Advanced Interfaces for Virtual Environments

Paul Brewster
NASA Langley Research Center
Paul.F.Brewster@nasa.gov

Abstract

This paper describes a framework for implementing advanced interfaces for virtual environments. The framework is robust, modular, and extensible so that it can be fine-tuned to take full advantage of given hardware systems. The benefit of this framework is that large, complex virtual environments with a high degree of interaction can be rapidly implemented because the developers only need to spend time and effort on application and domain specific areas. An example of a high-level intelligent interface based on a multi-agent, or swarming, paradigm built on top of the framework is also described.

1. Introduction

Immersive virtual reality environments have been shown to offer many advantages in engineering applications such as in prototype design, molecular dynamics, and oil exploration. By immersion into a fully 3-D virtual environment, the users can gain a much greater, more comprehensive understanding of the application. The user is also able to interact with the environment in a more realistic and intuitive way. Although the concept of virtual reality has been around for over 35 years, only recently has the computational and display hardware advanced to the point where widespread adoption of this technology in engineering applications is feasible. In particular, consumer-level hardware-accelerated 3-D graphics cards, inexpensive head-mounted displays (HMD), and large-scale projection theaters (i.e. CAVE, PowerWall, etc.) are commercial of the shelf purchases, allowing virtual environments to be implemented easily on the desktop or in small group environments.

The software to support virtual environments, however, has not matured as fast as the hardware. Current software consists of low-level programming APIs, desktop applications extended to work on virtual reality hardware, or custom in-house programs. Programming APIs such as CAVELib, VR-Juggler, DIVERSE, and WorldToolKit provide the basic support for developing virtual environments by automating display configuration and accessing the input devices. Extended desktop applications, such as EnSight and Division, provide only a subset of its capabilities on VR devices. The full functionality of the application is still only available using the standard desktop interface. Custom in-house programs are often built on top of a low-level API and provide the most functionality and performance, but require a great deal of time and effort to develop.

The common limitation on all three of these software applications is the lack of a widespread, standardized user interface. Low-level APIs do not implement a high-level user interface, extended desktop programs still use the desktop user interface for most interactions, and custom applications require effort to develop a new interface for each application. Although Graphical User Interfaces (GUIs) are well-developed and standardized for 2-D desktops, there is no well-known interface in virtual environments.

This paper describes an object-oriented, event-driven interaction framework for building advanced user interfaces designed explicitly to take advantage of the extra capabilities of a virtual environment. The framework would serve as a higher-level programming API so that new applications could be created quickly and easily by implementing a standardized user interface. A standardized user interface would not only save development time, but would also lead to quicker adoption, since the user would not have to learn a new interface style or convention for each application.

The primary design goal of the framework is to provide a simple, natural interface that could be used on a wide variety of applications. In particular, the interface should not distract the user, nor disrupt the immersive environment. The goal of an immersive virtual environment is to help the user feel that they are in the simulated environment, so any interface element that hinders this illusion of reality diminishes the effectiveness of the environment.

Three methods of interaction were developed to provide an intuitive interface without disrupting the immersive nature of the environment: a system of movable, interactable objects, or draggers, voice recognition, and a multi-agent system built on top of the dragger and recognition interaction methods. The dragger system provides a modular, extensible method of interacting with the environment by selecting and manipulating individual objects in the environment. The behavior of the environment is determined by defining which objects are selectable and the behavior of the selected objects during manipulation. Voice recognition provides further control of the environment by allowing the user to interact in ways that are not easily mapped to a dragger-type system. The dragger and voice recognition system together provide a natural, transparent interface that is also fully functional. The multi-agent system, built on top of the dragger and recognition system, provides a higher-level interface where the user can quickly search a data set to find interesting

features. The user instructs a large collection of agents, or swarm, to find a given feature by giving a set of constraints that the agents must follow. By satisfying the constraints the agents extract the relevant feature.

