instructions (read, write).

The *VariableStepEvent* is triggered each time the graphics engine renders a frame in the virtual scene. The widgets registered to the class of event have and opportunity to change elements within the scene, such as animations or post-rendering calculations. This event provides the objects with the elapsed time since the last VariableStepEvent event.

The *ConstantStepEvent* is a generalization of a timer function, and it used to trigger events periodically. The physics world uses a *ConstantStepEvent* to update the physics simulation, since typically physics engines have fixed step periodicity. In addition to the physics engine, the objects may receive the same

event to update some physical values, such as acceleration, force, friction, etc.

The *ActionEvent* notifies that an action has occurred. For example, when a button is pressed it will trigger an *ActionEvent*s event. These kinds of events do not traverse the complete tree. To receive the action events, the widgets must register themselves as observers of the desired action events. For example, when a pause button is pressed, the video is paused or resumed, depending on its internal state.

COLLISION EVENTS MANAGEMENT

Geometric events are generated when the physics engine detects a collision between two objects. It does not matter whether the collision is between two free objects (whose movements are calculated by the physics engine), between two user controlled objects, or any other combination, as a user interaction is reduced to a directed ray from the 2D tabletop display to the 3D virtual world.

In all cases, the geometric events are described by

i) the involved objects identifiers,

ii) the collision points in both objects and

iii) the vectors of the collision.

In a single simulation step, the physics engine updated some physical data for each object, like accelerations, speed, orientation, position, and others. When the new positions of the objects are calculated, potential collisions occur. Depending on the number of objects and their geometric complexity, the number of collisions can be difficult to handle interactively. In order to reduce the amount of collision information, a filtering algorithm has been introduced (Figure 8).

Firstly, when a collision occurs, it is checked whether one of the collided objects wants to receive the collision. If neither of the objects wishes to receive the collision, the event is dropped. All the pure graphical and static objects should have this behaviour, since it will lead to better overall performance.

In the second stage, if another collision event has been previously triggered by both of the two objects, the event is dropped. This simplification is needed, since a collision is resolved by the physics engine during a period of time, not instantaneously. The collision events are prolonged in several simulation steps; this generates a large amount of events and consequently becomes difficult to handle in real time.

Each time two new objects collide and generate a collision event, their ID's are stored as a pair in an ordered list. Each time a collision is received, the ID's of the collided object are searched in the list. If they are found, the elapsed time is checked and the collision event is dropped if the time is less than a fixed time value (e.g., 200 ms). If the time is greater than the given threshold, the ID pair is removed from the list and the collision event passes. This stage is conceived as time filtering, allowing only collision events pass at a fixed rate (See Table 1).

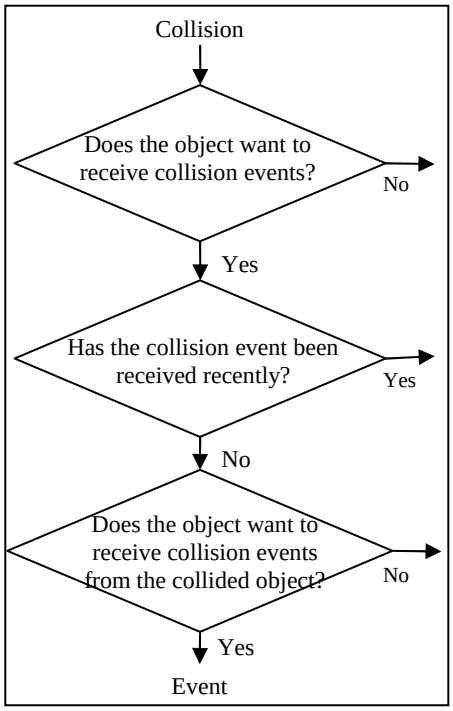

Figure 8. The diagram shows the Collision Event filtering algorithm.

In a third stage, one of the target widget filtering callback function is called, passing the reference of the other target widget as parameter. This filtering stage consists in determining if the collision between both widgets will succeed or not.

Filtering Time (s)	Max widget	Subjective Impressions
0.1	250	Interactive
0.2	140	Interactive
0.4	70	Bumpy
0.8	40	Bad

Table 1. For some different values, the table shows the number of widgets that the system can handle before the simulation becomes very slow and the subjective impressions with 100 widgets.

A scenario where this filtering is effective is when we have several widgets composed by low level and smaller widgets. When one of these smaller widgets collides against a high level one, it is better to drop the collision and let the high level widget react to the collision, since it will likely have an optimized

physical geometry. In this example, several complex collisions are dropped and only a single and simpler collision is calculated by the physics engine.

As the filtering algorithm will be executed several times per second, it should be implemented in the most efficient way. It is preferable to implement a fast but permissive filtering algorithm rather than an intensive and accurate one.

