3D Computational Steering with Parametrized Geometric Objects

Jurriaan D. Mulder

Centre for Mathematics and
Computer Science CWI
P.O. Box 94079, 1090 GB Amsterdam
The Netherlands
mullie@cwi.nl

Jarke J. van Wijk

Netherlands Energy Research
Foundation ECN
P.O. Box 1, 1755 ZG Petten
The Netherlands
vanwijk@ecn.nl

Abstract

Computational Steering is the ultimate goal of interactive simulation: researchers change parameters of their simulation and immediately receive feedback on the effect. We present a general and flexible graphics tool that is part of an environment for Computational Steering developed at CWI. It enables the researcher to interactively develop his own interface with the simulation. This interface is constructed with 3D Parametrized Geometric Objects. The properties of the objects are parametrized to output data and input parameters of the simulation. The objects visualize the output of the simulation while the researcher can steer the simulation by direct manipulation of the objects. Several applications of 3D Computational Steering are presented.

1 Introduction

Computational Steering can be considered as the ultimate goal of interactive computing. Computational Steering enables the researcher to change parameters of the simulation as the simulation is in progress, while viewing the simulation results simultaneously. Thus, the researcher can change variables on the fly, correct erroneous values for input, and, most important, the researcher can gain a large amount of additional insight if he can immediately observe the effect of changes in input parameters to dependent variables.

In recent years many methods, techniques, and packages have been developed for Scientific Visualization. However, most of these systems are limited to post-processing of data-sets, and therefore do not allow the user to interact with the simulation's data and input parameters. At CWI, a software environment for computational steering is being developed [15]. The aim is to provide the researcher with a general, flexible environment in which existing and newly developed scientific simulations can easily be incorporated to build custom computational steering applications.

In this paper, we present a 3D graphics tool (the *PGO editor*) which is part of this environment. The tool allows the user to directly interact with the simulation based on its visualization. The user can construct his own user interface for the visualization of the simulation's results and input parameters. This interface is built up with 3D Parametrized Geometric Objects (PGOs). The user can then change the values of the input parameters and state variables while the simulation is running by direct manipulation of these objects.

In section 2 we briefly describe the computational steering environment developed at CWI along with related work on computational steering environments, 3D visualization, and user interface creation. In section 3 the concept of 3D parametrized geometric objects is described. In section 4 the 3D interaction with these objects by direct manipulation is discussed. Some examples of the use of the graphics tool in computational steering applications are given in section 5, followed by a discussion and conclusion in section 6.

2 Related Work

2.1 Computational Steering Environments

Although many researchers have pointed out the importance of computational steering in next generation visualization systems (see for instance [4]), only few actual applications of visualization systems for steering have been developed. One example of such an application system is given by Marshall et al., who present a system for the visualization and simulation steering of a 3D turbulence model of Lake Erie [8, 16]. Examples of more general approaches can be found in [3] (the Visualization and Application Steering environment VASE), [5] (a rudimentary steering implementation with the use of AVS), and [6] (on the problems and advantages of the integration of scientific computations and visualization into one environment). However, no general applicable computational steering environment has yet evolved from these developments.

2.2 The Computational Steering Environment at CWI

The Computational Steering Environment which is being developed at CWI is based on two major concepts: A Data Manager which takes care of centralized data storage and event notification, and other processes called *satellites*, which can connect to and communicate with the Data Manager by the use of a 'publish and subscribe' paradigm, see figure 1.

(See color plates, page CP-36)

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

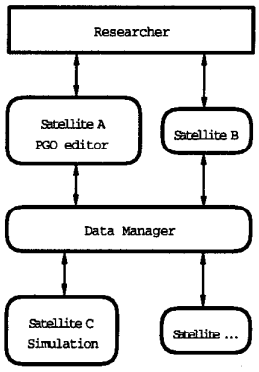

Figure 1: Architecture of the Computational Steering Environment.

The Data Manager

The central process in the CSE is the Data Manager. Other processes such as the simulation and visualization modules, can connect to and communicate with the Data Manager. The purpose of the Data Manager is twofold. A database of variables is managed, and it takes care of event notification. Satellites can create and read/write variables, and they can subscribe to events, such as notification of mutations of a particular variable. Thus, the Data Manager enables the different satellites to use the same data and to communicate with other satellites.