The framework was designed to support a wide range of possible applications of virtual environments. This paper will center specifically on an application for visualizing Computational Fluid Dynamics (CFD) simulations, although the framework has been used in other applications such as Mars mission planning, ISS radiation studies, and molecular dynamics simulations.

2. Related Work

Several common programming APIs for virtual environments, such as CAVELib and VR-Juggler, are available from industry and academia. These APIs are designed to handle the low-level aspects of VR, such as managing the display devices and sampling the inputs devices. They do not, however, implement an interaction framework.

Although not specifically designed for virtual environments, SGI's Open Inventor [10] was one of the first frameworks to include manipulatable objects, or draggers, in the scene. The draggers in Open Inventor inspired the dragger system described in this paper. The system presented here is much more flexible and robust than the draggers there. In particular, Inventor draggers are closely linked to specific transformation properties (i.e. translation, rotation, or scale); whereas, the system presented here can be used for more abstract concepts (i.e. time, density, length, speed, etc).

Bowman [1] gave a good overview of many of the concepts involved in developing a 3-D GUI for virtual environments. The discussion of interaction techniques in section 3 is based on their taxonomy.

Various object-manipulation frameworks have been proposed to support interaction in virtual environments [3][4][6][7][11]. Each one of these frameworks supports the manipulation of specific geometric objects in the scene to interactively translate, rotate, or scale elements. The dragger system described in this paper, however, facilitates the creation of new interactions that are not explicitly 3-D in nature. Further, this framework includes a voice recognition system to provide a complete interaction system that will not disrupt the 3-D immersive quality of the environment.

The multi-agent system described in this paper is based on the flocking work of Reynolds [8] and the smart particles work of Pang [5]. The primary difference being that each element is an agent that the user can interact with, rather than a particle that, once created, only obeys a given set of constraints. Here, the set of constraints can be arbitrarily manipulated, even while the agents are continuing to satisfy these constraints. Thus, the agents are searching for a given feature while, effectively, conversing with the user.

3. Framework

3.1. Orientation

The framework uses a modular, object-oriented, event-driven architecture for the development of new forms of interaction designed to explicitly take advantage of the special capabilities of a given I/O device. This framework implementation takes advantage of the extra interaction features provided by a CAVE or PowerWall system. The modular nature of the framework, however, provides a way to seamlessly adapt the interactions to work on other VR systems.

The hardware assumed in this discussion is based on a typical large-scale projection-based VR device, such as a CAVE or PowerWall. The display system is a fully stereo, head-tracked, 3-D system with one or more displays. The user interacts with the environment using a single, hand-tracked six-degree-of-freedom interaction device, or wand, with a minimum of two buttons and a 2-D joystick. A keyboard is not assumed to be available.

One of the goals in developing the framework was to create new forms of interactions appropriate to the 3-D nature of the environment, rather than to move traditional 2-D GUI objects into a virtual environment. As such, many common 2-D GUI elements were explicitly not included in the framework because they might disrupt the immersive quality of the environment. As the software mature, these elements could be included in the framework. For example, although a menu system is a well-known, functional 2-D GUI element, it was not included because its 2-D nature often disrupts the 3-D immersive quality of the virtual environment.

3.2. Draggers

The core of the interaction framework is the modular, event-driven draggers used to define the interaction points in the environment. A dragger is a geometric object embedded in the virtual environment, or scene, that can be interacted with and manipulated. As the user interacts with the dragger, events are passed back to the environment to communicate the interaction. Draggers are the only objects in the scene that can be interacted with. Draggers are independent objects composed of interdependent parts, where each part implements one aspect of the interaction. The function and behavior of the dragger is defined by how the parts are manipulated and how they interact with each other. For example, a dragger that defines a line segment in space may consist of one part to define the origin of the line, one part to define the direction of the line, and one part to define the length of the line. See Figure 1.

