A Prototype Hotel Browsing System Using Java3D

D. Ball and M. Mirmehdi
Department of Computer Science
University of Bristol
Bristol BS8 1UB, England

Abstract

Java3D is an application-centred approach to building 3D worlds. We use Java3D and VRML to design a prototype WWW-based 3D Hotel Browsing system. A Java3D scene graph viewer was implemented to interactively explore objects in a virtual universe using models generated by a commercial computer graphics suite and imported using a VRML file loader. A special collision prevention mechansim is also devised. This case study is reported here by reviewing the current aspects of the prototype system.

1. Introduction

A number of hotel companies, travel agents and other organisations currently have sites on the World Wide Web (WWW) where guests can get information about hotel accommodation. The aim of this project is to specify, design and implement a WWW application which will allow guests visiting such web sites to download 3D models of a hotel and take a virtual tour of the facilities available. For example, this may involve "walking" around a room or the sports centre of the establishment.

The number of hotels marketing themselves on the WWW is constantly increasing. A 3D hotel browser is one way in which a hotel company can stand out from its competitors on the Internet, and benefit its users at the same time. The current WWW hotel services exploit some advantages of the Internet, but provide little more information than a conventional printed brochure. This will only hold the users' attention for a short while. If, however, the user could interact with the WWW site by getting inside the hotel and walking around, then they are likely to be more interested, stay longer at the site, and remember what they have seen during their interesting (and fun) virtual tour.

There are many other immediate benefits of this system. It could be adapted for use by Estate Agents to present houses for sale or rent, or to allow prospective students to view University campus accommodation. In fact, the prototype system is a generic system and is called the Virtual Tour System (VTS), and while it is tailored towards hotel browsing, it is easily adaptable for other virtual tours.

Section 2 considers the tools used in this project. The prototype system is described in Section 3 along with some example images of the current state of development. Section 8 discusses the tasks remaining to complete the project in the near future.

2. The Tools

In this section we briefly review some (of the more important) software tools used in this project.

The Java3D Application Programmers Interface (API) [1] has been specified as a new extension to the core Java language which will support 3D computer graphics. Although this API has been finally specified, a full implementation is not yet available. The Computer Science Department at the University of Bristol was provided with a pre-alpha release of Java3D which has been used for this project. Some of the features currently supported by Java3D are: perspective or parallel projection, solid or wireframe rendering, flat shading engine of polygons, and multiple viewports (multiple views of the same scene) each with its own set of parameters, and full 3D frustum and screen level clipping.

Alias Wavefront [2] is a commercial graphics application which can be used for rendering photo-realistic 3D models. These models can be exported in a number of file formats which can then be loaded and used by other 3D viewers.

VRML (Virtual Reality Modelling Language) [3, 4] can be used to enable WWW browsers to download and display 3D models or virtual worlds. Models developed using Alias Wavefront are exported using the VRML file format.

It may well be possible to use other technologies, however we used the resources immediately available to us, and we believe it is the first time that VRML files are used in Java3D. In fact, VRML and Java3D are an ideal combination for this project as the Java3D scene graph matches very closely with the VRML structure.

3. System Overview

Naturally, robust design and performance issues are applicable here as in any software project. Also, human factors and general user requirements are imperative necessities. These were considered through liaison with the external industrial partner. The basic architecture of the prototype system involves three major components:

- Hotel Database - the hotel database contains all the information required for the hotel browser to generate virtual tours of featured hotels. This content comprises 3D model data and general hotel information. The database is located on a single Web server host. The database may contain references to external information sources on the same site or at other locations on the Internet.
- Hotel Browser - the hotel browser is a Web-based application which accesses information from the database via the WWW. It operates within a conventional Web browser, integrating the virtual tour capability with existing WWW hotel information services.
- Database Editor - the database editor provides a fully featured tool to create and edit all sections of the database. Through a graphical user interface it allows a skilled operator to build and modify hotel models and configure other database information including selection, insertion, deletion, and parameter editing. We will not consider the Editor any further in this paper.

The system will consist of a single database and any number of hotel browsers. A hotel browser will interact with its user through a Java3D interface running in a WWW browser. Figure 1 illustrates the three main components in the prototype system.