Satellites

Different kinds of processes can be connected to the Data Manager and thereby become satellites. The connection of satellites to the Data Manager is easily achieved with the use of a small Application Programmers Interface (API). If we connect a simulation to the Data Manager such that the output data as well as the simulation's steerable items are stored in and retrieved from the Data Manager, we are able to visualize the output data of the running simulation, and in addition steer the simulation by the use of one or more visualization and data manipulation satellites. Such satellites retrieve and manipulate the simulation's data and parameters present in the Data Manager. Several general satellites have already been developed for this purpose, such as a 2D visualization and manipulation satellite (the 2D PGO editor), a logging satellite, a calculator satellite, and a slicing satellite.

2.3 3D Visualization and Interface Creation

Accurate visualizations and user interfaces are essential in computational steering applications. By allowing the researcher to construct his own visualization and interface with the simulation, they will be according to the researcher's wishes and demands, and can easily be adapted if the researcher's focus of interest changes.

In addition, the visualization and user interface satellite can be kept general applicable and therefore be used for different kinds of steering applications.

Building one's own custom 3D visualization and/or interface from geometric primitives is a topic of interest in both the visualization and user interface community. In [10] a tool called Glyphmaker is described, developed for data visualization/analysis. It allows users to build customized representations of multivariate data and provides interactive tools to explore the patterns in and relations between the data. With the provided primitives points, lines, spheres, cuboids, cylinders, cones, and arrows, the user can draw glyphs using the 3D Glyph Editor. Properties of the glyphs can be bound to data using the Glyph Binder. Raw (simulation) data is transcribed by the Read Module into Explorer data structures which are used by the Glyph Binder. These bindings however, are only uni-directional; only from the data to the glyphs. Therefore, Glyphmaker does not allow the user to steer the simulation by manipulating the geometric objects.

In [11] an architecture for an extensible 3D interface toolkit is presented. The toolkit can be used for construction and rapid prototyping of 3D interfaces, interactive illustrations, and 3D widgets. By direct manipulation of 3D primitives through a visual language, widgets, interface objects, and application objects are constructed whose geometry is affinely constrained. The four basic primitives of the toolkit are: the point primitive, the vector primitive, the plane primitive, and the graphical object primitive. Although the system does allow a high degree of direct manipulation to construct an interface, there remain several operations that have to be performed in an indirect manner, such as the definition of constraints between objects in a separate window with no visual feedback from the objects themselves.

Our goal when developing the 3D PGO editor was to provide a tool with the following properties:

- It has to provide accurate visual feedback of all the concepts used for the construction and editing of the visualization and interface;
- It has to provide bi-directional bindings between the objects' parameters and the simulation's parameters and state variables;
- It has to allow direct manipulation on the objects. Both when the visualization and interface are created and edited, as when the objects to steer the simulation are manipulated;
- It has to be simple and easy to use, yet remain effective and powerful.

3 3D Parametrized Geometric Objects

The 3D graphics editor uses Parametrized Geometric Objects (PGOs) for the visualization of the simulation results and the user interface. Properties of simple basic objects such as spheres, cylinders, and boxes, can be parametrized to values of variables in the Data Manager. By changing the value of a variable in the Data Manager, the objects parametrized to this variable change their visual appearance and, in the opposite direction, by manipulating an object the values of the variables to which the object is parametrized are changed in the Data Manager. Thus, there is a bi-directional symmetrical binding for input and output between the objects and the simulation's parameters.

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

The graphics editor has two modes: specification and application, or *edit* and *run*. In edit-mode, the user can create or edit geometric objects and parametrize properties of the objects to values of variables in the Data Manager. Hence, the researcher draws a specification of the interface. In run-mode, a two-way communication is established between the graphics tool and the simulation by binding these properties to variables. Data is retrieved and mapped onto the properties of the geometric objects. The researcher can enter text, pick and drag objects, which is translated into changes of the values of variables.

3.1 Point-Based PGOs

We considered three different approaches to define 3D parametrized geometric objects (see figure 2):

1. Based on definitions per object;
2. Based on local coordinate frames;
3. Point-based.

With the first approach, for each type of object a separate set of parameters is defined. Each object has a dialogue-box associated with it in which the actual values of those parameters can be defined. This approach has been used in Glyphmaker [10]. With the second approach, each object is assigned a local coordinate frame which is used to define the object's position, orientation, and size. With the third approach, the objects are defined by two or more *control-points* that determine the object's position, orientation and size. For instance, a sphere can be defined by the use of two points: one for the center of the sphere, defining the sphere's position, and one on the surface, defining the sphere's radius. A cylinder can be defined with three points: two for the center-axis of the cylinder, defining its position, orientation, and length, and one for its radius.