Combining these three stages, a high performance collision events handler is plausible, with little impact on the overall system performance. The time filtering interval is the only configuration variable and its effects are noticeable on the system reaction time. With higher values, the system tends to react very slowly to collisions and this can cause side effects such as disturbing the physics engine calculation in the following iterations.

Figure 9. Low Level widgets. From left to right and top to bottom: ImageNode, VideoNode, RenderNode and ButtonNode.

4. IMPLEMENTATION

The implementation of the proposed event-handling middleware has been done through the implementation of a Virtual 3D Desktop application, without a strict validation of what are the benefits of the desktop metaphor itself [Trist01].

The VDesktop contains a top view of a virtual, 3D and visually attractive desktop metaphor, where the user documents are represented as physical widgets following fundamental physical laws, i.e. gravity, collision and friction.

IMPLEMENTED WIDGETS

The implemented widgets are divided into high level widgets (Figure 10), akin to the users understanding of document, and low level widgets (Figure 9), that are used to compose the high-level ones. These are the low level widgets:

The **GraphicalNode** represents a simple drawable and physical object. The graphics engine will render the 3D geometry into the graphics card's frame buffer and the physics engine will use the physical geometry in the physics simulation. Both geometries could be different, for example, a highly detailed 3D model using its own bounding box as collision geometry. All of the following widgets inherit this behaviour, and therefore omitted from their descriptions.

Figure 10. High Level widgets. From left to right and top to bottom: ImageDocument, VideoDocument, MeshViewer, FolderContainer, RecycleBin and the VDesktop.

The **ImageNode** represents an image as a physical box with the image itself as the texture of the surface of box. It has internal functionality to access the image information, including the width, height and the pixels of the image.

The **VideoNode** represents a video as a physical box with the first video frame as the box surface texture. It has internal functions to access the video playback functions, i.e., play, stop and pause.

The **RenderNode** represents a 3D world as a physical box. The internal virtual world is rendered to a texture that is set as the top surface of a physical box. This widget has internal functions to manage several aspects such as the camera position.

The **ButtonNode** is the translation to the 3D world of the 2D classical button present in the 2D GUI toolkits. When it is activated or pushed, a customised *ActionEvent* is triggered, enabling the registered objects to react to the button action.

The high level widgets share some common functionality. For example, when they are resized down to a specified threshold, they are represented as a simpler box with a proper texture on its top surface. When they are scaled up, the following behaviour is obtained:

The **ImageDocument** represents a very simple image editor. It is composed by an *ImageNode* with some *ButtonNode*'s offering an editing functionality.

The **VideoDocument** contains a *VideoNode* with some *ButtonNode*'s to control the video playback functions.

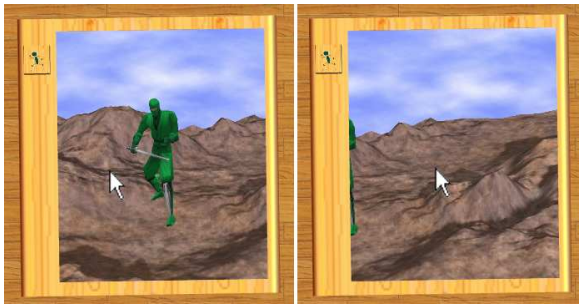

Figure 11. The input events are passed to the internal RenderNode of the MeshViewer

The **MeshViewer** represents a simple 3D model viewer. The widget has a menu with a *ButtonNode* for each animation stored in the 3D model. When such buttons are pressed the selected animation is loaded in the model and visualised. The internal camera is moved passing input events to the internal *RenderNode* widget (Figure 11).

Figure 12. Desktop application showing various *ImageDocument, VideoDocument* and *RecycleBin* widgets. The textured marbles are *GraphicalNode*'s with no other major functionality.

The **FolderContainer** is the 3D representation of a folder, which acts as a documents container that can be extracted or inserted.

The **Recycle Bin** represents the traditional place where deleted documents go. Using the physics system, the user can literally throw the document to the trash. As it has depth, thrown objects can be recovered back by simply dragging them.

VDESKTOP APPLICATION

The VDesktop itself is a widget composed of several other widgets, and defines the common physical rules that will be used by the physics engine (mainly gravity and friction).

The application is composed of a single VDesktop widget and an eagle eye camera correctly positioned to give the users a correct perspective of the widgets (Figure 12).

The OGRE high performance graphical library [OGRE07] has been used as the graphics engine, achieving real time rendering (for OpenGL and DirectX) having a GeForce 5200 FX graphics card with 32 Mb, installed on a AMD XP 2600 MHz CPU with 512 Mb RAM, which is a common basic PC specification.

The ODE [ODE07] physics engine has been used for the physics simulation, using the OgreODE wrapper [OGREODE07] to integrate the ODE functionality in OGRE projects.

5. RESULTS AND CONCLUSIONS

This work presented a generic methodology to handle user input events in tabletop displays showing 3D virtual world with physical behaviour.

