Collision Handling and Shadow Casting in Virtual-Prismaker

Víctor López, Antonio Fernández-Caballero, Pascual González,

Francisco Montero & José Pascual Molina

Regional Development Institute, University of Castilla-La Mancha (Spain)

{victor, caballer, pgonzalez, fmontero, jpmolina}@info-ab.uclm.es

Abstract. Our research team is developing a computer construction game called Virtual-Prismaker based on the physical game Prismaker[tm]. We have had to deal with all typical and essential features of any realistic 3D modelling and animation environment. A first aim in these kinds of environments is to provide an easy way to manipulate the game pieces. The need to use standard input devices in a natural manner has guided us to limit movement freedom. This imposed limitation allows the pieces to be easily assembled. As we try to achieve a realistic simulation, collision handling and shadow casting are two more problems found up to date. In this paper we comment some lacks of the Java 3D[TM] application programming interface in this context, and we propose our solutions. Collision handling is faced by using a 3D grid that stores the position of all objects of the virtual playground. Shadow casting is simulated by drawing polygons according to a single light source and the environmental light and using the associated 3D grid position information.

Keywords: Virtual reality, Visualization and manipulation of virtual objects, Collision handling, Shadow casting, Educational games.

1. Introduction

The creation of software for children is usually linked to the binomy computer games and visual interfaces. Their playing issues and their attractive interfaces make these kinds of tools especially interesting for educational environments. Using computer games provides

new characteristics that improve any learning process. According to Sedighian et al [15] computer games supply meaningful learning, goals, success, challenge, cognitive artifact, and association through pleasure, attraction, and sensory stimuli. These factors are even enhanced when using 3D environments and attractive visual interfaces.

Virtual-Prismaker [7] is a project being carried out in conjunction with the corporation that created the physical game Prismaker[tm] [9]. This project tries fundamentally to analyze the game's utility inside educational environments. In order to achieve this goal a multidisciplinary team has been created, formed of computer engineers, psychologists, and pedagogues.

(a) (b)

Figure 1. (a) Prismaker[tm] system. (b) Virtual-Prismaker playground.

Prismaker[tm] is a construction game that provides a reduced number of kinds of pieces (see figure 1a) to build with. Besides conventional possibilities, you can use logos to assign a meaning to these pieces, and, this way, it is possible to extend the educational features of the game. This game is intended for children older than 3 years. The physical version of this game has received several prizes because of the educational features it incorporates.

In the current development of the virtual version of Prismaker[tm] we provide children

with a working environment as similar as possible to the way things exist in the real world. Virtual-Prismaker is a 3D environment based game (figure 1b) that tries to best simulate the possibilities offered by the physical version of Prismaker[tm] game. This environment uses the different kinds of pieces of Prismaker[tm] system to develop many different constructions.

Playing inside our 3D environment is carried out in a virtual room. Virtual-Prismaker has been designed for people without computer knowledge. To make these people easier to use Virtual-Prismaker, we are engaged in removing computer concepts, such as “load/save”, “menus”, and so on. We have designed a user interface based on metaphor use that offers the concepts the way a child is familiar with. For instance, we have replaced standard I/O operations (load/save) with some shelves where users can store their work, so it can be resumed later on at any moment.

In this same context it is necessary to greatly facilitate object manuipulation. The actual typical interaction devices have not been designed to control 3D movement. An arbitrary degree of freedom in pieces movement complicates any assembly task. That’s why we have decided to limit movement capacity, restricting to fixed rotation angles and fixed translation steps.

2. Java 3D: A brief description

The tool chosen to develop Virtual-Prismaker is Java 3D[TM]. The election was due to the fact that with Java 3D API constructs, application developers can describe very large virtual worlds, which, in turn, are efficiently rendered by the Java 3D API. Java 3D API's scene-graph based model makes it ideal for virtual reality systems and other applications that wish to represent and navigate complex 3D worlds.