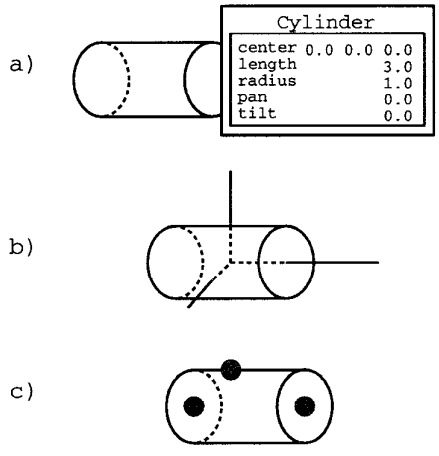

Figure 2: Different definitions for parametrized geometric objects: per object (a), frame based (b), and point based (c).

A point in 3D space has three degrees of freedom. Therefore, the total number of degrees of freedom for the set of points used to define an object will be a multiple of three. However, an object like a cylinder has only seven degrees of freedom. In such a case, not all the degrees of freedom of the points that define the object are used. For the cylinder one point is used only for a single parameter: the radius. Nevertheless, the point-based approach has a number of advantages over the other two:

- It is uniform. All geometric objects can be defined in a similar manner;
- It is general applicable. Objects like polygons and polylines can easily be defined and parametrized;
- It is easy to use. No knowledge about (local) coordinate frames is required;
- It allows for easy interaction through direct manipulation. Object positions, orientations, and sizes can be changed by manipulation of the control-points;
- Several relations between different objects can be easily defined. By sharing control-points among objects, constraints like keeping different objects at the same position, or letting objects' surfaces touch can be enforced;
- It is flexible. Different object parametrizations can be achieved by changing the parametrizations of the control-points.

The point-based approach was also used in a previously developed 2D editor and with good results. For these reasons we have chosen the point-based approach to define the objects in the 3D PGO editor.

A set of standard geometric objects is provided by the PGO editor: polyline, fill-area, box, sphere, cylinder, cone, and text. The geometry of these objects is defined by their control-points. These points are independent of the geometric objects themselves, so that one point can be shared by various geometric objects. The objects can be displayed as solids or as wire-frames, and have four additional attributes: the hue, saturation, and value (intensity) of its color, and the linewidth used for the object or its wire-frame. These attributes can also be parametrized to values of variables in the Data Manager and are presented to the user via a separate attribute window. In text objects references to values of variables in the Data Manager are replaced in run-mode by the value of the corresponding variable. Figure 3 shows some examples of the standard geometric objects with their control-points.

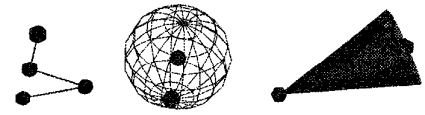

Figure 3: A polyline, sphere, and cone with their control-points.

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

3.2 Inter-Point Connections

The researcher can define relationships between points and thereby enforce constraints on the objects. Any point can have another point as a reference-point or parent-point. These relations are shown as grey lines with yellow arrows, and can be accomplished by simple dragging one point's 'connector' (a red cone) to another point. Cycles are not allowed, thus, the structure is a forest of trees, with points as nodes and leaves. These relations are used when points are moved. How this is done depends on the type of the point. Two types of points can be used:

- Hinge points (depicted as spheres). When a hinge point is moved, the same translation is applied to all its child-points. So, hinge points are suited for the translation of a set of points;
- Fixed points (depicted as diamonds). At a fixed point the angles between the connections that start or end at this point are constrained to be fixed. They can be used if a set of points is to be rotated around some other point (the grandparent-point).

Next these transformations are recursively applied to the children of the transformed points. Besides moving points, the user can also move objects. If an object is picked and moved, all the object's control-points are moved with it.