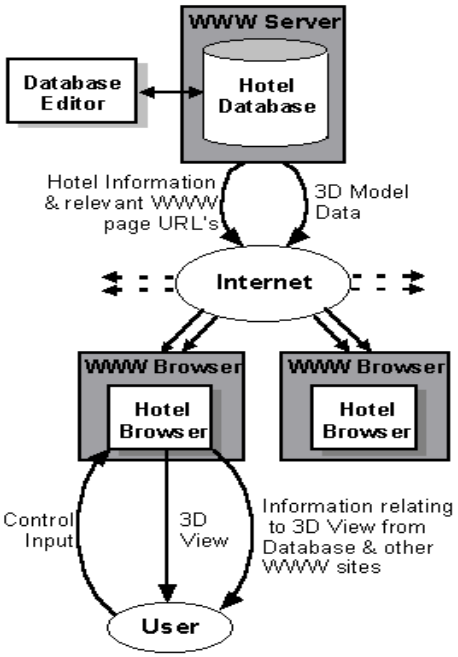

Figure 1. An overview of the Hotel Browser System.

4. Hotel Database

The database provides a description of a closed hotel *universe* (Figure 2). It consists of a hierarchy of hotel description objects. There are four types of hotel objects: Hotel Group, Hotel, Room, and Feature. Each object has a number of properties and the following are common amongst them: *Name*, *Description*, and *References* (to related WWW Resources).

A Hotel Group object collects together one or more Hotel objects and has the aforementioned properties only. A Hotel Object collects together one or more modelled rooms from the same hotel. It has the following additional properties: *View Parameters* which should be defined to give the user the best view of the room models belonging to the hotel, and *Avatar Parameters* which should describe the size of the avatar such that it is in proportion with the room models belonging to the hotel.

A Room object defines data to generate a 3D model of a hotel room or facility and groups together one or more features which are contained within that room. Additional properties include *3D Model Description Data* which can be used to construct a model for the user to view, *Location* to determine the physical position of the room within the hotel, *Room Bounding Region* which describes the physical volume in space occupied by the room, and *View Position* which defines a viewpoint within the room at which the virtual tour should start (this might typically be set at the doorway into the room). Figure 4 shows the definition of a typical double bedroom (specified using the Database Editor interface).

A Feature object defines a 3D model of a physical feature within a hotel room, such as a bed or a lamp-shade. The additional properties are similar to a Room object and provide 3D model description of the feature, its physical location in the room, and the preferred viewpoint for showing the feature to the user. Typically, moving the user to the view position will generate a 'close-up' of that feature.

The system user will submit control input through a user interface displayed in their WWW browser. These controls will enable the user to visualise the information stored in the hotel database. Figure 1 also provides a schematic view

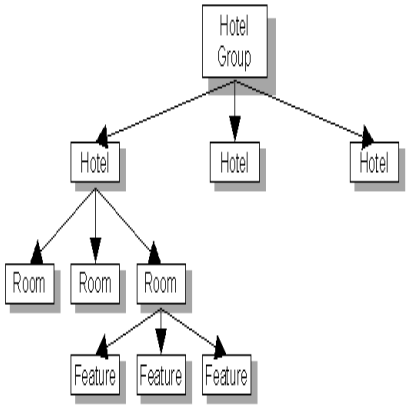

Figure 2. Hotel universe structure.

of the flow of data in the 3D Hotel Browser system.

5. Our Use of Java3D

This section briefly reviews some of those aspects of Java3D used in this application which are regularly referred to in the next sections of this paper. Central to Java3D, is the scene graph method for describing the content and behaviour of objects in a 3D universe. The scene-graph approach to describing a 3D universe is not new, or unique to Java3D. It has been used in various forms, perhaps most notably in VRML [3, 4]. A scene graph in Java3D is a directed acyclic graph [5]. The basic idea is that each object in the feature of the universe is described by a node, or small collection of nodes in the graph, sometimes called a group. The exact state of a feature (i.e. its location, orientation, size, etc.) is determined by the nodes traversed in the path from the root of the scene graph to the feature's node (or nodes). We use the Java3D scene graph for various purposes one of which is as part of a collision detection mechanism as described later below.

Java3D introduces a new View model which aims to ensure compatibility of applications across a wide range of visual hardware from simple desktop monitor screens to full virtual reality head-mounted displays. Conventional 3D APIs such as OpenGL utilise a camera-based view model where the programmer has ultimate control and must implement the exact position, orientation and other view properties in order to render a scene. Java3D by contrast separates the virtual property of the viewer's position in the Virtual Universe from physical display properties such as the number of screens, field of view or orientation of the viewer's head (for a head-mounted display). This is achieved within the context of a scene graph. A View Platform leaf node is provided which can be added into the scene graph, perhaps below a Transform Group node to allow its position and orientation to be altered by the application. Physical information about the user and the display hardware are then accessed through special information objects. For more details on Java3D the reader is referred to [1].

