VIRTUAL ENVIRONMENT OPERATING SYSTEM -- OVERVIEW

William Bricken
July 1992

This paper describes the Virtual Environment Operating System (VEOS). The VEOS kernel, which provides communications, database, and process management is currently in beta release. This kernel has been stable for more than a year, and is now being optimized for performance. The VEOS description includes design goals, system functionality, and system architecture.

VEOS provides the substrate for

- construction of experimental environments
- integration of multiple participants
- autonomous entities
- virtual world configuration
- interaction paradigms
- sensor and display integration

We propose to develop from design specifications several interaction tools, including the Virtual Body, Multiple Participants, and Interactive Construction.

Design of Operating System Infrastructure

Virtual reality applications present the most difficult software performance expectations to date. VR challenges us to synthesize and integrate our knowledge of sensor fusion, databases, modeling, communications, interface, interactivity, and autonomy, and to do it in real-time.

The Virtual Environment Operating System (VEOS) has been under development at HITL for the past two years. VEOS is designed to integrate the diverse components of a virtual environment. Currently VEOS is implemented as a UNIX shell.

VEOS consists of several software subsystems. The kernel manages processes, memory, and communication. The entity interface permits modeling objects in the environment, and the environment itself, in a consistent, object-oriented manner. The interaction tools empower a participant within the virtual environment.

The design of VEOS reflects multiple objectives, many practical constraints, and some compromises. Most importantly, VEOS is a research prototype, constantly undergoing revision and iterative refinement. As a research vehicle, VEOS emphasizes functionality at the expense of performance. It is a synthesis of known techniques rather than basic research into one technique. Since a research prototype must prepare us for the future, VEOS is designed to be as generic as possible; it places very little mechanism in the way of exploring diverse and unexpected design options.

VEOS Practical Design Decisions

Research prototype, 5-10 years ahead of marketplace
Functionality rather than efficiency
Incorporate commercially available software
Synthesis of known software technologies
Rapidly reconfigurable

VEOS Functional Design

Coordination between distributed, heterogeneous resources
Interactive rapid prototyping and reconfiguration
Entity-based modeling
Multiple participants
Concurrent divergent worlds

VEOS was designed for rapid prototyping. The VEOS interface is interactive, so that a programmer can enter a new command or world state at the terminal, and on the next frame update, the virtual world display will change. The database must accommodate update from multiple processors. We have adopted a Linda-like communication model which cleanly partitions communication between processes from the computational threads within a process.

The design of the virtual world could readily overwhelm a programmer if the programmer were responsible for all objects and interactions. Virtual worlds are simply too complex for monolithic programming. Entities within the virtual world must be modular and self-contained. The designer should be able to conceive of an entity, say a cow, independently of all other aspects of the world. VEOS is structured so that each entity is designed to be independent and autonomous. The system itself takes care of the lower level details of inter-entity communication, coordination, and data management.

In VEOS, all entities are organizationally identical. Only their structure, or internal detail, differs. This means that a designer needs only one metaphor, the entity, for developing all aspects of the world. Changing the graphical image, or the behavioral rules, or even the attached sensors, is a modular activity.

Entity modularity is particularly important when one recognizes that hardware sensors, displays, and computational resources are themselves first class entities. The entity model provides integration modularity for any new components to the VR system, whether they are graphical images, added cpus, or new input devices. Entities can be run independently, as worlds in themselves, or they can be combined into complex worlds. This means that devices and models can be tested and debugged modularly.

Because entities consist of both data and operating system processes, an entity can use other software modules available within the larger operating system. An entity could, for instance, initiate and call a statistical analysis package to analyze the content of its memory for recurrent patterns. The capability of entities to link to other systems software make VEOS particularly appealing as a software testing and integration environment.

Entity autonomy is best modeled by assigning a separate processor to each entity. This approach makes VEOS essentially a distributed operating system. Distributed resources arise naturally in VR, since the virtual environment is a social place, accommodating multiple concurrent participants.

