ICAT 2003
December 3-5, Tokyo, JAPAN

Manipulation Using Magnet Metaphor for 2D and 3D Integrated Toolkit Systems

Yoshihiro Okada[1,2], Yoshiaki Akazawa[1] and Koichi Niijima[1]

[1]Graduate School of Information Science and Electrical Engineering, Kyushu University
6-1 Kasuga-koen, Kasuga, Fukuoka, 816-8580, JAPAN
{okada, y-aka, niijima}@i.kyushu-u.ac.jp
[2]Intelligent Cooperation and Control, PRESTO, JST

Abstract

This paper proposes a component-based 3D object manipulation framework using a magnet metaphor for 2D and 3D integrated toolkit system. The authors have been studying component based 3D software development systems. For the development of 3D graphics applications, the layout of 3D objects is important factor. However, it is very laborious work and it takes a long time because 3D objects have six degrees of freedom (DOF) and it is not easy to lay out them using a standard 2D input device, i.e., a mouse device. In the real world, every object exists contacting with other objects and then it has three DOF, i.e., x-y translation and z-axis rotation because the gravity exists. Then, the authors introduced new component that behaves like the magnet to simulate the simplified gravity effect for positioning 3D objects. Furthermore, 3D graphics applications need 2D graphical components for the interface of interactive operations, e.g., pop-up menus, toggle buttons, pull-down menus and so on. This paper also proposes the construction of such 2D interfaces using 3D primitives. The new 3D component using a magnet metaphor also makes it easier to construct such 2D-like 3D graphical user interfaces.

Key words: Virtual Reality, 3D toolkit, 3D layout, Object manipulation, *IntelligentBox*

1. Introduction

This paper proposes a 3D object manipulation method using a magnet metaphor for 2D and 3D integrated toolkit systems. The authors have been studying 3D software development systems. The research group, to which one of the authors previously belonged, has already proposed a component-based 3D graphics software development system called *IntelligentBox* [1,2]. *IntelligentBox* provides 3D software components called *boxes*. Each *box* has a 3D visible shape and a unique functionality. *IntelligentBox* also provides a dynamic data linkage mechanism called *slot connection*. With this mechanism, the user can construct 3D graphics applications by combining existing *boxes* through direct manipulation on a computer screen without writing any text-based programs. Actually, for the development of 3D graphics applications, the layout of 3D objects is important factor. However, it is very laborious work and it takes a long time because 3D objects have six degrees of freedom (DOF) and it is not easy to lay out them using a standard 2D input device, i.e., a mouse device. In the real world, every object exists contacting with other objects and usually it has three DOF, i.e., x-y translation and z-axis rotation as shown in Figure 1 because the gravity exits. Therefore, in order to make the layout of 3D objects easier, we introduced a gravity field effect as a software component, i.e., a particular *box* called *MagnetBox*, into *IntelligentBox*. Indeed, the simulation of the complete gravity is impossible in real time so our *MagnetBox* only behaves like the magnet and it works in real time. This is a 3D visual component so the user can adequately use it for his/her applications in various ways. For example, this enables to provide a virtual 2D desktop. On this desktop, the user can lay out 3D primitive parts to build 2D-like 3D GUIs as if he/she would construct 2D GUIs on a 2D desktop. As a result, our system *IntelligentBox* came to have aspects of 2D and 3D integrated toolkit systems.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 explains essential mechanisms of *IntelligentBox*. Section 4 explains how *MagnetBox* works and shows several construction examples of 3D composite models. Section 5 explains the construction of 2D-like 3D GUIs. Finally section 6 concludes the paper.

2. Related work

Our research purpose is to clarify software architecture that makes it easier to develop interactive 3D graphics applications. Its related works are Virtual Reality construction toolkit systems including MR Toolkit[3], MERL[4] and so on. Most of them provide a script language like VRML [5]. When developing 3D graphics applications, developers have to make programs to define the behavior of each object existing in a 3D virtual world using a script language. As well, there are some distributed Virtual Reality systems such as MASSIVE[6], dVS/dVISE[7], DIVE[8]. Although

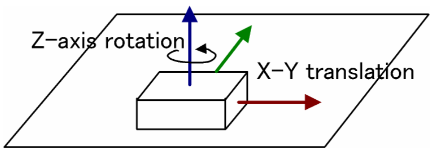

Fig. 1 Three DOF of the object contacting with on other object

these are very powerful systems, it is not easy to use their essential mechanisms for end-users.