6. Hotel Browser Application

The hotel browser consists of the View Display, Avatar Control Panel, Hotel List, Room List and Feature Lists components. It starts automatically when the user selects a link on the appropriate Web page. The application appears within the original Web browser window on the user's computer and loads a specific Hotel Group from the database according to the exact link selected by the user. The user will be given clear instructions at all times to assist them in using the browser.

The functions supported by the browser are the selection of a Hotel, selection of a Room, navigation between rooms in the View Display part of the browser, selection of a Feature for detailed viewing, etc.

One novel idea is to allow the user to select the type of features he/she requires (from the Features List) in order to define the sort of room they would like to stay in at that hotel. This means the room will be custom designed by the user and will be prepared by the hotel management in time for the user's arrival. Better still, this idea could be applied when users wish to specially arrange a conference or seminar room in the hotel.

6.1. Tour Simulation Behaviour

The user should be able to explore the hotel models in a manner similar to walking around a real hotel. This section briefly specifies the behaviour of the system.

The parameters used to specify the view rendered from a specific viewpoint in the hotel universe are the standard Graphics viewpoint geometry parameters: the user Field Of View Angle, the Front Plane, defined by a distance from the view point beyond which objects become visible, and the Back Plane, defined by a distance from the view point beyond which objects become invisible. Only objects in the visible region will be rendered and seen by the user.

The avatar description is used to determine how the user interacts with the scene. It comprises height, depth and width dimensions defining a bounding box approximation of a human body, and gravity simulation. This bounding box will be used to determine collisions between the avatar and objects in the universe. A step height parameter is used to determine the maximum height of objects above the base of the bounding box which the avatar can step up onto. The avatar and view parameters will be set on a per-hotel basis

according to the values specified for the current Hotel object which describes the hotel being viewed.

Java3D supports the animation of 3D objects and user interaction via the Behaviour leaf node. Behaviour nodes can be used to capture keyboard or mouse input from the user and manipulate (for example) the position of the View Platform. Java3D provides various features which can be combined with mouse events to support the picking of objects in a 3D scene. Other uses of Behaviour nodes include continually modifying a Transform Group node to animate part of a scene graph, altering the intensity of a light or starting a sound node. A basic approach to user interaction support through the Behaviour leaf node might be to implement a Behaviour node which has a reference to the View Platform Transform Group. This Behaviour would specify, for example, a key press as its initial wake up condition. When the user presses a key an event is generated which the Java3D scheduler passes to the Behaviour node to notify it that the wake up condition has been met. The Behaviour node can examine the event generated by the key press and alter the Transform Group matrix based on which key the user actually selected. To refine this solution further the Behaviour node might alter its wake up condition after a key has been pressed to be the disjunction of a key released event and the elapsing of a single frame. The next time the scheduler awakes the Behaviour node, it can examine the wake up condition satisfied. If a frame has elapsed then the Behaviour would repeat the last move made (still based on the key which the user last pressed). If, alternatively, a key released event was generated by the user then the Behaviour node ceases to alter the View Platform Transform Group, and return its wake up condition to waiting for the next key press. With this approach the user can hold down a key to instruct the Behaviour node to repeat the same operation, for example 'step forward', several times.

When this system of control was implemented in a test prototype two problems were highlighted. Firstly, binary (on/off) type inputs such as buttons and key presses were not found to be user friendly when trying to navigate through a 3D scene. This was due to the need in some cases to travel quickly, in large steps, and at other times to travel slowly. The naive control logic described above offers the user only one speed of action. Secondly, and more fundamentally, a property of the Behaviour scheduler came to light which is undesirable in this type of control system. When rendering more complex models a tendency emerged for the scheduler to 'lose' Java Advanced Window Toolkit (AWT) events. The result was that key released events would not be received by the Behaviour node and motion of the View Platform would therefore continue after user actually requested it to cease.

A solution was devised for this project which combines the use of the Java3D scheduler, a Behaviour node and the Java1.1 AWT event model used in 'conventional' Java window applications. User input events are detected using the implementations of the Java AWT event listener interfaces and custom developed user interface components. When the user activates one of these components a Behaviour node is notified directly by the component. This bypasses the Java3D scheduler as the Behaviour no longer needs to specify AWT events in its wake up conditions. When a component is activated the notified Behaviour node sets its wake up condition to be one elapsed frame. The Java3D scheduler will then wake the Behaviour every frame allowing it to move the View Platform according to the user's exact input.