Java 3D API was designed to satisfy among others the following goals [8]: (1) high performance, (2) rich set of 3D features, (3) high-level, object-oriented paradigm, (4) wide

variety of file formats. The Java 3D API's scene graph-based programming model provides a simple and flexible mechanism for representing and rendering potentially complex 3D environments. The scene graph contains a complete description of the entire scene, or virtual universe. This includes the geometric data, the attribute information, and the viewing information needed to render the scene from a particular point of view.

3. Java 3D: Some flaws

3.1. Collision handling

Since the advent of computer graphics, programmers have continually devised ways to simulate the world more precisely. Collision detection is one of the most important features to create realism. Nevertheless, collision handling in 3D is very difficult to implement in a realistic way. As virtual game users demand increasing levels of realism, collision detection and handling has to be implemented to approximate the real world in virtual worlds as closely as possible.

Java 3D uses *behaviors* [3] for animation and interaction. *Behaviors* are links to user code to update scene objects, both graphics and sound. *Behaviors* are fired when they get a stimulus. A stimulus is received when moving a mouse, pressing a key, getting some objects collided, detecting some other event, or a combination of them. Before an object is added to the scene graph (in order to become “live”) you have to set its *capabilities*. *Capabilities* define which object parameters can be modified. One such capability is the ENABLE_COLLISION_REPORTING capability.

So, Java 3D is able to report on collisions. Where is then the problem? The problem is that a collision is reported when it has already happened. This way, when you are able to stop animation, it is too late. In the real world, when two objects collide they crash. Thus, in a realistic virtual environment, collisions have to be detected before allowing any movement.

3.2. Shadow casting

Improvements in graphics accelerators have let programmers advance beyond the limitations of 2D games and create 3D environments with remarkable realism. But, even with fancy perspective divides and texture perspective correction, the onscreen rendering lacks a true sense of depth if there is no shadow casting. Without shadows the 3D illusion is sorely lacking. Indeed, real-time dynamic shadowing represents a huge leap forward in realism, depth perception, and the overall presence of objects within a 3D environment. Figure 2 illustrates how hard it is to find out where an object is really placed in a 3D world where there is no shadow casting.

(a) (b)

Figure 2. (a) Scene with no shadow casting. (b) Scene with shadow casting

Take a look at the picture on the left (figure 2a). It depicts two spheres sitting on, or floating above a base. It's hard to tell whether they are touching the base or if they are floating above it. Now look at the picture on the right (figure 2b). Suddenly all has become clear. The sphere on the right corner is resting at the back of the base. The other one is floating above the base, and it is closer to the observer. The only difference is that the picture on the right incorporates shadows. Thus, we must agree that casting shadows is essential to manipulate a 3D virtual world.

Unfortunately, Java 3D does not support shadow casting. That’s why this process must be done using conventional software algorithms. And that is the reason why we have added our proper shadow casting support to Virtual-Prismaker environment.

4. Proposed solutions

The kind of people the virtual game is designed for, and the limitation in the input devices currently in use, impose some restrictions to realism in order to achieve more manipulation and usability conditions. This is the reason why we have limited the allowed movements (translations and rotations) up to a reasonable degree. To solve this problem, a 3D grid capable of facing collision handling and shadow casting is used.

4.1. A 3D grid for collision detection

Different ways to manage collision detection have been considered so far [10] [11] [12]][13] [14]. The same way most of them are strongly dependent on the problem they deal with, our solution also does. In our virtual playground, children build constructions by only using pieces of a few different kinds.