In summary, VEOS is a significant effort to provide transparent low-level database, process, and communications management for arbitrary sensor suites, software resources, and virtual world designs. VEOS is the glue under VR. As such, it provides a strong integration environment for any team wishing to construct, experiment with, and extend VR systems.

VEOS Functionality

The Virtual Environment Operating System is a software suite currently written in C as a shell around the UNIX operating system. VEOS provides resource and communication management for coordination of the modules which make a VR system:

- i/o hardware, behavior transducing input and display devices
- world construction kits, CAD packages
- dynamic simulation kits, for interaction and animation
- virtual world tools
- computational and display processors

Figure 1 represents the functional connectivity between the system modules.

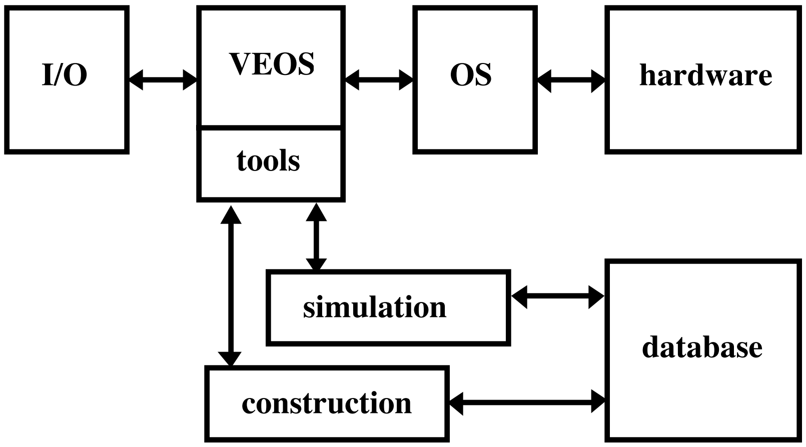

Figure 1. Functional overview of the current VEOS implementation

The primary functional subsystems of the Virtual Environment Operating System include:

Interpretation: couples the input activities of the participant to computational processes.

Modeling: manages the computational behavior and model of virtual environment elements.

Display Integration: integrates output signals from the model and from other sources to drive the display devices which create the virtual environment.

VEOS Architecture

Figure 2 below represents the VEOS system architecture. Arrows indicate the direction of dataflow. The architecture in Figure 2 contains three subsystems.

The behavior and sensory transducing subsystem

This subsystem implements the fundamental interface paradigm shift of VR, from user actions that accommodate the needs of symbolic computation to natural participant actions which are interpreted by the computational system. It consists of:

- The participant.
- Sensors which convert natural behavior from participant to digital streams.
- Display devices which convert the digital model to subjective sensory information.