6.2. View Platform Collision Prevention

A fundamental issue in interactive applications such as computer animation and virtual environments is collision detection [6, 7]. When navigating a virtual scene, it is reasonable for the user to expect that the View Platform cannot pass through a solid object. Java3D offers fairly comprehensive high-level support for collision detection in which each object may define a region known as its collision bounds. These collision regions may be described as simple box or sphere objects, or more complicated regions comprising a combination of simple objects. According to the Java3D specification, the Java3D renderer runs an infinite loop consisting of the following operations:

```
while(true) {

 1. Process Input
 2. If (Request To Exit) break
 3. Perform Behaviours
 4. Traverse Scene Graph and Render
    Visible Objects

}
```

What this brief description does not make clear is exactly where the occurrence of collisions is calculated. Various test applications written for the project suggest that the collision calculations are performed as part of step 1. This was deduced by the fact that if a Behaviour modifies part of the scene graph such that a collision between two nodes occurs, it is does not result in the interaction of any collision related Behaviour nodes until the current frame has been rendered. In practice this means that all collisions are rendered to the screen before they are reported. This property is not unreasonable given that if the actions of one Behaviour node were able to stimulate another in the same frame a dead-lock situation could occur where competing Behaviours prevent the renderer ever completing a frame.

Hence, since in Java3D all collisions are rendered to the screen before they are reported, we have designed a "1-step

ahead" collision detection scheme to work alongside the Java3D mechanism. In the context of this application, we refer to this mechanism as Collision Prevention (rather than Collision Detection). Since the requirement is to give advanced warning that the viewer is *about* to intersect an object, collision prevention is a more appropriate action which implicitly deals with collision detection.

It is not possible in Java3D to specify visible geometry or collision bounds for a View Platform node. To enable proper collision prevention, a separate geometric representation of the View Platform must be added as a separate node to the View Platform's Transform Group (Figure 3). Collisions are then detected between this extra node, which will retain a constant position with respect to the View Platform. The "1-step ahead" scheme developed extends this strategy further by separating the collision bounded object completely from the View Platform, as shown in Figure 3.

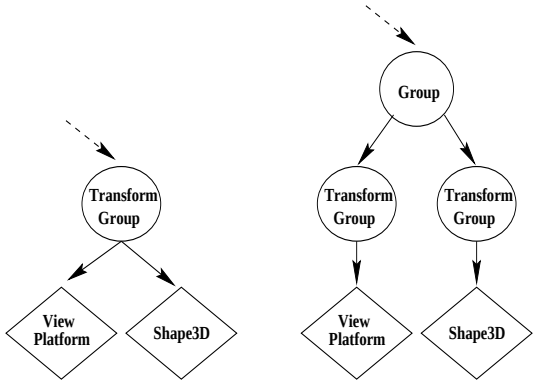

Figure 3. (a) Conventional scene graph, (b) Separating the transform of a View Platform and its associated geometry and collision bounds.

The View Behaviour node described in the previous section is extended to control the two Transform Group nodes now used. When user input generates a movement of the View Platform, the new transformation is first applied to the Transform Group of the collision bounded node while the View Platform transformation remains unchanged. The current frame is rendered and Java3D performs the collision calculations. If a collision is detected for the collision bounded node, then its transform is returned to that of the View Platform. However, if no collision occurs, the View Platform transform is set equal to that of the collision bounded node. In the latter case when more user input is received, the collision bounded node is moved on to the next transformation.

The result of this system is that movement of the View Platform lags one frame behind that of its associated collision bounds. If a collision occurs for a specific transformation the View Platform is prevented from moving to that position. Thus, Collision Prevention is achieved at the cost of a one frame delay in the response of the View Platform to the users commands.

