MTK: A Direct 3D Manipulation Toolkit

MARCELO DE GOMENSORO MALHEIROS
FLÁVIO NAVARRO FERNANDES
WU, SHIN-TING

Grupo de Computação de Imagens
DCA/FEEC/UNICAMP - C.P. 6101
13083-970 Campinas, SP – Brazil
{malheiro,navarro,ting}@dca.fee.unicamp.br

Abstract: Advances in computer technology have made interactive 3D graphics systems a common tool for a variety of areas. Notwithstanding, most of such systems relies on popular 2D devices. To relieve system developers from the burden of implementing mapping techniques between 2D devices and 3D application data, some generic libraries have been proposed. However, those libraries still present limitations, either being complex to program or being limited in its interactive flexibility. As an alternative, we present a general purpose toolkit called MTK, which provides only basic interaction tasks. We also show that these tasks may be used to build sophisticated direct 3D manipulation techniques.

Keywords: Interactive 3D Graphics, Object-Oriented Design, Direct Manipulations, User Interface.

1 Introduction

Advances in computer technology have made interactive graphics systems a practical tool in a variety of areas such as engineering, science, art, medicine, entertainment, advertising, and education. Several interactive 2D graphics toolkits are already successfully used for the development of 2D graphics applications. Despite the availability of 3D graphics toolkits, such as GKS [1], PHIGS [11], and OpenGL [15], writing interactive 3D graphics applications is still a tedious, time-consuming, and difficult task.

According to Strauss and Carey [18], the main problem of those 3D toolkits is that they "are committed to display-oriented application development and provide little help for direct, 3D interaction support." Therefore, those toolkits can be thought of as *display list-oriented*. They also claim that traditional graphics packages cause data duplication, since "applications must convert their objects into structures required by the graphics packages, which maintains an additional copy." As a solution they propose the Open Inventor [20] toolkit, in which 3D objects are not merely display lists: they are abstract representations that can render themselves and make geometric computations when necessary. Open Inventor provides several levels of extensibility to enable applications to create their own objects with their own rendering methods and direct interactions.

By integrating application data into the 3D graphics toolkit, any required object properties, geometry, and operations need to be part of the toolkit. This *abstract 3D object-oriented* approach certainly facilitates the management of direct manipulations on the 3D objects held by that architecture. However, it is much more than a simple graphics toolkit. It is indeed an extensible 3D modeling, display, and interaction toolkit. Therefore, the simplicity and modularity of the resulting design could be compromised, as already observed by Shroeder et al. [17].

Another drawback of this abstract 3D object-oriented approach is that it solely reduces the application development efforts when the object information can be directly represented by the structure available in the toolkit. Otherwise, the developer must extend the toolkit by adding new object representations, with all the necessary geometric, display, and interaction operations. It again can be a non-trivial and time-consuming task. Furthermore, since there are no conventions for the new object code design, its reusability is arguable.

As an intermediary solution of the two abovementioned approaches, this paper presents a toolkit that follows a *smart display list-oriented* approach. This approach is "smarter" than the conventional display lists in the sense that the toolkit has more knowledge about the 3D object representation, just enough to allow direct manipulations. And, in comparison with the abstract 3D object-oriented one, the data structure needed by the proposed toolkit is much simpler, comprising only identifiable sets of graphic primitives (points, lines, and polygons).

The key idea of our approach is to establish a precise frontier between application data models and application-independent visualization and interaction techniques, without affecting the flexibility of direct manipulations achieved by the abstract 3D object-oriented approach. Thus, it is possible to design a 3D graphics toolkit which is only responsible for the application-independent functionalities. Consequently, the reusability of the toolkit is ensured.

This decoupling is accomplished in two fronts: visualization and interaction.

The visualization process adopted is similar to the conventional display list-oriented approach: the application needs to convert its internal representation into simple primitives. The difference resides in the fact that the application can arrange such primitives into identifiable sets, establishing a level of granularity.

The interaction process occurs restricted to the granularity imposed by the application. That is, only sets of primitives meaningful to the application can be manipulated, and each set is treated as a single unit. Additionally, the toolkit provides functionalities that define the *syntax* of the interactions, like virtual replacements for the application objects, called *draggers*. The actual *semantics* of manipulations is given by the application itself.