Our research system *IntelligentBox* provides various 3D software components represented as visible, manually operable, and reusable objects. Furthermore *IntelligentBox* provides a dynamic data linkage mechanism called *slot connection*. These features make it easier for even end-users to develop interactive 3D graphics applications. This is the main difference between *IntelligentBox* and others.

Concerning the higher DOF problem for 3D object manipulations, there are researches on the use of higher DOF input devices. For example, the Roller Mouse (a three DCF mouse) [9], the Bat (a six DOF mouse) [10], and Data Glove exist. These devices allow the user to manipulate 3D objects as if he/she would do in the real world. However, these devices have two main problems. One is that these devices have their own manual operation way so the user suffers from the difference among such various ways. The other one is that specialized hardware is expensive and difficult to get.

Smith et al. [11] proposed the manipulation of 3D objects using a 2D user interface. This system employs contact constraints among 3D objects to allow the user to position 3D objects using a mouse device. Xu et al. [12] proposed a constraint-based automatic placement system using the pseudo-physics engine [13]. The purpose of these researches is the same as that of our research. However, the approach of our research is different from the others. In this paper, we propose the component-based approach.

3. Essential mechanisms of *IntelligentBox*

IntelligentBox employs the following essential mechanisms inherited from *IntelligentPad*[14,15], which is a 2D synthetic media system because *IntelligentBox* is an extension of *IntelligentPad* to 3D graphics applications.

3.1. Model-Display object (MD) structure

As shown in Figure 2, each *box* consists of two objects, a *model* and a *display object*. This structure is called an *MD* (*Model-Display object*) structure. A *model* holds the state values of a *box*. They are stored in variables called *slots*. A *display object* defines how the *box* appears on a computer screen and how the *box* reacts to user operations.

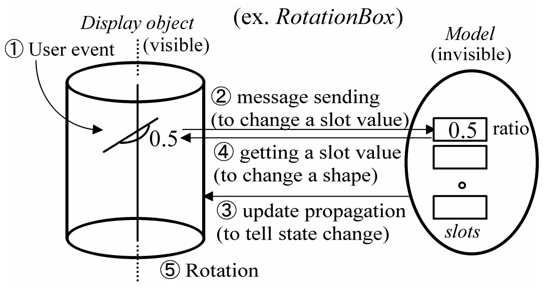

Fig. 2 *MD* structure of a *box* and its internal messages

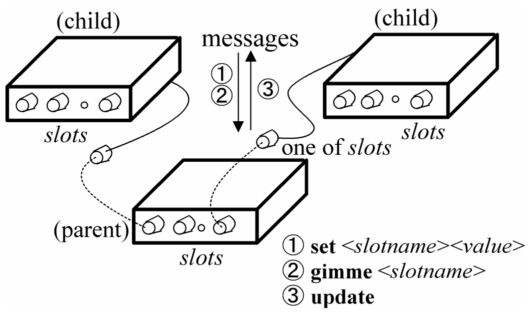

Fig. 3 Standard messages among *boxes*

Figure 2 shows messages between a *display object* and a *model*. This is an example of *RotationBox*. *RotationBox* has a *slot* named 'ratio' that holds a double precision number, which means a rotation angle. Through direct manipulations on a *box*, its associated *slot* value changes. Furthermore, its visual image simultaneously changes according to the *slot* value change. Then a *box* reacts to the user manipulations according to its functionality.

3.2. Message-sending protocol for slot connections

Figure 3 illustrates a data linkage concept among *boxes*. Each *box* has multiple *slots*. Its one *slot* can be connected to one of the *slots* of other *box*. This connection is called *slot connection*. The *slot connection* is carried out by three messages, i.e., a *set* message, a *gimme* message and an *update* message, when there is a parent-child relationship between two *boxes*. These messages have the following formats:

(1) Parent box *set* <slotname> <value>.
(2) Parent box *gimme* <slotname>.
(3) Child box *update*.