3.3 Point Parametrization

The position of the points can be parametrized to values of variables in the Data Manager by the use of Degrees of Freedom (DOFs). Each DOF has a minimum, a maximum, a current value, and possibly a Mapped Variable that is bound to the DOF. A DOF is presented to the user as a line, bounded by two discs. The line represents the range of allowed positions for the control-point. An arrow-head indicates the direction of the DOF. In edit-mode all aspects of the DOFs can be changed interactively via dragging and text-editing, in run-mode only the current value can be changed. Since we are working in 3D, three DOFs per point can be used. Two types of DOFs are available for the parametrization of a point: Cartesian DOFs in x, y, z, or any combination of these, and spherical DOFS in *radius*, *azimuth*, *elevation*, or any combination of these, see figure 4. The reference frame for the Cartesian DOFs is the world coordinate system, i.e. the x, y, and z DOFs are aligned with the X, Y, and Z-axis of the world coordinate system. The reference for the sphere DOFs is provided by the parent and grandparent-point: the azimuth DOF lies in the plane perpendicular to the line defined by the parent and grandparent point, and the elevation DOF is perpendicular to this plane. If the point does not have a grandparent-point then the azimuth DOF is parallel, and the elevation DOF perpendicular to the XY-plane of the world coordinate frame. It is required that the point has a parent-point for the use of spherical DOFs.

With these options in combination with the relations between points a wide variety of coordinate frames can be defined: relative, absolute, Cartesian, spherical, cylindrical, hierarchical, although the concept of a coordinate frame itself is not provided. This is a typical example of the main principle that has guided us: provide a set of simple primitives and useful operations to combine them, so that the user can easily construct a wide range of higher order concepts.

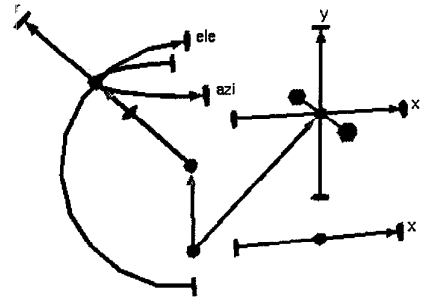

Figure 4: Visual representations of Degrees of Freedom for points.

3.4 Mapped Variables

The name that a researcher can specify for a DOF refers to a Mapped Variable (MVAR). The data of a MVAR are references to a Map and a Variable, and two on/off switches for input and output. With these switches the researcher can select if input information (from the researcher to the Data Manager) and output information (from the Data Manager to the researcher) can pass or not.

The Map contains a specification on how values must be mapped in the communication with the Data Manager. The current implementation is simple: only linear mappings are supported, hence the specification of a minimum and maximum value suffices. Furthermore, the Map contains a specification of the format for the textual output. A Variable is a local copy of the information in the Data Manager. Here, some bookkeeping information, such as type, size, name and Data Manager id, and the current value(s) are stored. Several Variables can share the same Maps and one variable can have several associated mappings.

From all geometric objects multiple instances can be generated by mapping the DOFs of the control-points or the attributes of the object to arrays present in the Data Manager. The number of instances depends on the size of the arrays which may vary dynamically during run-mode. In addition, polylines and polygons can be expanded inline within a single object. This means that instead of creating multiple instances of the object, the number of control-points defining this object is set according to the array size.

4 3D Interaction

The editing of the user-interface and the manipulation of the graphical objects and points requires a 3D interaction technique. We need to be able to position the control-points in 3D space, assign DOFs to these points, make connections between them, define geometric objects on them, and reposition the points and objects in both edit-mode and run-mode. We wanted a technique that

- does not require any special devices, only a traditional display, 2D mouse, and a keyboard;
- allows for direct manipulation on the points and objects in the 3D space;

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

- allows for precise positioning in the 3D space;
- is easy to learn and work with.

In order to achieve this, we adapted techniques from [14] and [9] (a 3D crosshair cursor), and [2] (Interactive Shadows).

4.1 3D Crosshair Cursor

In [14] a method is presented for geometric transformations of objects in 3D space through direct manipulation on a 3D crosshair. The transformations provided are translation, scaling, and rotation. Since in our system scaling and rotating of objects is achieved by repositioning the objects' control-points, we restricted the technique to only positioning and translating points and objects in the 3D space.