This paper is organized as follows. In section 2 some related works are discussed. In sections 3 and 4 the architecture of the proposed toolkit is presented. An overview of its implementation is given in section 5. The evaluation of the toolkit is discussed in section 6. Finally, in section 7, some conclusions are drawn.

2 Related Works

Even though 2D input devices could be considered more limited than 3D devices, the two-dimensional ones present some advantages over its 3D counterparts [13]:

- They are still the most popular devices for graphic input.
- The forearm can usually rest on the table, while three-dimensional devices fatigue the operator.
- Normally, 3D interaction is embedded in a 2D window, and switching between the modes can be an unproductive task.
- As they have one or two degrees of freedom, they are easier to control than a device with a higher number of degrees of freedom.

Probably these advantages are a major motivation for several works that address the problem of providing direct 3D manipulations using only 2D devices.

Early studies were made in order to create techniques to perform rotation of objects in the three-dimensional space. Evans' [8] *trackball* and Chen's [6] *virtual sphere* are very similar in function, as both allow continuous rotation in x and y by dragging the mouse in straight directions, and rotation in z through circular gestures.

Another approach, based on the geometry of objects in the scene and the use of a 3D cursor, was adopted by Nielson and Olsen [16] and Bier [2, 3]. Nielson and Olsen explored the geometry of the object actually being manipulated to perform geometric transformations, namely translations, rotations, and scaling. They also used a 3D cursor, the *triad*, to specify a point in three-dimensional space. On the other side, Bier used the geometry of the object being manipulated to control the spatial movements of another 3D cursor, called *skitter*. Bier also introduced the so-called *alignment objects* and *gravity functions* to produce more precise and smoother *skitter* motions.

Research in manipulations through auxiliary 3D graphic entities was also developed. Emmerik [7] presented an interface to enable the specification of 3D transformations (translation, rotation, and scaling), based on a virtual object represented by three orthogonal axes. Castier et al. [4] proposed two manipulation techniques using a bounding box. The first provided constrained rotations through the interaction with the box edges, while the second enabled translations on the directions perpendicular to the box faces.

Herndon [10] presented an interesting technique, the *interactive shadows*, which is a set of widgets that allows constrained direct 3D manipulations. It also provides valuable perceptual cues about the spatial relationships between objects. These widgets are projections of a 3D object onto a plane, like a shadow on the wall. The projection and the object are related, so that operations performed on one are reflected on the other.

Since such techniques are largely used by interactive 3D graphics applications, efforts have been made in developing commercial and research libraries providing such functionalities.

One example of such toolkit is Open Inventor [20]. It is *abstract 3D object-oriented* in the sense that it defines an object-oriented framework for describing and manipulating 3D scenes. Scenes are arranged in a *scene graph*, built with *nodes* that hold information as shapes, geometric transformations, materials, lights, and cameras. Interaction is accomplished through the use of *manipulators*, a special kind of node that reacts to events and can be edited directly by the user.

Open Inventor was designed with extensibility in mind, enabling application writers to add new, specific objects to the toolkit when necessary. The way to extend the set of objects is to derive new classes from existing ones. Because this extension demands the knowledge of the underlying implementation details, it might be a complex process. As an alternative approach, we propose a context-free toolkit that does not rely in extension mechanisms for supporting different applications.

Another toolkit, the Act [5], is also an *abstract 3D object-oriented* library, being inspired by Open Inventor. It supports an interpreted language, Lua [12], that allows dynamic inclusions of new objects. It also supports configurable constrained movements of a 3D cursor. Despite these features, the incorporation of application-dependent data still has the same drawbacks as in the case of Open Inventor.

The IQL, *Interactive Quick Line* [14, 19], was designed to be a *smart display list-oriented* toolkit. Inspired by PHIGS [11], it provides a hierarchical model of 3D graphic entities, used to display and manipulate the application data. Some interaction techniques such as Nielson's *triad*, Evans' circular *trackball*, and an alignment plane are available. However, IQL suffers from poor 3D graphics support, being entirely written on top of Xlib.

In sections 3 and 4 we present the design of our toolkit, denominated MTK. This toolkit provides visualization and basic 3D interaction tasks, such that each application can define its own manipulation semantics accordingly to its requirements.

3 Design Issues

The toolkit aims to offer facilities for implementing 3D interactive graphics interfaces, without knowing what the underlying applications will be. It is actually a mediator between applications and users in two major processes: visualization (output) and interaction (input).