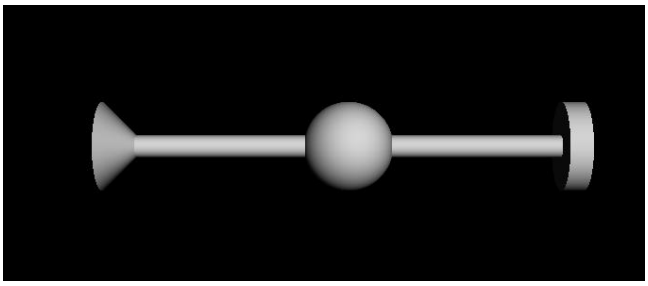

Figure 1 Line segment dragger consisting of three parts: a sphere to define the location, a cone to define the orientation, and a cylinder to define the length.

Each part in a dragger can be one of four possible types: a cyclic toggle, a 3-D location, a scalar value, or a 3-D vector value. Boundary constraints may be placed on each part to further specialize the interaction. For example, a dragger used to define a seed rake for creating streamlines in a CFD visualization may consist of one part that controls the location of the rake, one part that controls the vector orientation of the rake, one part that controls the scalar length of the rake, and one part that controls the scalar number of seeds on the rake. See Figure 2.

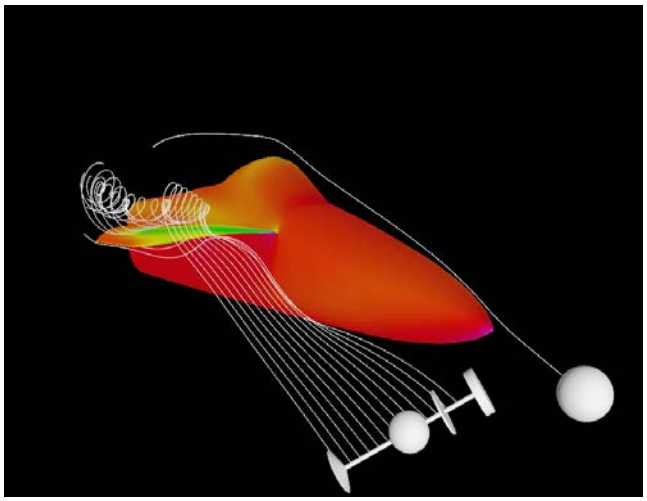

Figure 2 Two sets of streamlines originating from a seed dragger. The single streamline uses a simple 3-d location dragger to locate the seed. The streamrake uses a line segment dragger to locate the rake of seeds. The cylinder in the middle of the dragger is used to control the number of seeds. The streamlines show the flow of air around the CFD solution of the F-18 forebody.

The shape of a dragger is defined by both the interactions between the parts and the shape of the parts themselves. Each part can have an arbitrary, possibly changing shape. Further, the results of interacting with one part may change the shape of another part. Draggers may be added geometry to the scene or they may be actual parts of the scene. The streamline rake described above is added geometry to the scene, that is, the rake is only there to be interacted with. In an ISS radiation study, different hardware racks on the ISS may be draggers, allowing the user to interact with each rack [9]. In a steered molecular dynamics simulation, the whole scene may be one large dragger with each atom a 3-D location part.

Interaction in virtual environments is broken down into four areas: selection, manipulation, navigation, and control. The framework will be described by discussing each of these four areas independently.

3.3. Selection

Selection refers to the process of choosing an object in the scene for interaction. In the interaction framework, only draggers can be selected. Selection is done with a user-controlled pointer that is placed on top of objects. The pointer is a 1 inch round ball that lies 2 feet in front of the user's wand. This method is similar to the aperture-based method [2].

In order to select an object, a ray is cast from the users eye (user-definable to left or right) through the center of the pointer. The ray is tested for intersection with all draggers. If a dragger is hit, an event is passed to it indicating the selection. An unselect event is passed to the dragger when the ray no longer intersects with the dragger. Visually, selection occurs when the pointer lies on top of, or underneath, the dragger.