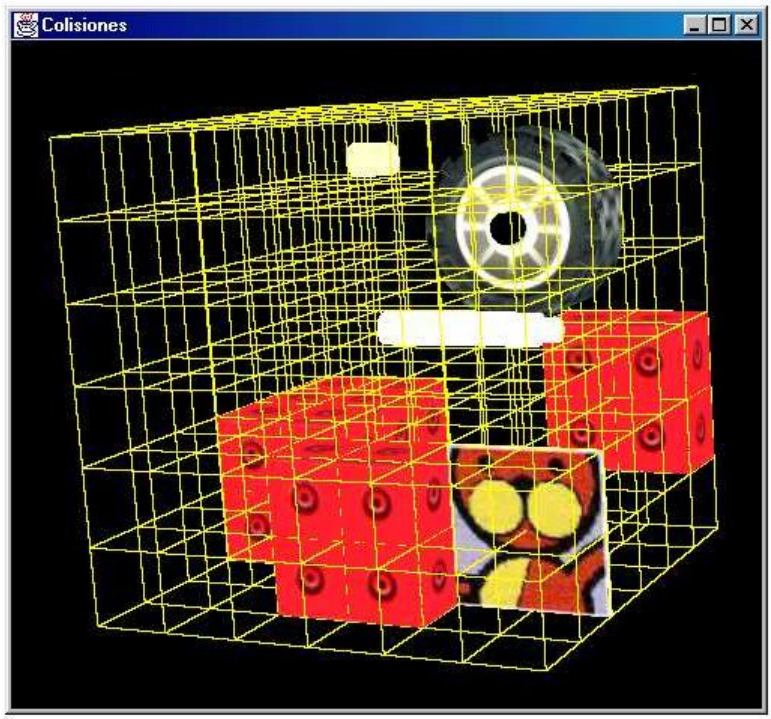

Figure 3. 3D grid to fit building blocks

We have broken these different kinds of pieces in smaller blocks that could fit in a 3D grid. This 3D grid fills the whole playground, as you may observe on figure 3. Our proposed solution is based on the capability of the 3D grid to store the positions of all objects present in the virtual playground.

When a movement takes place in Virtual-Prismaker, we have to follow two steps. Firstly, collision detection is performed. This step begins marking the current positions of the moving objects as free. Then, all the final positions of the moving blocks are tested to be free in the grid. If this is the case and none of these positions are outside the grid limits, it is concluded that no collision has occurred. If a collision is detected, movement is stopped. Secondly, scene positions and 3D grid positions of the objects are updated if no collision has been detected. This is the way we achieve fast collision detection to simulate real world behavior in our virtual playground.

4.2. Polygons for shadow casting simulation

As long as current 3D graphics cards do not support shadow casting, we have to deal with the problem directly. Many different algorithms for shadow casting have been proposed. Most of them are based on BSP trees [4], shadow volumes [5], or raster lines to merge shadow processing and visible surfaces [1] [2] [6]. All of them are fast, but maybe not enough for real-time rendering. Our proposal introduces a realistic shadow casting algorithm.

Apart from the environmental light, shadows in our virtual world are due to one single light source. All shadows are simulated using polygons. That’s why a first step will lead us to find out the shape of the polygon of the processed object’s shadow. As already mentioned, in our virtual environment there is a limited number of kinds of pieces, so the number of possible shapes is limited to a bunch of polygons.

In order to figure out the shape of the whole shadow the 3D grid is again found to be

of a great interest. Indeed, the fact that each cell of our 3D grid may offer part of a piece's shape, makes shadow casting faster and easier. If there is no block below a occupied 3D grid cell, then this cell will be used to create the shape of the shadow polygon, otherwise it will not. The union of these cells will give the parts of the objects that take place in obtaining the resulting complete shadow.

Lastly, the right color and the right size of the shape are obtained. To find out the right size is equivalent to getting the right scale for the polygon. As our single light source (some kind of lamp) is located above the playground, the scale we are looking for is proportional to the distance between the light source and the object. On the other hand, the transparency value (shadow color) is inversely proportional to the distance between the object and the floor. Transparency will decrease as the object gets nearer to the virtual floor. Figure 4 shows an example of shadow casting of a complicated construction.

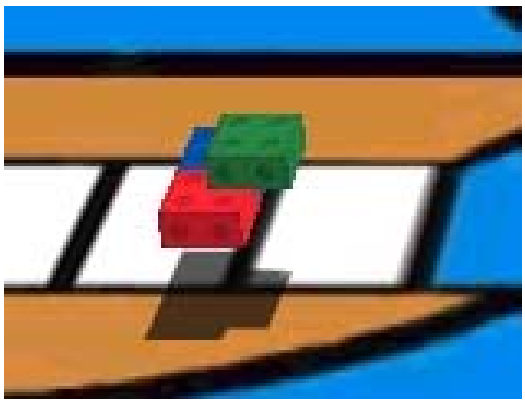

Figure 4. Shadow casting in Virtual-Prismaker

5. Conclusions

In this paper we have offered our solutions to the main problems we have found during the development of the construction game Virtual-Prismaker. First of all, we have faced the game's usability. An imposed limitation to movement allows the pieces to be easily assembled. This way we may lose some degree of reality but manipulation is greatly

improved. To recall a realistic 3D modelling and animation environment, a correct implementation of collision handling and shadow casting have appeared to be of a tremendous importance.

Collision handling has been performed by using a 3D grid able to store the positions of all objects in the virtual playground. As opposite to Java 3D focus, in our specific collision detection algorithm, collisions are detected before they really occur. A movement is permitted or denied on the basis of the 3D grid cells occupation states.

Shadow casting has been simulated by means of polygons from the 3D grid object's position information. Each cell of our 3D grid offers part of a piece's shape. The union of these shapes result in the complete shadow. The scale value is proportional to the distance between the single light source and the object, while the transparency value is inversely proportional to the distance between the object and the floor.

Acknowledgements

This work is supported in part by the CICYT–FEDER 1FD97–1017 grant and by the 2000020264 JCCM and European Social Fund.

References

[1] A. Appel, "Some Techniques for Shading Machine Renderings of Solids", *SJCC*, 37-45, 1968.

[2] W.J. Bouknight, "A Procedure for Generation of Three-Dimensional Half-Tones Computer Graphics Presentations", *Communications of the ACM*, 13 (9), 527-536, 1970.

[3] D.J. Bouvier, "Getting Started with the Java 3D API", Sun Microsystems, Inc., 1999.

[4] Y. Chrysanthou, M. Slater, "Shadow Volume BSP Trees for Computation of Shadows in Dynamic Scenes", *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, 45-

50, 1995.

[5] F.C. Crow, “Shadow Algorithms for Computer Graphics", *SIGGRAPH 77*, 242-247, 1997.

[6] J.D. Foley, et al., "Introduction to Computer Graphics", Addison-Wesley, 1994.

[7] P. González, F. Montero, V. López, A. Fernández-Caballero, “A Virtual Learning Environment for Short Age Children”, To appear in *Proceedings IEEE International Conference on Advanced Learning Technologies, ICALT 2001*.

[8] http://java.sun.com/products/java-media/3D/index.html

[9] http://www.prismaker.com/

[10] P.M. Hubbard, "Collision Detection for Interactive Graphics Applications," *IEEE Transactions on Visualization and Computer Graphics*, 1 (3), 218-230, 1995.

[11] P. Jiménez, F. Thomas, C. Torras, “3D Collision Detection: A Survey”, *Computers and Graphics*, 25 (2), 269-285, 2001.

[12] Y. Kitamura, A. Smith, H. Takemura, F. Kishino, "A Real-Time Algorithm for Accurate Collision Detection for Deformable Polyhedral Objects", *PRESENCE*, 7 (1), 36-52, 1998.

[13] J. Klosowski, M. Held, J. Mitchell, H. Sowizral, K. Zikan, “Efficient Collision Detection using Bounding Volume Hierarchies of K-dops”, *IEEE Transactions on Visualization and Computer Graphics*, 4 (1), 1998.

[14] M. Lin, S. Gottschalk, “Collision Detection Between Geometric Models: A Survey”, *Proceedings of IMA Conference on Mathematics of Surfaces*, 1998.

[15] K. Sedighian, A.S. Sedighian, “Can Educational Computer Games Help Educators Learn About the Psychology of Learning Mathematics in Children?”, *18th Annual Meeting of the International Group for the Psychology of Mathematics Education*, Florida, USA, 1996.