For the visualization, the data flow is typically from the application models to the presentation functionalities. Therefore, the interface between them is relatively simple, normally consisting of data representation conversions. This procedure is already vastly used by the conventional 3D graphics packages, so we chose to maintain this strategy by adopting a structure for identifiable sets of graphic primitives in our toolkit.

A suitable interaction strategy can be achieved by distinguishing the basic interaction tasks from the composite ones [9]. A basic interaction task corresponds to an indivisible *unit of information* that is meaningful in the context of the application. In the case of interactive 3D applications they are: positioning, identification, entering text, and entering numeric quantities. Composite interaction tasks are built on top of the basic ones, actually being combinations of them and performing a *unit of operation*. Typical composite tasks for interactive 3D applications are model construction and direct manipulations on this model.

Nowadays the entry of a unit of 2D information is directly accomplished by a 2D input device. It is usual to categorize, according to their basic tasks, such devices into five major logical classes: *locator* for positioning, *pick* and *choice* for selecting, *valuator* for entering numeric quantities, and *keyboard* for entering text. In spite of numerous efforts to extend 2D devices to provide 3D information, no solution has proved to be definitive. This arises from the lack of a third degree of freedom in 2D devices.

When we use a mouse for 3D interaction we need to map the generated 2D movements into motions in space. Additionally, by displaying 3D objects in 2D displays, we need to distinguish the spatial relationship between those objects and the pointing device. From our point of view, analogously with 2D input devices, the techniques that provide such mapping and improved perception are context-free, applicable to any 3D graphics applications. Thus, we propose designing a toolkit that includes the functionalities that can replace the role of 3D input devices.

According to our design philosophy, all functionalities should be implemented as modular components with simple interfaces, so that they can be readily assembled into composite interaction tasks. Note, however, that the interaction process depends on visualization techniques for providing visual feedback, and this dependency could compromise the aimed modularity. To solve this conflict we introduce the concept of *manipulators*, to be discussed in section 6.

4 Architecture

To accomplish our simplicity and modularity goals, the MTK toolkit is comprised of six components: graphics model, cameras, selection, constraints, draggers, and guides (see Fig. 1).

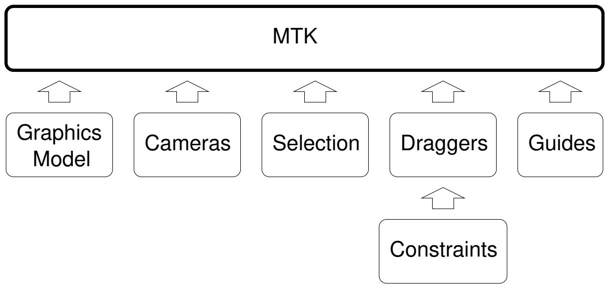

Figure 1: Toolkit architecture.

4.1 Graphics Model

The data structure maintained by the toolkit is based on identifiable display lists, called *elements*. An element is made of an arbitrary number of graphic primitives, which can be points, line segments, and polygons, together with their graphic attributes.

When the application creates an element, it has to choose a unique identifier for it. This identifier will be used to refer to that specific element in all operations of the toolkit. In particular, elements can be created and destroyed at any time, and more primitives can be added to their current contents.

In spite of the data duplication caused by the use of the toolkit along with the application, this approach seems well suited to our needs. First, the low-level geometric information stored in the toolkit is enough for the identification process. Second, by using an element as the smallest manipulation unit, the application can control how its model is perceived. For example, although a sphere could be made of small flat faces, if it is represented by a single element, then it will be manipulated as just one object. On the other hand, if the application creates one element for each face of the sphere, then every face will be individually distinguishable.

4.2 Cameras

This component is responsible for one or more virtual cameras maintained by the toolkit, which show a particular view of the objects belonging to the graphics model. Each camera is independent, having its own set of attributes, like viewing direction, position, aperture, and type of projection.

4.3 Selection

This component provides two basic functionalities: the maintenance of selection state and the identification mechanism.

The selection is a binary state associated with all elements in the toolkit data structure. The meaning of a selected state is given by the application. Typically it indicates which selected elements can be manipulated (see section 6).