This form of selection works well for objects near the user, but is difficult for objects far away. To mitigate this, an unseen halo is placed around each dragger part. This halo is of the same shape, but twice the size of the part. If no interaction is found between the ray and all draggers, a second intersection is tested between the ray and the halo of all draggers.

3.4. Manipulation

Manipulation refers to the process of geometrically transforming a selected object in some way. In the interaction framework, the user manipulates objects in the scene by selecting and dragging parts of a dragger. The user selects a dragger by moving the pointer over the dragger. The user then manipulates the dragger by pressing the left button on the wand when a dragger is selected. Events are generated and passed to the dragger when the button is initially pressed, released, or while the button is held down. The selection mechanism is disabled between corresponding button press and release events so that the initially selected dragger will not be unselected in the middle of a button down event. Dragger parts may be manipulated in four different ways, corresponding to the four types of parts.

Toggle manipulations are the simplest form of manipulation. When receiving the button press event, the dragger cycles to the next state from two or more possible states and sends a state changed event to the environment.

The manipulation can be constrained by limiting the number of possible states.

3-D translation manipulations are used to locate a dragger in the scene. When receiving the button press event, the dragger initializes the manipulation by calculating the distance from the eye to the dragger. During the button down event the dragger is moved so that it always lies exactly under the pointer, but at the given distance. This method defines a sphere of possible locations that the dragger could be moved to. In order to move the dragger nearer or farther, the distance to the dragger is calculated in relation to the distance between the eye and the pointer. Therefore, moving the pointer away from the eye moves the dragger further away. Likewise moving the pointer closer to the eye moves the dragger closer. The manipulation can be constrained by limiting the translation to within a given region of space, such as a plane or a box.

Scalar manipulations are interaction tools for inputting a single number into the environment. These manipulations behave much like a slider bar in a 2-D GUI, and could be used to enter information like time, length, scale, or speed. The scalar value is defined by moving a dragger part along a line. Typically, the line is another part in the dragger defined as an origin and a direction. During the button down event, the part is moved to the point on the line corresponding to the projection of the pointer onto the line. The scalar value is the distance from the origin to the location of the part. The manipulation can be constrained by limiting the scalar to within a given range, such as the positive numbers, or between zero and one.

Vector manipulations are interaction tools for inputting a 3-D vector into the environment. A common use for a vector manipulation is to control the orientation of an object. The vector is defined by moving a dragger part in relation to an origin. Typically the origin is another part in the dragger. As in 3-D translation, the distance from the eye to the part is calculated during the button press event. The direction is determined during the button down event by calculating where the vector manipulation part would be if it was translated as a 3-D translation. The normal vector from the origin to this hypothetical new location is the resulting vector. The manipulation can be constrained by limiting the vector to certain directions, such as on a plane or a hemisphere.

3.5. Navigation

Navigation refers to the ability of the user to move through a scene, or, more specifically, the ability to translate, rotate, and scale the user's viewpoint within the scene. In the interaction framework, the environment is divided into three navigation spaces: World Space, Person Space, and Horizon Space. Any geometric object created must be assigned to one of these navigation spaces.

World Space is the holding space for navigable scene objects. The user is able to navigate by translating, rotating, or scaling the objects contained in this space. Most scene content will lie in the World Space

Horizon Space is used to hold distant scene objects that, while part of the complete visual experience, are not part of the users full navigation space. Object in Horizon Space may be rotated, but not translated or scaled. Uses for the Horizon Space include background star fields for ISS simulations and the sky in a Mars lander simulation.

Person Space is defined as the portion of the geometry that the user cannot translate, rotate, or scale. That is, the user cannot navigate. Such geometry would always lie in the same position relative to the person under any transformation. The most common example of an object in Person Space is a Heads-Up Display. See Figure 3.