Whenever a positioning or translating task is to be performed, a 3D crosshair cursor appears in the 3D space. The user can then pick the crosshair with the 2D mouse cursor and drag it to a new position. The effect of the 2D mouse movement on the crosshair depends on the direction of the movement and the orientation of the projected crosshair. For instance, if the 2D movement is along the projection of the y-axis of the crosshair, the crosshair will be moved along its y-axis. The selection of the axis along which the crosshair is to be moved is accomplished by matching the mouse cursor movement vector to the projections of all three axis of the crosshair. The axis with the best (directional) match is selected as the axis of translation. To avoid undesired movements along axes that are (almost) perpendicular to the screen, and thus have a projection that can hardly be seen, a threshold is used to allow only translations along those axes whose projections have a certain minimum length.

Additional features for (exact) positioning of the crosshair are a user defined grid-snap option, numerical feedback of the position of the crosshair as well as the possibility to numerically specify the crosshair's position, and the option to disable translations along one or more axes of the crosshair.

4.2 Shadow Editing

The virtual workspace in which the 3D visualization and interface are created is a 3D box with user-defined dimensions. The user can rotate the box, translate the viewpoint, and zoom in or out by moving the mouse over an additional box icon located in the projection window. This leaves the mouse to be used for manipulation on the objects in the workspace.

We adapted a technique from [2] to provide the user with an additional method to edit and manipulate the geometric objects and points. The user has the option to display the bounding planes of the workspace that face the user from the inside. On these planes orthogonal projections (*shadows*) of the objects in the workspace (the geometric objects, the control-points, the connections between the control-points, the 3D crosshair cursor etc.) can be displayed. Now the user cannot only interact with the 3D objects themselves, but also with their (2D) shadows.

For instance, if a point is to be repositioned, the user can pick a shadow of the point in one of the projection planes. Then the 3D crosshair appears at the position of the point in the workspace along with its projections at the positions of the shadows of the point to move. Now the movements of the mouse-cursor are mapped to the screen projections of the shadow of the crosshair and the appropriate axis of movement is selected.

These interactive shadows provide several benefits:

- They offer the user three additional representations of the 3D scene in one coherent display.
- It is possible to pick objects or points which are obstructed by other objects or points in the 3D scene. To simplify the editing of points, picking points precedes picking objects in the projection planes (all geometric objects' shadows are displayed in a grey color; point-shadows are superimposed in a darker color);
- The user can easily perform translations limited to a plane aligned with the world coordinate frame.
- It allows for the snap-on grid to be displayed. As there is no feasible technique to display a 3D grid, the projection planes can be used to display 2D projections of the grid;
- It serves as an additional depth cue.

4.3 Object Creation and Manipulation

Figure 5 depicts the specification of an arrow. Such an arrow could for instance be used to steer a field force in a simulation. Its length would then be parametrized to the strength of the force while its orientation would depict the direction of the force.

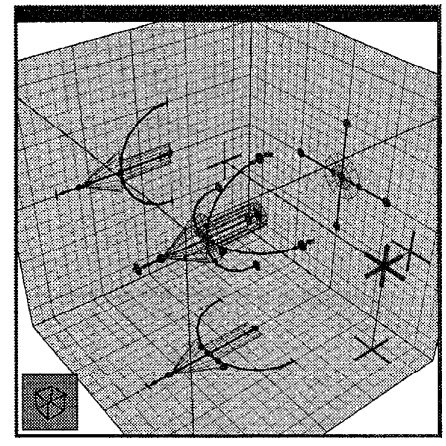

Figure 5: Arrow in edit-mode.

The arrow consists of two primitives: a cylinder and a cone. The cylinder is specified using three control-points: one for its base, one for its top, and one for its radius. The surface control-point is connected to the top control-point which in term is connected to the base-point. The cone is also specified by three control-points: one for the bottom, one for the tip, and one for the bottom radius. The cone's bottom-point is the same point as the cylinder's top-point, as to keep the cone on top of the cylinder. The cone's tip-point and radius-point are connected to its bottom-point (i.e. the cylinder's top-point). Now we can move the entire

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

arrow in edit-mode by moving the cylinder or the cylinder's base-point.

The cylinder's top-point is given three DOFs: azimuth, elevation, and radius. These DOFs can be parametrized to variables in the data base, such that the cylinder points in the direction of the force and its size depicts the strength of the force. By making the cylinder's top-point a fixed point, we assure that the cylinder's diameter-point and the cone's top-point and diameter-point keep the correct orientation and distance in regard to the fixed point. This way the arrow will not change it's diameter and will point in the correct direction.

A composite object like an arrow or a slider is useful for many applications. We can save such objects as macro's, and reuse them in their original form or tailored to the particular application.