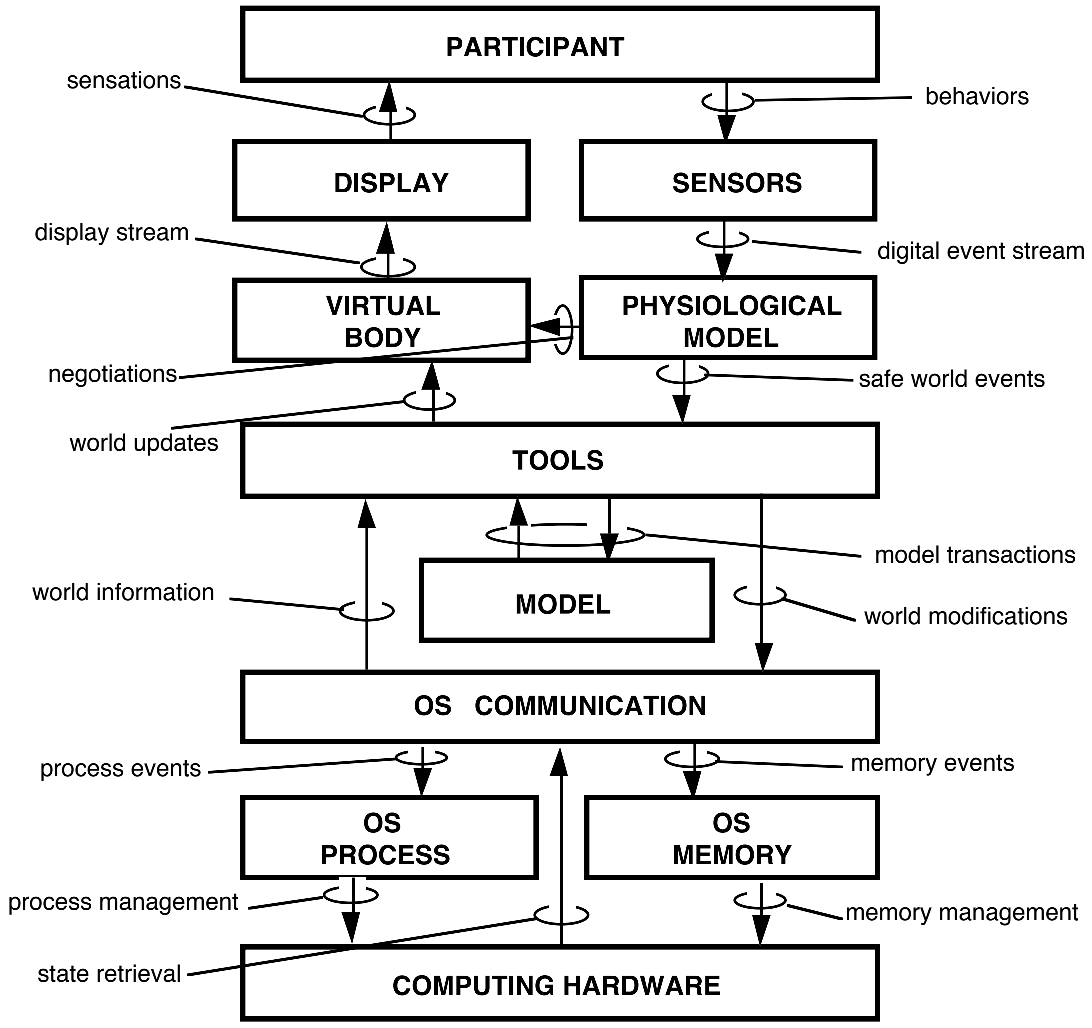

Figure 2. VEOS system architecture.

The virtual toolkit subsystem

The virtual toolkit subsystem coordinates display and computational hardware, software tools and resources, and world models. It provides a wide range of software tools for construction of and interaction with models, including editors of objects, spaces, and abstractions; movement and viewpoint control; object inhabitation; boundary integrity; display, resource and time management; multiple concurrent participants; programmable internal processes within models; and history and statistics accumulation. The components of this subsystem are:

- The physiological model which maps digital streams from input devices onto a representation of the participant within a virtual world.

- The virtual body which maps the physiological model onto virtual world behavior which can be displayed from a subjective perspective.
- Virtual world tools (compilers, editors, languages, interaction tools) which interface with the operating system, databases, and software and hardware resources.
- The virtual world database which stores world state, entities, and static and dynamic properties.

The operating subsystem

The operating subsystem customizes the VR software to a particular machine architecture. It consists of:

• Operating system communications	(message control)
• Operating system memory	(paging, garbage collection)
• Operating system processes	(threads and tasks)
• Computational hardware	(binary machine code)

Software Environment

The following sections describe the interactive tools for virtual environments that HITL proposes to develop or extend for this proposal. We suggest the development of interaction, construction, and assessment tools built on the VEOS testbed infrastructure.

This section describes the FERN model, our design of an entity management facility. FERN is implemented in prototype and is not as stable as the VEOS kernel. This section includes the design requirements for the entity manager, entity activity and architecture, the functions, resources and processes of the FERN model, and several examples of the use of the entity manager. HITL is still developing skills in the use of FERN and the VEOS kernel; we propose to extend and refine the FERN model and to develop it robustly.

The section presents several interaction tools we propose to develop. The Wand has been implemented in prototype, and will be significantly extended in this proposal. The Virtual Body is the central organizing concept and software tool for physiological calibration and measurement. We propose to build the Virtual Body, which is currently in design phase, in its entirety for this project.