The identification occurs when a user "clicks" a pointing device over the output of a camera. This process first detects whether that action occurred on top of some graphic primitive. If that happened, then the element that contains such primitive is located. Finally, the state of this element is changed accordingly to its previous state.

4.4 Constraints

When using a 2D device to interact with a 3D world, specific mappings are needed to establish a correspondence between 2D movements and 3D displacements. These mappings are defined by information provided by the cameras and by *constraints*. The constraints are used during the interaction to obtain an implicit depth coordinate, lost when projecting the spatial scene onto a 2D display.

There are several kinds of constraints, like those discussed in [3]. Among the many available, the toolkit tries to address the most common ones. For example, a movement in space can be restricted to lie on a given plane, or to occur parallel to a specified axis. Additionally, constraints can be continuous or discrete. In other words, the mapping can be free along the extents of some geometrical object or be restricted to a predefined set of disjoint points on that object.

All constraints follow the same mechanism. First, a constraint is selected. When an interaction starts, a pair of points is passed to the active constraint: a 2D point, obtained from the pointing device, and a 3D point, derived from the underlying 3D scene. Next, for all following device movements, the constraint mechanism is fed by the pointer offset. Each such offset is then converted into a 3D displacement, which tracks the pointer location into space.

4.5 Draggers

A *dragger* is a virtual object with a 3D graphic representation. Draggers have simple geometry and are used to give a consistent syntax to manipulate different objects. In some cases the dragger can actually substitute the object being manipulated, when the geometric complexity of the latter limits its real-time interaction.

A dragger is comprised of parts, which in turn are made of graphic primitives. Each part is sensitive, being capable of receiving user actions. A particular constraint can be attached to each part of a dragger, providing a convenient translation of the 2D input actions into 3D displacements.

Note that draggers present simple interaction mechanisms that can be used by the application to build complex operations. In fact, we can view the draggers as part of the *syntax* provided by the toolkit for direct 3D interactions. The specific *semantics* of those manipulations is given by the application itself.

4.6 Guides

The toolkit provides another set of virtual objects, called *guides*. These objects are passive, thus being unable of responding to user actions.

Their primary use is as visual aids. They can serve as a reference for the cameras, or can be used to improve the depth perception of the scene. Additionally, guides can provide a visual feedback for constraints applied to draggers. For example, a plane-like guide can be set to represent a "ground" for the scene, where draggers are restricted to lie onto.

5 Implementation

The proposed toolkit was developed using C++ in a Solaris environment with OpenGL hardware acceleration. The implementation resulted in a library containing about eight thousand lines of source code.

The toolkit functionalities can be highly customized to suit the application needs. For example, the selection mechanism can call application-defined callbacks to signal changes on the state of elements. The toolkit also provides a large set of constraints to control the interaction with the draggers. To cite a few: `sameDepth`, which restricts the displacements to a plane parallel to the projection plane; `snapAxis`, where the displacements occurs along a specified axis; `snapAxisD`, similar to the previous one, but giving a discrete motion; `snapParallelPlane`, makes the displacements lie on a plane parallel to another; and `snapSphere`, which restricts the motion to the surface of a specified sphere.

Currently the toolkit implements three types of draggers: a *bounding box* (Fig. 2), a *trackball* (Fig. 3), and a *jack* (Fig. 4). Those draggers were selected among the most common auxiliary objects found in the literature. An arbitrary number of draggers can be created and customized to implement a specific interaction strategy. In particular, the application can install callbacks in each dragger part, which are called when the user interacts with that part. The use of this feature is exemplified in section 6.

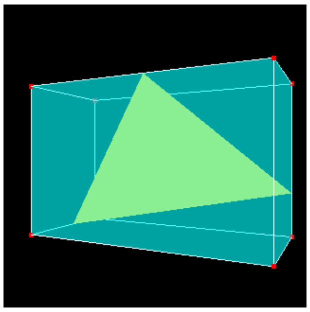

Figure 2: A bounding box dragger.

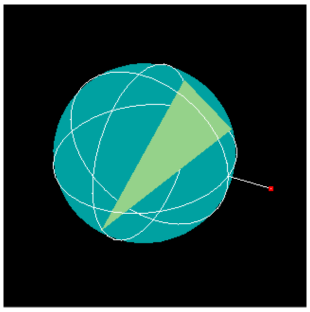

Figure 3: A trackball dragger.