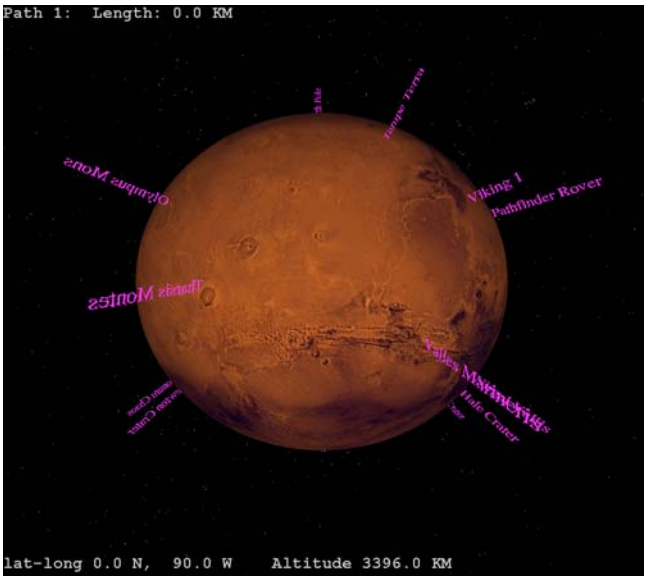

Figure 3 An annotated view of Mars from space. The planet and the geographic annotations lie in World Space, the star field in the background lies in Horizon Space, and the text at the top and bottom lie in Person Space.

The translation and rotation components of navigation are defined by specifying the topology of the World Space. The topology's primary function is to transform the 2-D joystick input from the wand into a reasonable navigation. The topology design allows the navigation framework to be extended specific to a certain application's requirements. For example, a flat topology with a 2-D translation and 1-D rotation describes a map-like environment where the user moves in two dimensions and rotates in one. This topology might be useful for an architectural walk-through of a building. A flat topology with a 3-D translation and 1-D rotation, on the other hand, allows the user to arbitrarily move in 3-D, but does not allow the user to change the pitch or the roll. This topology might be useful for visualization of a CFD simulation. A spherical topology with a 3-D translation and no rotation control would

describe a planet-like environment where the user could move to any longitude, latitude, and altitude, but would always orient toward the center. This topology might be useful for a satellite simulation.

Scale is defined independently of translation and rotation. By manipulating the scale, the user changes the relative size of the surrounding scene. For example, the user could be inspecting the outside of an airplane wing, then scale the scene larger in order to inspect the inside of the wing. In order to maintain the illusion of reality, scaling is not an immediate event. The scene grows or shrinks to the given size, rather than instantly scaling to a given size. The scene is grown or shrunk around the user's viewpoint so that the users position and orientation does not perceptually change during the scaling process.

An additional aspect of navigation is the ability to arbitrarily return to a given viewpoint. This is accomplished through the use of way-markers, or locations that mark a viewpoint. These way-markers can be created by the user or by the application. Later the user can return to a given viewpoint. The way-markers are scale-invariant, meaning that returning to a way-marker does not affect the scale of the scene, even if the way-marker was created at a different scale. Perceptually, the user will always return to the exact same spot, no matter what the current scale is. Again, in order to maintain the illusion of reality, the viewpoint does not immediately move return to the way-marker. The user "flies" back to the way-marker. The actual path the user takes is controlled by the topology of the space. The orientation is determined using a quaternion slerping operation between the current orientation and the way-marker orientation.

3.6. System Control

System control refers to the ability to give commands that affect the interaction mode or state of the environment. Examples would include loading a new data set, creating a new visualization tool, or changing the properties of a given tool. The manipulated dragger system provides a way to control many properties of the environment; however, it is not sufficient to control all of the properties in the environment. In particular, the dragger system is insufficient when a given control does not have a reasonable 3-D representation. In such cases another input system is required. This problem is particularly exacerbated because of the lack of keyboard or menu system, which would typically be used for system control.

Voice recognition is used to control the environment in an intuitive way that does not disrupt the 3-D immersive feeling of the environment. A grammar-based command and control recognition system is used, rather than a dictation-based system, in order to enhance accuracy and eliminate voice-training requirements. To simplify the recognition problem, only specific commands are recognized, rather than trying to recognize free speech. Each possible command was specified using a BNF-type grammar language. Related sets of commands are contained in a grammar object that can be enabled or disabled arbitrarily in order to further reduce the number of possible commands available at a given time. Each dragger defines its own grammar in the recognition engine. The dragger enables or disables its grammar on a button press or button release event respectively. Other elements in the virtual environment may also have grammars defined. For example, a grammar is defined in the World Space to control scaling and way-marker manipulation. Another grammar is defined for the overall environment to control universal events. Application specific grammars can also be created, such as a CFD grammar to load data and create visualization tools.

4. Multi-Agent Interaction

The interaction framework provides the building blocks for more advanced forms of interaction. This section discusses using the framework for building a multi-agent intelligent interface for interactive and automated feature extraction.

The multi-agent interface is inspired by the observation that in nature animals of relatively limited intelligence (ants, bees, birds, etc.) can perform complex actions by cooperating together as a flock or swarm. Although the intelligence of any given member is small, the collective intelligence of the group is high. In the multi-agent system, each agent moves through a problem space individually attempting to satisfy a set of given constraints. Due to their limited intelligence, the individual agents may succeed or fail in finding a particular location in the solution space, but the overall behavior of the group will be able to accurately extract the solution.

This swarming approach is a robust, generic algorithm to find nearly any feature in a data set that can be described as a simple set of constraints. Different features are extracted by providing different constraints, in the form of a mathematical relationship, to the agent members. Each agent uses local data to independently attempt to satisfy the given formula by moving through the data space in the direction indicated by the gradient of the data. For example, given a CFD solution, isosurfaces are defined by the constraint to find a given isovalue, cutting planes are defined by the constraint to solve a given plane equation, and vortex cores can be defined by a minimum pressure constraint. In this way, well-known algorithms can be redefined in this framework. This approach also provides a system to develop new methods and techniques by creating unique constraints, such as a constraint to find gradients perpendicular to the view vector, or to follow a vector field while constrained on a surface.

The swarming multi-agent system forms a user interface by allowing the user to interact, or converse, with the agents.

Through the voice recognition system, the user can create randomly located agents and control various properties of the agents, such as speed, color, size, etc. See Figure 4. The user is not interacting with a given agent, but rather the system of agents. The recognition system also provides an interface for the user to set constraints on the agents. For example, the user may say "find pressure value," or "find X value," indicating the constraint to find a particular pressure value, or a particular x-coordinate, respectively.

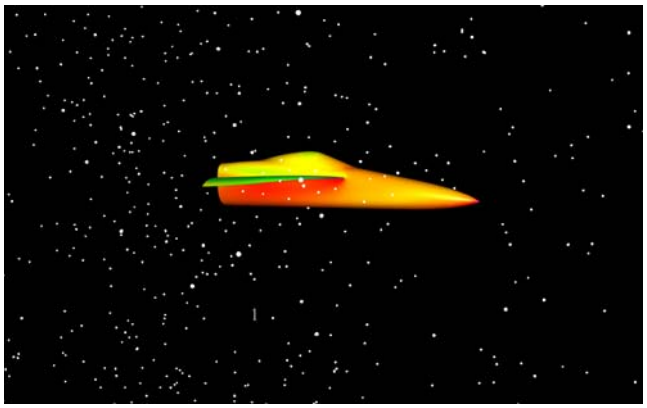

Figure 4 Swarming agents randomly distributed around the forebody of an F-18 CFD solution.

The dragger system provides further methods for the user to interact with the swarm. Seed draggers provide a method to continuously create agents in a local region, rather than randomly distributed throughout the space. Target draggers provide a method to further specify the agent constraints. In the example above, a translatable target dragger would be used to define which pressure value or x-coordinate to search for. As the user interacts with the dragger by arbitrarily moving it through 3-D space, the agents try to find the pressure or x-value at that location. The target dragger provides an easy capability to do interactive visualizations such as isosurfaces or cutting planes. Without a target dragger the agents would default to search for a minimum value. The constraint to find minimum pressure regions is equivalent to a constraint to find a vortex core, or the center of a spiraling region of air.

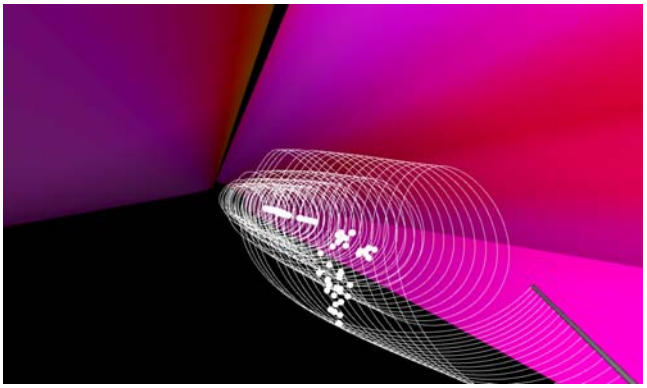

Figure 5 Streamlines and swarming agents visualizing the recirculation zone between a wing and wing-flap.

The swarming, or multi-agent interaction system is an effective high-level interface built on top of the interaction framework. The system provides a more intelligent, automated way for users to interact with their data and find interesting features in their data. See Figure 5 and Figure 6.

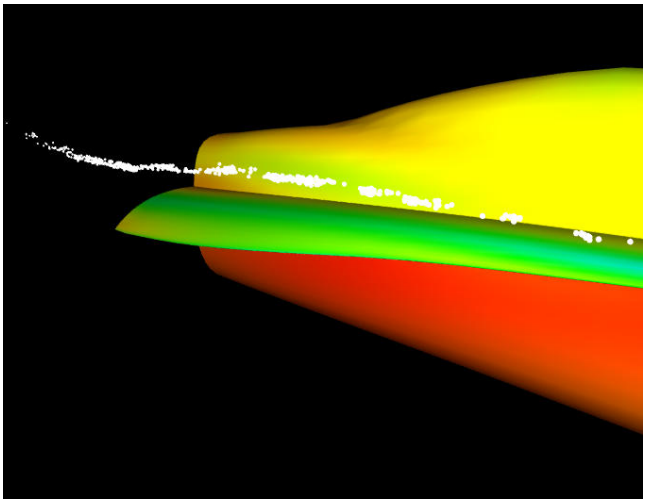

Figure 6 Swarming agents individually attempting to satisfy the constraint to find minimum pressure. The result is the vortex core of the airflow around the F-18 forebody.

5. Implementation

The interaction framework and swarming multi-agent interface were originally implemented on an SGI Onyx 2 graphical computer. The framework also works on Linux based PCs. The voice recognition component was implemented on a Windows-based PC networked to the SGI machine.

The framework was written in C++ using VRCO's CAVELib to handle the low-level VR configuration and SGI's OpenGL Performer as the 3-D graphics engine. The CAVELib interface is hidden in the framework, meaning that an application would not have direct access to those routines. Typically, an application developed in the framework would be OpenGL Performer based. The CFD visualization component was based on the Field Encapsulation Library from the NASA Ames Research Center. The voice recognition software was written in JAVA using IBM's ViaVoice recognition engine.

The framework was used in four widely diverse virtual environments: CFD visualization, ISS radiation studies, Mars mission planning, and molecular dynamics simulation. Initially the framework was developed for CFD visualization, but was continually modified and generalized for each new application.

6. Conclusion

The interaction framework discussed in this paper forms the basis for building new virtual environments quickly and easily. By providing a high-level, intuitive interface, developers can spend their time and effort solely in application and domain specific areas. The interface is modular and extensible so that it can be fine-tuned to take full advantage of a given hardware system.

A swarming multi-agent system provides an example of an intelligent interface into the environment. By interacting with agents the user can intuitively explore a given data set and quickly find interesting features in the set. The swarming interaction system is one tool in a visualization toolkit for exploring data sets.

7. Future Work

The behavior of a dragger is hard-coded by the application. A more generic method to use an XML configuration file to define the geometry and behavior of a dragger is planned. A distributed architecture will be used for implementing the interface as the front-end to a larger back-end simulation running on a remote machine, or cluster of machines. More modes of interaction will be added to the framework, such as a handheld computer or a haptic device.

The swarming interface needs a robust system for the user to enter an arbitrary constraint. Currently the constraints are limited to a collection of predefined constraints. Further enhancements to the concept include providing a limited form of communications between agents so that they do not congregate to the same location.

References

1. Bowman, D., Kruijff, E., Joseph, J., LaViola., & Poupyrev, I. (2001). “An introduction to 3-D User Interface Design” *Presence*, Vol. 10,No. 1, Feb 2001,96-108. © MIT.

2. Forsberg, A., Herndon, K. and Zeleznik, R., “Aperture Based Selection for Immersive Virtual Environments”, Technical Document, NSF Science and Technology Center for Computer Graphics and Scientific Visualization, Brown University. http://www.cs.brown.edu/research/graphics/research/pub/papers/sig93-widget.ps

3. Germans, D., Spoelder, H., Renambot, L., & Bal, H. (2001) “VIRPI: A High-Level Toolkit for Interactive Scientific Visualization in Virtual Reality” In *Proc. Immersive Projection Technology/Eurographics Virtual Environments Workshop*, Stuttgart, Germany.

4. Kessler, D. (1999). “A Framework for Interactors in Immersive Virtual Environments”, *Virtual Reality’99 Conference, March 13 – 17, Houston, Texas* IEEE Computer Society.

5. Pang, A., and C. Wittenbrink. (1995) “Spray Rendering as a Modular Visualization Environment”, *Computer Graphics, Vol. 29, No. 2, FOCUS: Modular Visualization Environments, Past, Present, and Future, edited by Gordon Cameron, 1995, pp. 33-36.*

6. Patel, Daniel (2003) “A Framework Architecture for Virtual Environments “ *CAVE Programming Workshop. August 29.* http://eve.hut.fi/cavews2003/favePerfAppPaper.pdf

7. Renambot, L., Bal, H., Germans D., & Spoelder, H. (2001). "Lightweight Programming for VR: Towards a Persistent Virtual Laboratory" *Workshop on Advanced Collaborative Environments. August 6*, IEEE International Symposium on High Performance Distributed Computing (HPDC-10), San Francisco, California.

8. Reynolds, Craig. (1987). “Flocks, Herds and Schools: A Distributed Behavioural Model” *Computer Graphics, 21(4), 1987, pp.25-34.*

9. VerHage, J.E., Sandridge, C.A., Qualls, G.D., Rizzi, S.A., (2002) "ISS Radiation Shielding and Acoustic Simulation Using an Immersive Environment," *Immersive Projection Technology 2002 Symposium, March 24-25, 2002, Orlando, FL.*

10. Wernecke, Josie (1993) “The Inventor Mentor : Programming Object-Oriented 3d Graphics with Open Inventor, Release 2” Addison-Wesley Longman Publishing Co., Inc.

11. Zelzenik, R., Herndon, K., Robbins, D., Huang, N., Meyer, T., Parker, N., and Hughes, J. (1993) “An Interactive Toolkit for Constructing 3D Interfaces.” *Computer Graphics (Proceedings of SIGGRAPH '93)*, 27(4), ACM SIGGRAPH, July, 1993, pp. 81-84.