The section discusses our proposal for world construction tools. The three tools we believe are necessary for the rapid construction of environments include Form Abstraction, Generalized Sweeps, and Image Conversion.

Entity Management (FERN)

The FERN (Fractal Entity Relativity Node) is the VEOS entity manager. It provides accurate and consistent world data management. The architecture of the FERN entity manager is presented in Figure 3.

FERN Design Requirements

Generic world structure: The computational system must impose as little predetermined structure upon the virtual environment as is possible. It should be able to support as many types of world designs, spaces, systems, and interactions as possible.

Distributed data and processing: The system must allow its needs to be distributed across many computer systems, not only permitting multiple participants, but also for the support of complex worlds on relatively low-end computing hardware.

Reliable data management: Due to the distributed nature of the system, there must be a reliable and consistent way to handle the flow of information between entities in the world.

Fast hardware updates: A requirement in any virtual reality system, hardware updates, such as the update of viewed images as the participant moves through the world, must not be sacrificed for information flow.

Furthermore, for a system to support the type of interaction between entities that is envisioned, entities themselves must have certain features:

Autonomy: Entities must be able to support and operate themselves, without dependence on any other entity or system.

Equality: Again because of the distributed nature of the system, plus the requirement for generic worlds and autonomous entities, all entities must be equal in structure and function.

Environmental model: The nature of the virtual world is that the world itself is an entity, and each entity is itself a world. This must be reflected in the system design.

Entity Activity

An entity in the FERN model is a single and complete set of processing functions embodying zero or more objects in the virtual world. Usually, an entity will be the whole of any one being in the world, but it is possible to have cooperating entities form a single being. Any set of objects that operate as a connected whole and have one set of behaviors and functions is in entity.

Entity Behavior: Entity equality and autonomy is fundamental to this system. As such, behavior cannot be based on control of one entity by another, but instead must be built on controlled individual reactions to world events. In the FERN model, all behavior is reduced to reactions to perceived data regarding other entities in the world. In order to create entity actions and interactions, including anything from a reaction to a tool to hunting for food, the world builder must create the corresponding set of behaviors within each entity.

Entity Interaction: In order to create the type of complex interactive worlds that are currently envisioned, have them run on hardware that is available currently, and allow multiple participation from potentially remote sites, virtual world entities must be autonomous and equal.

Entity Autonomy: In the FERN model, every entity has the same basic organization, and operates as a peer to all other entities in the virtual universe. An entity's capabilities are only limited by the extent to which they are programmed and the speed of the processor the entity is running on. In keeping with the reactive model of entity interaction, each entity is ultimately responsible for its own actions. Every entity has the opportunity to refuse to "obey" another entity.

Environmental Entity: The virtual world is an entity to itself, as much as any entity contained in it is an entity. In the FERN model, since every entity contains both the functions for participating in a world and maintaining a world, any entity may do either or both at the same time.

Entity Architecture

The primary organization within each modeled object is an input-process-output loop. Input is identified by the object's sensors, which themselves are subsystems. Output is defined by effectors. The structure of each object/system consists of:

Input buffer	(fed by sensors)
Priorities	(internal values selecting input)
Disposition	(rules triggered by selected input)
Knowledge	(state collected by rules)
Output buffer	(actions generated by rules)

Sensors store input in a buffer. Objects can be programmed by inserting rules into their disposition. A set of rules for attention select a single input item to compare to the trigger clauses in the set of disposition rules. When a particular rule is matched, the action it specifies is carried out. An action may be to store the input as knowledge or to cause an effector to change the state of the environment. Some rules may be contingent on stored knowledge to be triggered. Some rules may be independent of input, they form the internal processing disposition of the system.

This architecture allows situated responses. Objects can react to environmental changes (as perceived by input sensors and filtered by priorities) dynamically and opportunistically. Objects can also learn from experience and internally abstract experiences to form idiosyncratic knowledge bases. When rules include inference over the knowledge base, an object can behave as an expert system.

In summary, the software architecture of the VEOS and of virtual objects incorporates rule based logic programming locally in object-oriented autonomous systems. Programs are reactive and situational (data-driven) when internal priorities are satisfied by existing input, and autonomous and learning (goal-driven) when there is not prioritized input. Disposition rulebases are programmable by other systems (in particular by the participant).