The presented event management has been optimised to provide interactive visualisation when the object interaction generates a large quantity of physical collisions.

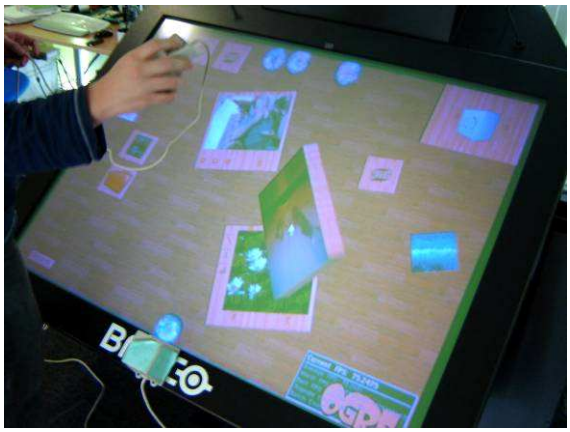

Figure 13. VDesktop demonstration shown in a BARCO display. The spatial 3D mouse is used to interact with the virtual world.

The implemented VDesktop prototype is focused in the validation of the proposed event management methodology, providing basic desktop metaphor elements adapted to the digital tabletop display paradigm. In this case, the implemented widgets are related to the multimedia domain, thus, images, videos and 3D objects are the principal document types the prototype can handle. The folder and the

recycling bin are other common elements in desktop that have been also implemented.

Finally, the VDesktop application has been tested on a BARCO Baron model [BARCO07] (Figure 13) using a magnetic tracking mouse for user input.

6. FUTURE WORK

The next steps should be focussed on the integration of hand gesture recognition in the system, which will provide users with a more natural interaction.

From the point of view of the applications, the presented event management provides a very suitable framework as a start point to the development of customised applications, such as kiosk information points, travel information access, museum story telling, etc.

In any case, tabletop display oriented applications should be involve the end user in design and development to ensure the final user experience is natural and intuitive.

7. ACKNOWLEDGMENTS

Dr. García-Alonso was supported by the Spanish Ministry of Education and Science, grant TIN2006-14968-C02-01.

REFERENCES

[Agar06] Agarawala, A., Balakrishnan, R. Keepin' it Real: Pushing the Desktop Metaphor with Physics, Piles and the Pen. Proceedings of CHI 2006 - the ACM Conference on Human Factors in Computing Systems. p. 1283-1292.

[BARCO07] BARCO Displays Website. Baron Product technical specifications. Last Visited on 2007-10-23: http://www.barco.com/corporate/en/products/product.asp?GenNr=324

[Conn92] Conner, D.B., Snibbe, S.S., Herndon, K.P. et al. Three Dimensional Widgets. Proceedings of the 1992 Symposium on Interactive 3D Graphics, Special Issue of Computer Graphics, Vol. 26

[Han06] Han, J., Perceptive Pixel Site. Last visited on 2007-10-23. http://www.perceptivepixel.com

[Forl06] Forlines, C., Shen, C. and Vernier, F. Under My Finger: Human Factors in Pushing and Rotating Documents Across the Table. IFIP TC13 International Conference on Human-Computer Interaction (INTERACT), September 2005

[ODE07] Open Dynamics Engine Website. Last Visited on 2007-10-23. http://www.ode.org

[OGRE07] OGRE Graphics Engine Website. Last Visited on 2007-10-23. http://www.ogre3d.org

[OGREODE07] ODE wrapper for the OGRE. Wiki, last Visited on 2007-10-23. http://www.ogre3d.org/wiki/index.php/OgreODE

[Patt01] Patten, J., Ishii, H., Hines, J. and Pangaro, G. Sensetable: A Wireless Object Tracking Platform for Tangible User Interfaces, in Proceedings of Conference on Human Factors in Computing Systems (CHI '01), March 31 - April 5, 2001). pp. 253-260

[Stac06] Scott, S.S. Carpendale, S. Interacting with Digital Tabletops, IEEE Computer Graphics and Applications, September/ October 2006, Volume 26; Issue 5.

[Shen04] Shen, C., Vernier, F.D., Forlines, C. abd Ringel, M. DiamondSpin: An Extensible Toolkit for Around-the-Table Interaction. ACM Conference on Human Factors in Computing Systems (CHI), ISBN: 1-58113-702-8, pp. 167-174, April 2004

[Jor06] Jordà, S., Geiger, G., Alonso, M., Kaltenbrunner, M. The ReacTable: Exploring the Synergy between Live Music Performance and Tabletop Tangible Interfaces. Proceedings of the first international conference on "Tangible and Embedded Interaction" (TEI07). Baton Rouge, Louisiana.

[Surface07] Microsoft Surface Site. Last Visited on 2007-10-23. http://www.microsoft.com/surface

[Trist01] Tristram, C. The next computer interface. MIT Technology Review, December 2001.