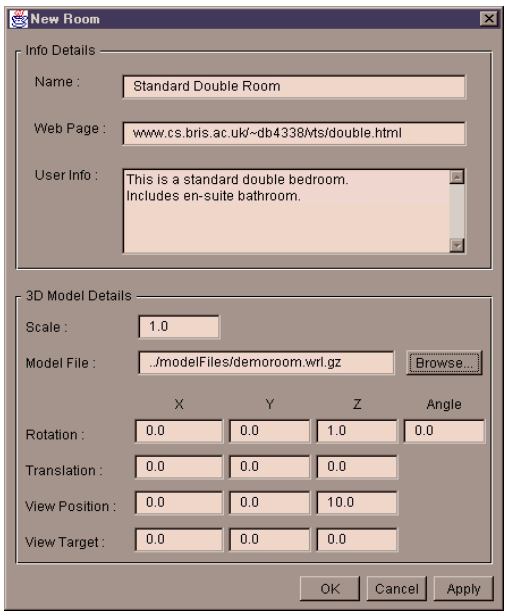

Figure 4. Definition of a new Room object.

Finally, in Figure 5 we show a snapshot of a hotel tour inside a room. The use of Java3D has enabled the key sections of the system to be independent of any 3D file format. It can be extended easily to support other file formats. The Hotel Browser is a good example of how simple HTML hypertext links can be used within an intuitive and interactive application user interface. In this case the distributed nature of the information on the internet is "hidden" and referenced from different features in a single hotel model.

7. Performance

At this stage in the development of Java3D much has been said about its performance with most observers claiming that it is exceptionally poor. The average duration of a frame render has been used as a rough benchmark in the development of this project, hence we measured the time in milliseconds between successive frames when the user moves the View Platform. By running the Editor application using the same hotel model and performing approximately the same manoeuvres it has been possible to compare the relative performance of different *dev08* and *Alpha01* implementations of Java3D. The results presented in Table 1 suggest that, while it is slow, performance improvements have been made in Java3D. The Just In-time Compiler (JIT) available for the JDK also provides a significant

Figure 5. A snapshot from a hotel tour inside a room.

JDK Version	Compiler	Java3D version	Time *ms*
JDK 1.1.5	Interpreter	dev08	2800
JDK 1.1.5	JIT	dev08	1490
JDK1.2 Beta3	Interpreter	Alpha01	1760
JDK1.2 Beta3	JIT	Alpha01	1270

Table 1. Performance comparison of various Java3D versions

performance improvement, however at the time of writing even the JDK1.1.5 JIT is specified as 'under development' and is not 100% stable. All the timings in milliseconds were recorded on a Pentium PC.

8. Conclusions & Future Work

In this case study, we have produced a full project plan including the final specification (not specified fully here) for a non-immersive VR application, namely a Hotel Browsing system. Some key aspects of this work are as follows.

The definitions are generic for other browsing applications; for example whether a system is required by an Estates Agent for viewing properties or a Car Dealership to demonstrate the inside of a vehicle, the principles described here will remain largely the same. Also, we have demonstrated a novel combination of Java3D and VRML. VRML was the ideal file format for this project as the Java3D scene graph matches very closely with the VRML structure. This correlation is likely to make it the preferred file format for many developers, although Java3D was designed to remain independent of any file format. Another novel method was the "1-step ahead" scheme. This was an intuitive necessity to comply with expected human movement in a virtual environment and had to be specifically designed to work around the Java3D provisions for collision detection. Finally, the user interfaces in this project were designed for (manipulation and visualisation of data by) non-VR experts.

All that remains is for the full system to be developed based on future funding. The full system will benefit from a greater range of Room types (to comprise the full hotel, i.e. corridors, seminar room, swimming pool, Bar, etc...) and a greater range of Feature objects. The future development of Java3D will heavily influence the technical aspects of this project.

Acknowledgements

The authors would like to thank Peter Gardner, Manager of Royal Swallow Hotel, Bristol, UK, for discussions and allowing us to measure rooms and other facilities in the hotel.

References

[1] Java3D. JavaSoft Java3D WWW Home Page. http://www.javasoft.com/products/java-media/3D.

[2] Alias Wavefront WWW Home Page. http://www.aw.sgi.com/.

[3] The VRML Consortium. The VRML Consortium WWW Home Page. http://www.vrml.org/.

[4] A. L. Ames, D. R. Nadeau, and J. L. Moreland. *The VRML 2.0 Sourcebook*. John Wiley & Sons, 1997.

[5] P.S. Strauss and R. Carey. An object-oriented 3d graphics toolkit. In *Proceedings of SIGGRAPH*, pages 341–349, 1992.

[6] K. Chung and W. Wang. Quick collision detection of polytopes in virtual environments. In *ACM Symposium on Virtual Reality Software and Technology*, pages 1–4, 1996.

[7] M. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In *Proceedings of IMA Conference on Mathematics of Surfaces*, pages 33–52, 1998.