Figure 6 depicts six instances of the specified arrow in run-mode. The DOFs elevation, azimuth, and radius were mapped to arrays of length 6 in the Data Manager. The hue attribute of the cylinder was mapped to its length. Each arrow can be picked separately and adjusted to a new length and orientation, upon which the values in the arrays at the appropriate index are updated.

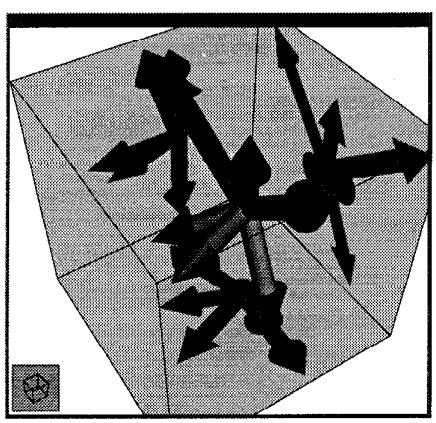

Figure 6: Multiple instances of the arrow in run-mode.

5 Applications

5.1 Path-planning

Figure 7 shows the interface to a path planning application developed by K. Trovato [13] and L. Dorst et al. [1][1]. The interface shows two representations of a car parking problem. One is the task space which visualizes the street, the car, and the two obstacles in between which the car is to be parked by the path planning program. The other is the configuration space (c-space). Here, the three parameters that describe the configuration of the car are visualized: two position parameters x and y, and the car's orientation ϕ. A hole in the representation indicates that the car can take that parameter configuration without interference with the two obstacles. The c-space is cyclic, and the user can examine the c-space by manipulating two boundary planes in its representation to select the region to be visualized.

The car can be dragged to a new initial or goal position by manipulating the car itself or its representation in the configuration space (a small sphere). Also, the two obstacles can be resized by direct manipulation. When the car is moved, by the user or the path planning program, the traveled path of the car is logged with the use of a general logging satellite. This log is visualized with wire frame projections of the car in the task space, and with a polyline in the configuration space.

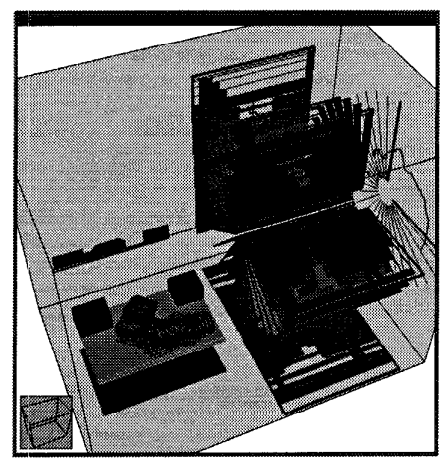

Figure 7: Interface to a path planning application.

Figure 8 depicts a robot arm. The robot arm comprises several different rotational and translational joints. Although this interface is not yet connected to a simulation, it illustrates the effectiveness of the 3D PGO editor in constructing compound objects with complex constrained relations between the different components. The user can control the robot arm by the use of the sliders, where each slider manipulates one joint, or by manipulation on the robot arm itself. We plan to develop similar robot arm interfaces in combination with path planning algorithms.

5.2 Cars Database

Figure 9 shows that the PGO editor and the Computational Steering Environment can be used for multi-dimensional visualization, and as a front-end to databases. We used the PGO editor to visualize a table of 400 different types of cars, where for each car a number of attributes such as displacement, horsepower, and acceleration are given[2]. We used a small satellite that reads in the data and writes these to the Data Manager, and that can be used to make selections of the data. Each entry in the database is represented by a small car in the 3D space. The color of

[1]The path planning software is ©by Philips Laboratories, 1988. Philips has four patents pending relating to the vehicle planning methods and control.

[2]The data set has been provided by the Committee on Statistical Graphics of the American Statistical Association for its Second (1983) Exposition of Statistical Graphics Technology.