There are also two types of guides available: an triple *axis* and a planar *grid*. The axis is made of three perpendicular segments, while the grid consists of a parallelogram on top of a predefined plane. Both can be seen in Fig. 5.

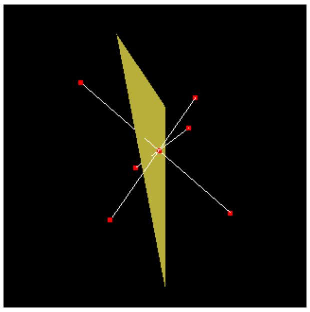

Figure 4: A jack dragger.

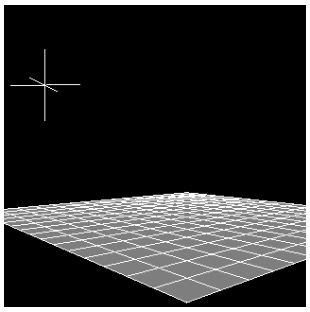

Figure 5: Two guides.

6 Empirical Evaluation

To evaluate the functionalities of MTK we implemented common composite 3D interaction tasks on top of the available draggers: the application of affine transformations. A *manipulator* associates a dragger with a set of composite 3D interaction tasks. A manipulator also coordinates the data exchange between a specified application model and the toolkit, in such a way that the application responds consistently to user actions.

In this section we describe the implementation of three manipulators, based on the draggers already provided by the toolkit. Then we exemplify the use of those manipulators in the context of a simple geometric modeler.

6.1 Manipulators

Typical operations on geometric data are rotations, translations, and scaling. These operations can be derived from sequences of positions in space. For example, to specify a translation we need two 3D points, which define a displacement vector.

To achieve this, we need to associate adequate constraints to each dragger part. When manipulated, these parts produce the desired sequence of points, which is then used by the manipulator to perform a specific operation.

To illustrate this idea, we present below three different manipulators, each one being capable of performing translations, rotations, and scaling.

6.1.1 Bounding Box

The bounding box dragger allows direct interaction with its eight vertices, twelve edges, and six faces.

In implementing our bounding box manipulator, which is inspired by [4], we assigned scaling to vertices, rotation to edges, and translation to faces. Two consecutive positions of a vertex are mapped into a scaling factor. Two consecutive points over an edge are mapped into a rotation about an axis, being parallel to the selected edge and passing through the center of the bounding box. Finally, two consecutive positions of a point on a face are mapped into a translation, constrained to the plane that contains the face.

6.1.2 Trackball

The trackball dragger is comprised of six sensitive parts: the sphere surface, three orthogonal rings, a handle, and the handle vertex.

This manipulator uses a variation of Evans' *trackball* and Chen's *virtual sphere*. Two consecutive positions of a point on the handle are mapped into a displacement along the direction of the handle. Two consecutive positions of the handle vertex are mapped into a uniform scaling operation. Two points on the sphere are translated into a rotation angle, relative to the axis perpendicular to the plane defined by those points and the sphere center. Finally, two consecutive positions of a point on a ring are mapped into the rotation angle about an axis perpendicular to the plane containing that ring.

6.1.3 Jack

The jack dragger is made of three intercepting orthogonal axes, and interaction is possible on its six edges and seven vertices.

The semantics of this manipulator is based on the work of Emmerik, allowing translation, rotation, and scaling operations. Two consecutive positions of a point on an edge are mapped into a translation, constrained to the line that supports that edge. Two consecutive positions of a vertex, in the same direction of its containing edge, are mapped into a scaling factor. Two consecutive positions of a vertex, in a direction parallel to one of the two remaining orthogonal edges, result in a rotation about the edge that was neither selected nor parallel. For instance, dragging one vertex in the x edge in a direction parallel to the y edge causes a rotation about the z edge. The vertex at the center was not used in our implementation, as it could eventually be hidden by the object the jack was controlling.

6.2 A Simple Application

A simple interactive 3D application was built on top of the toolkit. The application contains a basic geometric model, that can handle an arbitrary number of triangles. Each triangle has a unique identifier and can have its geometry transformed by using the manipulators described in section 6.1.

The application user interface was implemented using the XMotif library, providing additional functionalities, like selecting all existing triangles and removing the currently selected ones. The application also supports multiple cameras.

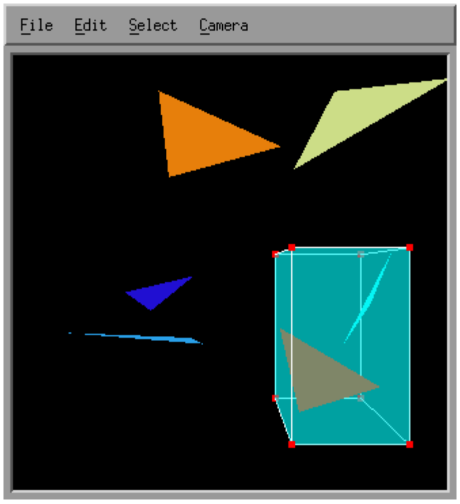

Figure 6: Test application.

As an example of the manipulations available, consider the scene depicted in Fig. 6, which contains six triangles. Each manipulator is responsible for a pair of them. Because the two triangles associated with the bounding box were previously selected, the corresponding dragger is made active.

When we apply a translation followed by a rotation to the bounding box, the associated objects are also affected by these transformations (Fig. 7). Note that one of the triangles associated with the jack manipulator was also selected, since a jack dragger is visible.

Next, we modified the scaling factor of the previous triangle by dragging one of its vertices. Then, we selected one of the triangles associated with the trackball and unselected the ones associated with the bounding box (see Fig. 8).

Finally, we rotated and changed the scale of the triangle associated with the trackball, manipulating both the handle and the sphere surface. The result is given in Fig. 9.

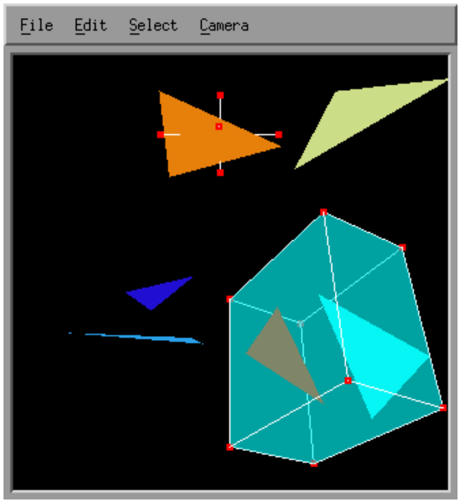

Figure 7: Rotation and translation made with a bounding box.

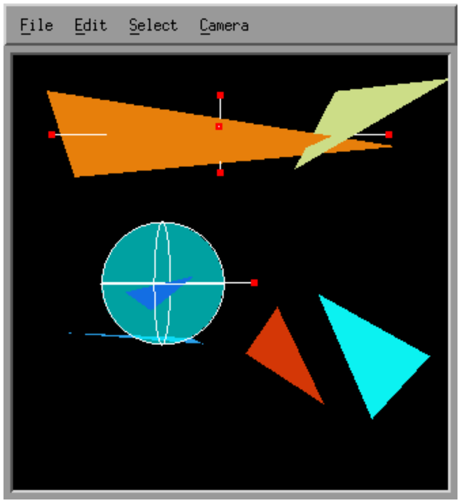

Figure 8: Non-uniform scaling done with a jack.

7 Conclusions

We have successfully developed a toolkit for 3D visualization and interaction. With this toolkit, we demonstrated the feasibility of designing and implementing a library with basic and context-free interaction tasks.

In section 6, we built some useful manipulators on top of MTK, and used them in a simple geometric modeler. Our implementation experiences let us conclude that the proposed toolkit is powerful and easy to use.

As future work, we intend to include additional features, such as other types of constraints and more specialized draggers. We also plan to develop more sophisticated manipulators and use the toolkit in larger applications.

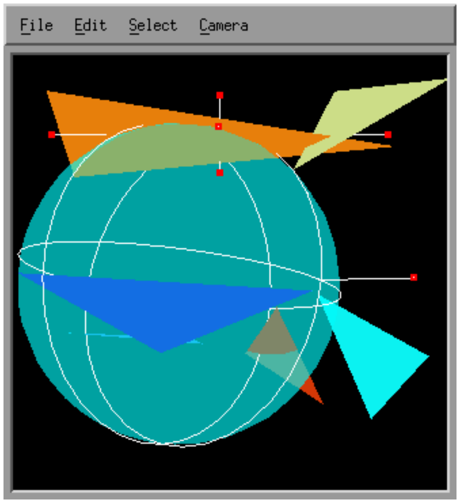

Figure 9: Rotation and scaling made with a trackball.

Acknowledgments

We would like to thank FAPESP for the financial support given to this work, under grant 96/0962-0. We also thank CNPq for the scholarship of one of the authors.

References

[1] International standard information, processing systems - computer graphics - graphical kernel system for three dimensions (gks-3d) functional description. American National Standards Institute, New York, 1988. ISO Document Number 8805:1988(E).

[2] Eric A. Bier. Skitters and jacks: Interactive 3d positioning tools. In *Proceedings 1986 ACM Workshop on Interactive 3D Graphics*, pages 183–196, Chapel Hill, North Carolina, New York, october 1986.

[3] Eric A. Bier. Snap-dragging in three dimensions. In Rich Riesenfeld and Carlo Sequin, editors, *Proceedings of the 1990 Symposium on Interactive 3D Graphics*, volume 24, pages 193–204, Snowbird, Utah, 25-28 March, 1990, march 1990.

[4] B. Castier, L. F. Martha, and M. Gattass. A taxonomy for interactive manipulation and visualization of 3d objects. In *Sibgrapi94*, pages 149–156, 1994. In Portuguese.

[5] W. Celes and J. Corson-Rikert. Act: an easy-to-use and dinamically extensible 3d graphics library. In *Sibgrapi97*, pages 26–33, Campos de Jordão, 13–16 October, 1997, october 1997. SBC, IEEE Computer Society.

[6] Michael Chen and Joy Mountford. A study in interactive 3d rotation using 2d control devices. In

Proceedings of ACM Siggraph'88, pages 121–129. Addison-Wesley, 1988.

[7] M. van Emmerik. A direct manipulation technique for specifying 3D object transformations with a 2D input device. *Computer Graphics Forum*, (9):355–361, 1990.

[8] Kenneth Evans B., Peter P. Tanner, and Marceli Wein. Tablet-based valuators that provide one, two, or three degrees of freedom. In *Proceedings of ACM Siggraph'81*, volume 15, pages 91–97. Addison-Wesley, august 1981.

[9] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 2nd. edition, 1990.

[10] K. P. Herndon, R. C. Zeleznik, D. C. Robbins, D. B. Conner, S. S. Snibbe, and A. van Dam. Interactive shadows. In *Proceedings of the ACM UIST'92 Symposium on User Interface Software and Technology*, pages 1–6, California, november 1992.

[11] W. T. Hewitt. Phigs: Programmer's hierarchical interactive graphics system. *Computer Graphics Forum*, 3(4):299–300, december 1984.

[12] R. Ierusalimschy, L. H. Figueiredo, and W. Celes. Lua: an extensible extension language. *Software: Practice & Experience*, 26(6):635–652, june 1996.

[13] Lutz Kettner. Theoretical foundations of 3d-metaphors. In *Workshop on the Challenges of 3D Interaction at the CHI'94*, february 1994.

[14] Fritz Loseries. Functional interface of the IQL frame type of PRODIA/11. Technical report, IGD, FhG, 1990.

[15] Jackie Neider, Tom Davis, and Mason Woo. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, release 1*. Addison-Wesely, 1993. ISBN 0-201-63274-8.

[16] G. M. Nielson and D.R. Olsen Jr. Direct manipulation techniques of 3D objects using 2D locator devices. In *Proceedings 1986 Workshop on Interactive 3D Graphics*, pages 175–182, Chapel Hill, North Carolina, New York, october 1986.

[17] William J. Schroeder, Kenneth M. Martin, and William E. Lorensen. The design and implementation of an object-oriented toolkit for 3d graphics and visualization. In *Visualization'96*, pages 93–100, november 1996.

[18] Paul S. Strauss and Rikk Carey. An object-oriented 3d graphics toolkit. In *Computer Graphics*, volume 26, pages 341–349, july 1992.

[19] Délia Perla Patrícia Velásquez Alegre. Basic techniques for 3d interaction with a mouse. Master's thesis, FEEC – Unicamp, Campinas, october 1995. In Portuguese.

[20] Josie Wernecke. *The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor, release 2*. Addison-Wesley Publishing Company, 1994. ISBN 0-201-62495-8.