The FERN Model

All entities are organizationally equal. The details of the standard entity architecture are presented in Figure 3 and described in the sections which follow.

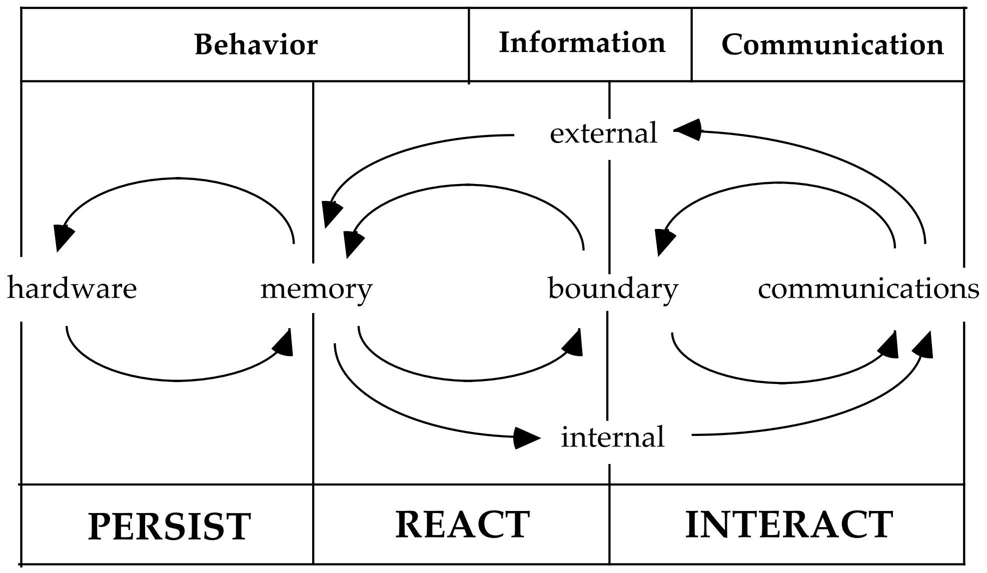

Figure 3: The FERN Model

FERN Functions

Each entity performs three functions:

Database: Each entity maintains a database of personal attributes, sibling attributes, and children attributes. It uses the searching language Nancy, a basic component of the VEOS system, to accomplish this task.

Behavior: Each entity has two function loops that process input data, both from the entity's own internal states and the states of other entities in the world, into state changes. These processes are controlled by programming in the LISP language.

Communication: Communications with "parent" and "children" entities are the backbone of the system. It is the only way for the entity to be aware of the state of the worlds both inside and outside the entity. The communication process is facilitated by the use of Talk, another basic component of the VEOS system.

FERN Resources

The data used by the entity's processes are stored in areas called resources. (The five resources in Figure 3 are shown in typewritten font.) They are both the sources and the sinks for the data used and created by the entity. There are three resources that are specifically stored in the entity’s data space accessed through Nancy:

Boundary: Data about the self that is meant to be communicated with the parent and thus shared with as many siblings as are interested is stored in the boundary. This area of data space is readable and writable. An entity would read the boundary to get its current information and write to it to change its state through a reaction.

External: The external partition of the data space contains information regarding the entity's siblings in all the worlds that the entity is a member of. The entity can set perceptual filters to tell the parent how to filter all the world data into only that information the entity is able to react to. Unlike the boundary, this area is readable only.

Internal: The internal data space is comprised of all data collected from the boundaries of contained entities. While it cannot be written, it is meant to filtered and communicated to all contained entities.

The other two resources in the FERN model contain data about the entity that is never passed to the rest of the world. They are involved with the ongoing maintenance of the entity in relation to itself and to the physical world ("real" reality).

Memory: This resource contains data pertaining to states that are not directly communicated to other entities. Memory can exist in either the entity's data space, LISP space, or a combination of both, depending on the needs of the entity and the structure of the data.

Hardware: This is data produced and/or used by an entity's related hardware. Data can be read in from a hardware device or written to it. An example of a device that would be read in would be a position tracker, providing both position and rotation in the world with regard to the transmitter. Data might be written to the hardware resource, for example, to enable the imager to create the proper images. Position and rotation data can be attached directly to the imager to provide the fastest refresh rate to the participant.

FERN Processes

Processes are the operational functions the entity uses to exist in the virtual world. There are three types of processes that occur (shown in large letters in Figure 3). The processes that provide reactions based on input stimuli are PERSIST and REACT. The other process is FERN, which maintains all necessary communications with other entities in the virtual universe.

PERSIST: These are the processes that are independent of any factors outside the entity. They rely only on data known to the self. The PERSIST loop happens in local time - as often as the hardware will allow. Unlike REACT, which must wait for an update from the parent to occur, PERSIST will occur as often as computationally possible. An example of a PERSIST process is the update of the image due to the position and rotation of the head.

REACT: These are processes that both depend on and affect factors inside and outside the entity. The REACT loop only takes place as new updates to the boundary and external partitions are made. FERN time, as opposed to local time, is completely dependent on the speed of the network and of the parent, and therefore of the processor that the parent is running on.

FERN: The FERN function handles all communication with the parent and children entities. As a parent, or world, the FERN keeps track of all entities in the world. It accepts updated boundaries from each entity and stores it in it's internal data space partition. In turn, the parent updates each internal entity's external partition with the current state of the world, in accordance with the entity's perceptual filters, as often as the entities send updates themselves.

Examples

Entering a world: To enter a new world, an entity will send it’s boundary to the entity containing the world. Subsequent updates to other entities within that world will include information about the recently-joined entity.

Follow: An entity can add a distance offset to the location of another entity, in effect following that entity. Following is dependent on another entity's behavior, but is completely within the control of the following entity.

Move with wand: The wand, like any tool entity, will publish its current values to its boundary. An entity that wants to use these values, such as the virtual body using the wand to move, will be sensitive to these published values when they come into its external partition.

Inhabitation: Simply, the inhabiting entity will use the inhabited entity’s relevant boundary information as it’s own, thus creating the same view and movements as the inhabited entity.

Portals: An entity sensitive to portals can move through the portal to another world or to another place in the current world. Upon colliding with the portal, the entity will change its boundary attributes into the position, rotation, and inhabited space values attached to the portal.

Learning: Entities can learn new behaviors by receiving new function definitions and then adding the new function call to their behavior loop. Such code would likely include both the function definition and the command to add the command to either the entity's REACT loop or PERSIST loop. More intelligent entities might build in the ability to discern different types of code and make an informed decision about what code to incorporate and from whom.

Interaction Tools

Interaction within the virtual environment is mediated by primarily two VEOS tools: the wand and the virtual body. Rudimentary aspects of these tools have been incorporated into the testbed. We propose to enhance their functionality in the following ways.

The Wand

The Wand is an interface tool which uses a simple physical device for a wide range of functions. The physical device is a rod with a 6 degree-of-freedom sensor on one end which supplies position and orientation information to the model. The sensor information inhabits a virtual rod held by a virtual hand. We assign functionality to the Wand by attaching a voice sensor to it and inserting rules into its set of dispositions. Some functions of the Wand include:

- Ray on/off: A ray emanates from the end of the virtual rod, collinear with it.
- Identify: The first object which the ray penetrates returns its name.
- Distance: the length of the ray vector, expressed in the metric of the intervening space, is returned.
- Connect: Construct a communications port between the rod and the identified object.
- Jack: Teleport the viewpoint of the rod (along the ray vector) to the identified point on the object.
- Grasp: Attach the end of the ray to the identified object. When the Wand is moved, the object stays attached. When the Wand is rotated, the object rotates.
- Normal: Rotate the identified object so that the intersecting ray is normal to the object's surface.
- Sight: Jack into the Wand, the viewpoint of the patron issuing the command is linked to the ray vector.
- Move faster/slower: Move the viewpoint of the patron along the ray vector.

The Virtual Body

Since the participant is included within the virtual environment, the representation of self is fundamental to virtual interface design. The Virtual Body is the primary reference point, the interface between the user and the virtual environment. It provides direct access to computational graphic objects; it is the channel of direct action and control. Monitoring the Virtual Body provides the computational system with a complete record of actions taken by the participant.

The Virtual Body is a software toolkit for:

- attaching arbitrary hardware input devices to arbitrary representations of components of our body. Usually the linkage will emphasize naturalness.
- making psychometric measurements of behavior in a virtual environment, and
- maintaining coherence between the participant's model of physical activity and the virtual representation of that activity.

In the VEOS architecture, the Virtual Body consists of two components, the Physiological Model and the Virtual Model. The physiological model is intended to be anthropomorphically correct. It contains information about the participant's physical body and calibrates this information to the virtual environment. The virtual model consists of the mapping of accurate physiology onto whatever (arbitrary) form the participant chooses as a self-representation in the virtual environment.

The unique aspects of the Virtual Body tool are:

Generic: The Virtual Body includes software for mapping the sensor data stream to representations of body components in a virtual environment display, interpreting the data stream as instructions to change the virtual environment, and collecting and analyzing the data stream as psychometric information.

Transparent: The sensor measuring participant activity is designed to be transparent. Natural physical movement directly affects the computation; there is no apparent interface. The Virtual Body software maintains the illusion of direct interaction.

Interactive: Mapping between physical action and computational effect is flexible and dynamic. A spoken word, for example can change the computational effect of shifting one's gaze from "Identify that object" to "Transport me to that object."

Natural: Since action can be taken literally (there is no symbolic transcription filtering the meaning of behavior), performance in a virtual environment mimics performance in reality. As long as the representation of the task is valid, the user's behavior directly indicates the user's ability to perform. Physical actions in a virtual environment furnish psychometric data on performance, resource expenditure (load), and cognitive model.

Multiple Participants

HITL is developing the mathematical tools which coordinate actions of multiple participants in virtual worlds. Virtual worlds permit mutually inconsistent models across multiple participants. Each participant can maintain a separate personal environment concurrently in the same virtual space. Communality of mutually shared perspectives is negotiated rather than assumed. VEOS provides a prototype capability for participants:

- to share environments with an indefinite number of other participants,
- to communicate naturally by conversation, gestures, and movement in a visual space,

- to elect to partition the virtual space into parts which are communal and parts with are private, and
- to access personal environments for individual interaction with data and information.

World Construction Tools

A variety of world construction tools will be needed in robust virtual interface research environments, including commercial three-dimensional modeling packages, simple inclusive construction kits, and entity disposition editors. The applications relevant to this proposal will, in addition, require fast segmentation and model extraction from spatially distributed sensor data. Computational algorithms for these functions will be developed and tested.

Form Abstraction

A virtual environment can be abstracted into the composition of three elements: the individual objects or entities, the space these entities jointly occupy, and the spatial relationship between each entity and the origin of the space, as expressed in the metric of that space. The advantages of form abstraction include:

- positional information is separate from the geometric form of an entity
- space can be subdivided recursively
- entity interactions can be treated as pair wise
- entities can be locally independent of the space they occupy

We propose to explore recursive techniques for rapid construction of environments.

Generalized Sweep

Construction of virtual environments from within the environment is an unsolved problem. We believe that sweeps, initiated by hand and arm gestures are the most natural way to generate solids in virtual spaces. We propose to develop a generalized sweep construction tool with the following properties:

- hand and arm gestures trigger the creation of solid objects
- sweeping a point will generate a one dimensional line; sweeping a line, or bending a line, will generate a plane; sweeping a plane will generate a solid

- translational, rotational, and scaling sweeps
- sweep curve may be freehand or specified by a function
- freehand sweeps will have a TIDY option which will smooth it to a function of specified degree
- object libraries will include the basic solids

Assessment Tools

The Participant Tracking System described above will require the development of a number of software tools, including event history recording, time-tagged communication, instant replay and backtracking, as well as tools for automated generalization, classification, distribution statistics, correlational statistics, and trend analysis.