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

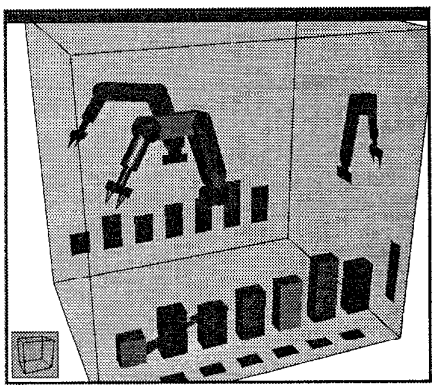

Figure 8: Robot arm.

the cars depict their origin (blue: USA, green: Europe, and red: Japan) and the positions of the cars is currently parametrized to three properties: acceleration, displacement, and horsepower. The user can select a region of interest in the 3D space by manipulating a bounding box, and he can select a car whose particular properties are then displayed.

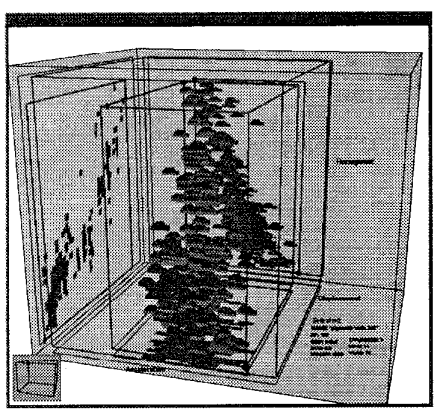

Figure 9: A front-end to a multi-dimensional database.

5.3 Vector Field Visualization

We have also used our environment for vector field visualization. A small satellite was implemented that calculates the motion of a particle, advected by a vector field. A standard logging satellite was used to store the history of the calculated positions. Figure 10 shows the well-known Lorenz attractor, the result when the Lorenz-equations [7] are used for the vector field. The user can drag the particle to different positions, and it is very fascinating to observe how after a number of time-steps the particle cycles again around the attractors. The implementation of this user interface took less then half an hour.

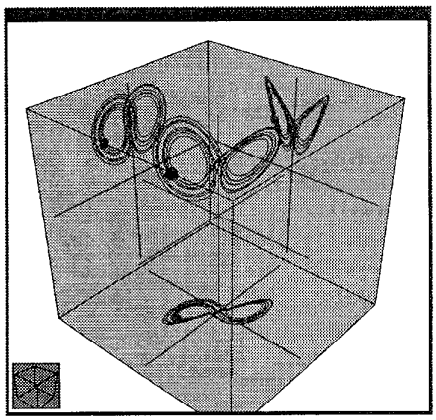

Figure 10: Particle tracing.

5.4 Bouncing Balls

Figure 11 shows an interface that was constructed for the steering of a simulation of bouncing balls. Numerous steerable parameters are present in the simulation. In this particular interface, the user can steer the dampening property of the medium in which the balls move, the number of balls, and the radius of the balls by the use of sliders. The user can also steer a field force by manipulating the arrow in the small box. The direction of the arrow depicts the direction of the force, while its length represents the strength of the force. Furthermore, the user can pick and drag one of the balls to a new position. The balls are visualized with an additional arrow to depict their velocity vector.

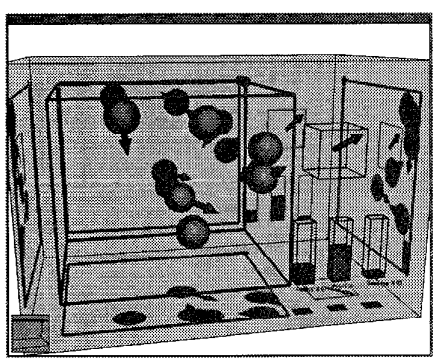

Figure 11: Simulation of bouncing balls.

6 Conclusion

We have shown how 3D parametrized geometric objects can be used in a tool for computational steering. The objects are used for both the visualization and the steering of the simulation. The user can easily create his own interface with the simulation by constructing glyphs and input/output widgets from the geometric primitives provided by the editor. Properties of these primitives can be bound

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

to variables in the central Data Manager. This can be a bi-directional binding: from the Data Manager towards the editor (whenever the data in the Data Manager changes, the visual representation changes), and from the editor towards the Data Manager (if the visual representation is changed, the data in the Data Manager is changed).

The main difference between the 3D PGO editor presented in this paper and the Glyphmaker presented in [10] is that the Glyphmaker does not allow for computational steering; the bindings between the data and the glyphs are uni-directional. Other differences are the point-based object definitions in the PGO editor versus the more ad-hoc based approach in Glyphmaker, and the lack of direct manipulation techniques on the objects in Glyph Editor. Although the method of defining constraints as used in the toolkit presented in [11] allows for more complex and perhaps more powerful constraint relations between objects than the method used in the PGO editor, our method has some advantages over the other:

- It provides accurate visual feedback;
- The relations between points, the type of points, and the degrees of freedom of a point can all be changed by direct manipulation;
- It is very effective, yet remains simple and easy to use.