A <value> in a format (1) represents any value, and a <slotname> in formats (1) and (2) represents the user-selected *slot* of the parent *box* that receives these two messages.

A *set* message writes a child *box slot* value into its parent *box slot*. A *gimme* message reads a parent *box slot* value and sets it into its child *box slot*. *Update* messages are issued from a parent *box* to all of its child *boxes* to tell them that the parent *box slot* value has changed. Each *box* has three main flags that control the above message flow, i.e., a *set* flag, a *gimme* flag, and an *update* flag. These flags are the properties of a *display object*. A *box* works as an input device if its *set* flag is set to true. Contrarily a *box* works as an output device if its *gimme* flag is set to true. A *box* sends *update* messages if its *update* flag is set to true. Then child *boxes* take an action depending upon the states of the *set* flag and the *gimme* flag after they receive an *update* message or after they individually change their *slot* values.

4. 3D object manipulation using magnet metaphor

In this section, we introduce standard 3D object manipulations supported by *IntelligentBox* and describe their problem when positioning 3D objects. Then we explain how *MagnetBox* works in *IntelligentBox* to compensate it, and shows several positioning examples.

4.1 Standard manipulation of 3D objects

Figure 4 shows the standard coordinate system of 3D graphics systems. *IntelligentBox* also has the same coordinate system. Figure 5 shows standard translation operations in *IntelligentBox*. The left-right movement of a mouse device corresponds to the x-translation, its up-down movement corresponds to the y-translation, and the up-down movement with pushing the right button of

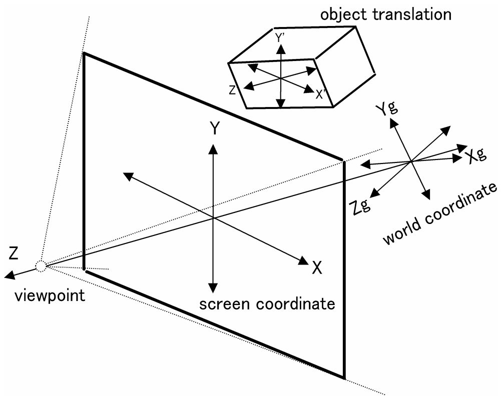

Fig. 4 Standard coordinate system

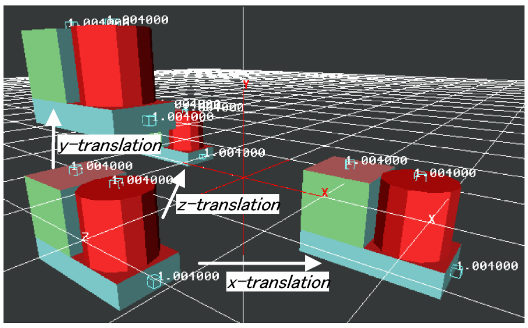

Fig. 5 Standard translation operations in *IntelligetnBox*

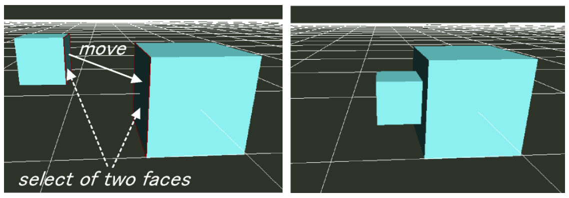

Fig. 6 Moving an object to make its face touch the face of other object

a mouse device corresponds to the z-translation. These x-, y- and z- axes mean the screen coordinate as shown in Figure 4. With these translations, it is still difficult to put 3D objects on other object and move on it. Indeed, *IntelligentBox* provides three dedicated operations for such a case. One is to move a 3D object to make its specific face touch the specific face of other 3D object as shown in Figure 6. Actually, this operation needs extra operations to select the two specific faces. The remaining two are to move and to rotate a 3D object on its specific face as the same as shown in Figure 1. These operations need many menu selections. This is not convenient and not intuitive manner. Furthermore, it needs many set of operations for positioning 3D objects especially on a composite object composed from several primitive objects. *MagnetBox* makes it simple as follows.

4.2 Functionality of *MagnetBox*

As mentioned in Sec. 1, *MagnetBox* has not a real gravity effect but a magnet effect. That is, each 3D object defined as a descendant of *MagnetBox* always touches its parent object. When, through the manual operation using a mouse device, the user translates any 3D object that is a descendant of *MagnetBox*, the 3D object moves on the surface of its parent object like a creeping motion as shown in Figure 7 (a). Normal arrow

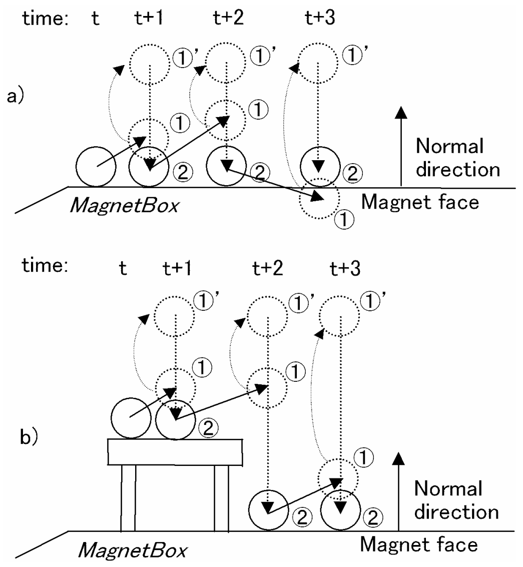

Fig. 7 Moving an object with touching the surface of its parent object

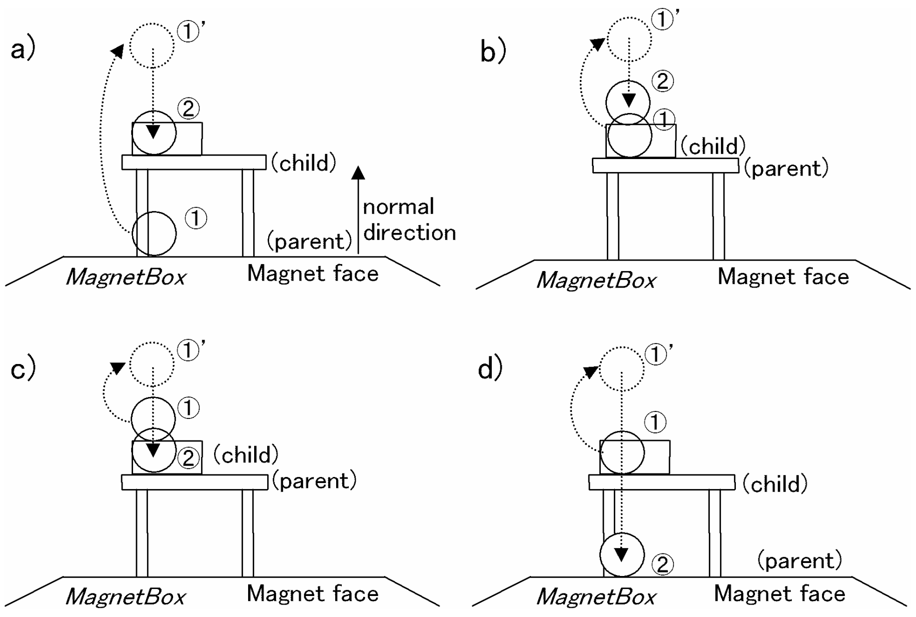

Fig. 8 Object positioning example using *MagnetBox*

lines mean the movement of a 3D object by a standard translation operation. t, t+1, t+2 and t+3 mean frame times. In each frame, the standard translation operation executed by the user moves the target object to the position ①, and then, *MagnetBox* once moves it to the position ①' where is enough high from the magnet face and moreover moves the target object down until it touches its parent object (or its ancestor object shown in Figure 7 (b)). A magnet face is the user-specific face of *MagnetBox*. Finally the target object will be located at the position ②. As a result, the target object moves on the surface of its parent object like the sequence of position ② shown at each frame time.

As explained above, usually 3D object moves on the surface of its parent object. However, there is the case that a desk exists on a floor and the target object also exists on the floor, and the user wants to move the target object onto the desk. Also, there is the opposite case that the target object exists on a desk and the user wants to move it down onto the floor. Our implementation makes it possible by using a keyboard operation. Figure 8 (a) and (d) show the former case and the latter case respectively. When the user clicks a specific key of a keyboard, e.g., a 'u' key, the 3D object goes down under the desk as shown in Figure 8 (d). On the other hand, when the user clicks another key, e.g., a 'u' key, the 3D object goes up on the desk as shown in Figure 8 (a). Moreover, in this case, if the user clicks a 'u' key again, the 3D object goes upon the object located on the desk as shown in Figure 8 (b). From this situation if the user clicks a 'd' key again, the 3D object goes down onto the desk as shown in Figure 8 (c). In the all cases, the target object once goes up to the position ①' from the initial position ①, and then, goes down until it touches any other object. Strictly speaking, when the user clicks a 'u' key, i.e., the cases (a) and (b), *MagenetBox* moves the target object up to the position ①', and then, moves it down until it touches any sibling object. On the other hand, when the user clicks a 'd' key, i.e., the cases (c) and (d), *MagnetBox* once moves the target object up to the position ①', and then, moves the target object down until it touches any ancestor object of its parent.

As you see figure 8 and imagine, these operations also change the parent-child relationship concerning the target object automatically. For example, in the case of Figure 8 (a), after the operation, the target object becomes the child of the desk. If the user moves the desk, the target object still moves with attached to the desk. In the case of Figure 8 (b), after the operation, the target object becomes the child of the object located on the desk.

4.3 Discussion

Figure 9 shows an actual layout example. Usually, without using *MagnetBox*, it takes a couple of minutes to compose the composite component of the right figure from separate primitive objects of the left figure. However, with using *MagnetBox*, it takes less than one minute. This is a very simple composition example and its elapsed time was very short. However when making more complicated scene, it needs longer time. In such a case, the use of *MagnetBox* comes to have the big

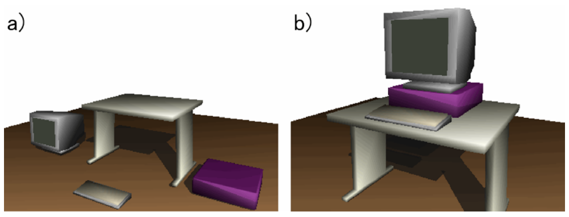

Fig. 9 Composition result (b) composed from primitive components (a)

advantage.

There is a problem because, for simplicity, our algorithm cannot detect the collision of a target object with inside faces of other object. For example, a bookshelf has several shelves its inside so it is impossible to locate books on its shelves. Figure 10 shows one of the solutions to this problem. Each shelf should be an individual component and a parent-child relationship should be assigned between any two adjoining shelves. In such a case, with using keyboard operations, the user can put a book on any shelf.

Indeed, we have already proposed a random layout algorithm for 3D scene generations [16, 17]. Figure 11 shows four random layout results generated by the algorithm. However, to obtain user required 3D scenes by modifying the 3D scenes once generated by the random layout algorithm, user's manual operations are necessary. In such a case, 3D object manipulations using *MagnetBox* must reduce the cost of such layout work drastically.

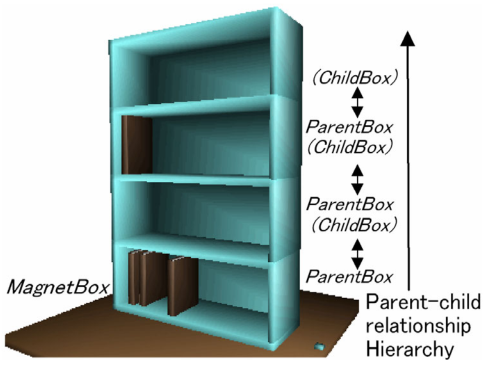

Fig. 10 Positioning books in a bookshelf using *MagnetBox*

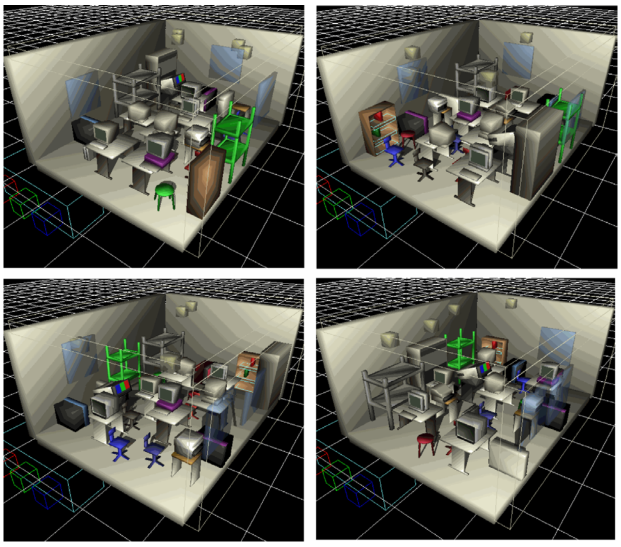

Fig. 11 Four random layout results generated by a prototype system

5. Virtual desktop: construction of 2D-like 3D GUIs

3D graphics applications need 2D graphical components for the interface of interactive operations, e.g., pop-up menus, toggle buttons, pull-down menus and so on. Our *MagentBox* help us to construct such 2D-like 3D GUIs.

5.1 3D Primitives

3D primitives of a screen image of Figure 12 are *RotationBoxes*, *SliderMeterBoxes*, *ListBoxes* and so on. With using *MagnetBox*, the user can lay out them to build 2D-like 3D GUIs as if he/she would manipulate on the real desktop of a computer display like pasting/peeling operations. This reduces the time consuming for building 2D-like 3D GUIs.

5.2 Composition example of 2D-like 3D GUIs

Figure 13 shows one composition example of 2D-like 3D GUIs and its hierarchical structure. In the figure, the symbols 'ratio', 'state', 'alpha_value' and 'opacity' mean *slot* values of *boxes*. The root object is *IOBufferBox* and its alpha_value *slot* is connected to the ratio *slot* of *RotationBox* and also connected to the ratio *slot* of *SliderMeterBox*. The opacity *slot* of the *IOBufferBox* is also connected to the state *slot* of *ToggleSwitchBox*. The *ToggleSwitchBox* controls availability of the opacity of the *IOBufferBox*. As well, both the *SliderMeterBox* and the *RotationBox* works to change the transparency level of the *IOBufferBox*.

To understand how this composite *box* works in more detail, see the demonstration video on the web of

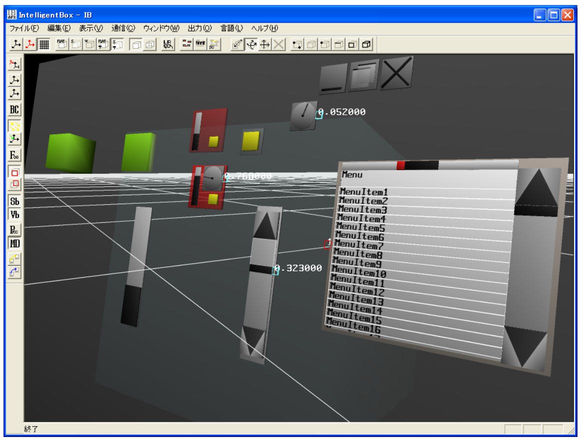

Fig. 12 Virtual desktop: positioning 3D primitives by pasting/peeling operations

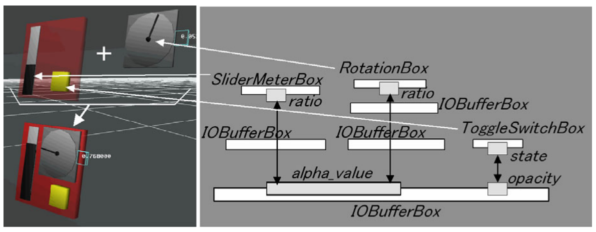

Fig. 13 Composition example of 2D-like 3D GUIs

ICAT2003 conference or on http://www.i.kyushu-u.ac.jp/~okada/extern/Publication/ICAT2003/index.html .

5.3 *VisorControlBox*

To provide a virtual desktop, another *box* called *VisorControlBox* is used with *MagnetBox* in practical cases. *VisorControlBox* controls the view direction in order to keep it as the same as the normal direction of the magnet face of *MagnetBox*. With *VisorControlBox*, the user sees 2D-like 3D primitive components on the magnet face as if they would exist on a 2D flat display screen.

6. Concluding remarks

This paper proposed a component-based 3D object manipulation framework using a magnet metaphor for 2D and 3D integrated toolkit systems. By considering the gravity, it is possible to reduce the DOF of 3D objects and to make the positioning of 3D objects simple. We introduced a particular component called *MagnetBox* into *IntelligentBox* in order to simulate the simplified gravity effect. *MagnetBox* is also used to provide a virtual desktop in a 3D space and hence is useful for the construction of 2D-like 3D GUIs. In this paper, we explained how *MagnetBox* works and describe its usefulness.

The functionality of *MagnetBox* is based on the collision detection. Since we do not use any efficient collision detection algorithm, we will have to introduce one of the efficient collision detection algorithms to improve the performance of *MagnetBox*. Furthermore, we will have to develop some practical application examples of 2D-like 3D GUIs to evaluate the availability of *MagnetBox*. These are our future works.

References

1. Okada, Y. and Tanaka, Y., *IntelligentBox*: A Constructive Visual Software Development System for Interactive 3D Graphic Applications, Proc. of Computer Animation ''95, IEEE Computer Society Press, pp. 114-125, 1995.
2. Okada, Y. and Tanaka, Y., Collaborative Environments in *IntelligentBox* for Distributed 3D Graphic Applications, The Visual Computer (CGS special issue), Vol. 14, No. 4, pp. 140-152, 1998.
3. Shaw C, Green M, Liang J, Sun Y, Decoupled Simulation in Virtual Reality with the MR Toolkit. ACM Transaction on Information Systems, Vol. 11, No. 3, pp. 287-317, 1993.
4. Anderson DB, Barrus JW, Howard JH, Rich C, Shen C, Waters RC, Building Multiuser Interactive Multimedia Environments at MERL. IEEE Multimedia, 2(4): 77-82, 1995.
5. Ames, A. L, Nadeau, D. R. and Moreland, J. L., The VRML Sourcebook, John Wiley & Sons, Inc. , 1996.
6. Greenhalgh, C., Benford, S., MASSIVE: a Distributed Virtual Reality System Incorporating Spatial Trading. Proceedings of IEEE 15th International Conference on Distributed Computing Systems (DCS'95), pp. 27-34, 1995.
7. Grimsdale, C., dVS-Distributed Virtual Environment System. Proceedings of Computer Graphics '91, London, UK, Bleinheim Online, pp. 163-170, 1991.
8. Hagsand, O., Interactive Multiuser VEs in the DIVE System. IEEE Multimedia, Vol. 3, No. 1, pp. 30-39, 1996.
9. Venolia, D., "Facile 3D direct manipulation", *ACM SIGCHI*, pp. 31-26, 1993.
10. Ware, C., and Jessome, D.R. "Using the Bat: a six dimensional mouse for object placement", *IEEE computer Graphics & Applications*, 8(6): pp. 65-70, 1988.
11. Smith, G., Salzman, T., and Stuerzlinger, W., Integration of constraints into a VR Environment, VRIC 2001, pp. 103-110, 2001.
12. Ken Xu, James Stewart, Eugene Fiume, Constraint-based Automatic Placement for Scene Composition, Graphics Interface, pp. 25-34, 2002.
13. M. Shinya and M.C. Forgue, Laying out objects with geometric and physical constraints, The Visual Computer, (11):188-201, 1995.
14. Tanaka, Y., Meme Media and a World Wide Meme Pool, Proc. of ACM Multimedia '96, pp. 175-186, 1996.
15. Tanaka, Y., Meme Media and Meme Market Architectures for the Reediting and Redistribution of Knowledge Resources, Proc. of MultiMedia Modeling '98, IEEE Computer Society Press, pp. 2-11, 1998.
16. Akazawa, Y., Okada, Y. and Niijima, K. : Automatic 3D Object Placement for 3D Scene Generation, The European Simulation and Modeling Conference 2003 (ESMc2003), short paper, pp. 316-318, 2003.
17. Akazawa, Y., Okada, Y. and Niijima, K., 3D Scene Generation System and Its Intuitive Interface, to appear in EUROSIS 4th annual European GAME-ON Conference (GAME-ON2003), 2003.