Superficially, PGOs have similarities with the primitives offered by IRIS Inventor [12]. However, they have a different scope: Inventor consists of tools and libraries that can be used to develop applications, whereas the system described here provides an integrated environment for end-users.

The PGO editor can easily be extended with other basic primitives. The current primitives are discrete objects. It is an interesting challenge if new primitives can be developed for continuous surface and volume data. Furthermore, the PGO editor is very suited to develop more complicated visualizations (such as glyphs) by combining simpler ones. In addition, special complex manipulators could be developed and evaluated for the control of complex data input configurations.

Acknowledgements

We would like to thank Karen Trovato of Philips Laboratories and Leo Dorst of the University of Amsterdam for providing the path planning software, Robert van Liere of CWI for his help on the implementation of the work described here, and Frans Groen of the University of Amsterdam for his valuable comments on the draft versions of this paper.

References

[1] L. Dorst, I. Mandhyan, and K. Trovato. The geometrical representation of path planning problems. *Robotics and Autonomous Systems*, 7:181–195, 1991.

[2] K.P. Herndon, R.C. Zeleznik, D.C. Robbins, D. Brookshire Conner, S.S. Snibbe, and A. van Dam. Interactive shadows. In *Proceedings UIST '92*, pages 1–6, November 1992.

[3] D.J. Jablonowski, J.D. Bruner, B. Bliss, and R.B. Haber. Vase: The visualization and application steering environment. In *Proceedings of Supercomputing '93*, pages 560–569, 1993.

[4] M. Jern and R.A. Earnshaw. Interactive real-time visualization systems using a virtual reality paradigm. In M. Göbel, H. Müller, and B. Urban, editors, *Visualization in Scientific Computing*, pages 174–189. Springer-Verlag Wien New York, 1995.

[5] G.D. Kerlick and E. Kirby. Towards interactive steering, visualization and animation of unsteady finite element simulations. In *Proceedings of the Visualization '93 Conference*, pages 374–377, 1993.

[6] U. Lang, R. Lang, and R. Rühle. Integration of visualization and scientific calculation in a software system. In *Proceedings of the Visualization '91 Conference*, pages 268–274, October 1991.

[7] E. Lorenz. Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, 20:130–141, 1963.

[8] R. Marshall, J. Kempf, S. Dyer, and C.-C. Yen. Visualization methods and simulation steering for a 3D turbulence model of Lake Erie. *Computer Graphics*, 24(2):89–97, 1990.

[9] G.M. Nielson and D.R. Olsen. Direct manipulation techniques for 3D objects using 2D locator devices. In *Proceedings of the 1986 Workshop on Interactive 3D Graphics*, pages 175–182, 1986.

[10] W. Ribarsky, E. Ayers, J. Eble, and S. Mukherjea. Glyphmaker: Creating customized visualizations of complex data. *IEEE Computer*, 27(4):57–64, July 1994.

[11] M.P. Stevens, R.C. Zeleznik, and J.F. Hughes. An architecture for an extensible 3D interface toolkit. In *Proceedings of the UIST '94 Conference*, pages 59–67, November 1994.

[12] P.S. Strauss and R. Carey. An object-oriented 3D graphics toolkit. *Computer Graphics*, 26(2):341–349, July 1992. Proceedings SIGGRAPH '92.

[13] K. Trovato. Autonomous vehicle maneuvering. In *Proceedings SPIE Volume 1613*, pages 68–79, November 1991.

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

[15] J.J. van Wijk and R. van Liere. An environment for computational steering. Technical Report CS-R9448, Centre for Mathematics and Computer Science (CWI), 1994. Presented at the Dagstuhl Seminar on Scientific Visualization, 23-27 May 1994, Germany, proceedings to be published.

[16] C.-C.J. Yen, K.W. Bedford, J.L. Kempf, and R.E. Marshall. A three-dimensional/stereoscopic display and model control system for great lakes forecasts. In A. Kaufman, editor, *Proceedings of the Visualization '90 Conference*, pages 194–201, October 1990.

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE