# SOME SOFTWARE TECHNIQUES FOR VIRTUAL ENVIRONMENTS

William Bricken
February 1993

VR is the body of multidisciplinary techniques that apply computation to the generation of experientially valid realities. Broadly, virtual reality is that reality which people construct from information, a reality which is potentially orthogonal to the reality of mass. Within computer science, VR refers to the computer-based construction of participatory information, providing the direct experience of a digital environment.

Computer-based VR consists of a suite of four interrelated technologies:

| | |
|---|---|
| Behavior Transducers: | hardware interface devices |
| Inclusive Computation: | software interaction techniques |
| Intentional Psychology: | biological constraints and plasticity |
| Experiential Design: | functionally aesthetic construction |

*Behavior transducers* map natural behavior onto digital streams. Natural behavior is what two-year-olds do: point, grab, issue single word commands, look around, toddle around. Behavior transducing interface devices convert our actions into digital information, extending the body with position tracking, voice recognition, kinesthetic feedback, infrared vision and remote manipulation. Transducers work in both directions, physical behavior to digital information and virtual display to subjective experience.

*Inclusive computation* provides tools for construction of, management of, and interaction with digital environments which surround, or include, a participant. Participation within information is also called immersion. Inclusive software techniques include physiological models, behavioral entities, inconsistency maintenance, autonomous agents, sensory cross-mapping, active environments, and asynchronous parallelism.

The *intentional psychology* of VR addresses how to structure environments that respond to our intentions as well as our actions, that reflect our imagination as well as our formal specifications. This requires the integration of digital computation with human functionality, our physiology, our perceptions, our cognition, and our intentions. Intentional psychology incorporates situated learning, interactive construction, individual customization, participant uniqueness, subjective evaluation, multiple concurrent interpretations, satisficing solutions, choice-centered computation, and personal, mezzo and public spaces.

*Experiential design* seeks to unify inclusion and intention, to make the virtual world feel good. The central design issue is to construct inclusive environments which are fun and functional for a participant. There is no interface to design, but their is a world of creation to explore. The tools

for experiential design may include wands, invocations, embedded narrative, adaptive refinement, genetic algorithms, artificial life, and social construction.

VR unifies a diversity of current computer research topics, providing a uniform metaphor and an integrating agenda. The physical interface devices of VR are similar to those of the teleoperation and telepresence communities. VR software incorporates artificial intelligence, expert systems, pattern recognition, and reactive planning. VR worlds provide extended senses, traversal of scale (size-travel), synesthesia, fluid definition of self, super powers, hyper sensitivities, and meta physics. VR requires innovative mathematical approaches, including visual programming languages, spatial representations of mathematical abstractions, imaginary logics, void-based axiomatics, and experiential computation. The entirely new interface techniques and software methodologies cross all disciplines, creating totally new alignments between knowledge and activity.

## Software Techniques

The primary task of a virtual environment operating system is to make computation transparent, to empower the participant with natural interaction. The technical challenge is to create mediation languages which enforce rigorous mathematical computation while supporting intuitive behavior. Since humans evolve in a spatial environment, VR uses spatial interaction as a mediation tool. The design goal for natural interaction is simply direct access, interaction not filtered by a layer of abstract representation. This implies both eliminating the keyboard as an input device, and minimizing the use of text as output.

In order to turn the machine into a tool of the mind, we must develop programming techniques based on behavioral rather than processing metaphors. VR operating systems must attempt to restructure programming tools, from the bottom up, in terms of spatial, organic models.

Virtual environment software tools coordinate display and computational hardware, software functions and resources, and world models. Software tools for construction of and interaction with digital environments include movement and viewpoint control; object inhabitation; boundary integrity; editors of objects, spaces, and abstractions; display, resource and time management; coordination of multiple concurrent participants; and history and statistics accumulation.

Five central components of a virtual environment software suite follow:

The *physical model* maps digital input onto a realistic digital representation of the participant and of the physical environment the participant is in. This model is responsible for screening erroneous input data and for assuring

that the semantic intent of the sensor input is appropriately mapped into the world database.

The *virtual body* customizes effects in the virtual environment (expressed as digital world events) to the subjective display perspective of the participant. The virtual body is a representation of the participant's manifest form in the virtual world, and is tightly coupled to the physical model of the participant in order to enhance the sensation of presence. Differences between physical input and virtual output, such as lag, contradiction, and error, can be negotiated between these two components of the body model without entering the world model.

*Virtual world tools* program and control the virtual world, and provide techniques for navigation, manipulation, and participatory interaction. All transactions between the model and the system resources are managed by the tool layer.

The *virtual world database* stores world state, and static and dynamic properties of software objects within the virtual environment. Tools access and assert database information through model transactions.

The *virtual environment operating system* manages transactions between computational software and hardware. Since machine level architectures often dictate efficiency, the operating system is particularly important for real-time performance, including update rates, complexity and size of worlds, and participant responsiveness.

The operating system *communications management* (messages and networking) coordinates resources with computation. The intense interactivity of virtual worlds, the plethora of external real-time devices, and the distributed resources of multiple participants combine to place unusual demands on communication models.

The operating system *memory management* (paging and allocation) coordinates storage and retrieval. Virtual worlds present massive databases, concurrent transactions, multimedia data types, and partitioned dataspaces.

The operating system *process management* (threads and tasks) coordinates computational demands. Parallelism and distributed processing are fundamental to VR systems.

## The VEOS Project

The Virtual Environment Operating Shell (VEOS) is a software suite currently written in C that wraps around the UNIX operating system. It provides a tightly integrated computing model for data, processes, and communication. VEOS is platform independent, and has been extensively tested on the DEC

5000, Sun 4, and Silicon Graphics VGX and Indigo platforms. The programmer's interface to VEOS is XLISP 2.1, written for public domain by David Betz.

VEOS consists of four tightly integrated software subsystems.

SHELL manages entity initialization and linkages.
TALK manages interprocess communications.
NANCY manages the distributed parallel database.
FERN manages entity processes.

An *entity* is a coupled collection of data, functionality and resources, which interacts with an environment using a behavioral metaphor. Each entity within the virtual world is modular and self-contained, each entity can function independently and autonomously. VEOS takes care of the lower level details of inter-entity communication, coordination, and data management. In VEOS, everything is an entity (the environment, the participant, hardware devices, software programs, and all objects within the virtual world).

Several specialized entities (such as Mercury, the fast virtual body; UM, the behavior specification package; MIDI, the musical interface; and BlockLogic, the mathematical visualization tool) were built as the FERN system was iteratively refined. As well, over a dozen mid-size applications have been developed using suites of VEOS entities. These projects range over several Masters Theses, and include applications to manufacturing (a bicycle company assembly line, integrated with AutoMod simulation software), education (worlds created by high school students during five day world building projects), perception (a comparative study of virtual and physical spaces), scientific visualization (the Mars database, optics tracking, modeling semiconductor junctions), group interactivity (Catch, building dynamic environments, group blocks world), and demonstration systems (The Boeing Virtual Osprey, TopoSeattle, and the Metro).

*TALK* uses heavyweight sequential UNIX processes to connect networks of workstations into a virtual world processor. TALK uses two simple message passing primitives, SEND and RECEIVE. Messages are transmitted asynchronously and reliably, whether or not the receiving entity is waiting.

*NANCY* provides a content addressable database accessible through pattern-matching. It is modeled on Gelernter's coordination language Linda. The Linda approach separates programming into two essentially orthogonal components, computation and coordination. Computation is a singular activity, consisting of one process executing a sequence of instructions one step at a time. Coordination creates an ensemble of these singular processes by establishing a communication model between them. Programming the virtual world is then conceptualized as defining "a collection of asynchronous activities that communicate" [Gelernter and Carriero ACM'92].

Structurally, the database consists of a collection of fragments of information, labeled with unique syntactic identifiers. Collections of related data can be rapidly assembled by invoking a parallel pattern match on the syntactic label which identifies the sought after relation.

All database operations in VEOS are implemented at the lowest level by a single computational algorithm: match-and-substitute. Elements which have a complex syntactic description (such as the structure 2x+3x) are identified by pattern matching. Then a simpler expression (the result 5x in the example) is substituted for the complex one.

## Dynamics

Entity dynamics is modeled by associating behavioral functions and rulebases with sensory input and with processes internal to the entity. Entities exhibit persistent behavior by running algorithms that do not interact with the environment. Entity behavior can be classified by the complexity of the behavioral functions and rules.

*Reactive* entities have rules that trigger when specific events are posted to the environmental database.

*Responsive* entities react to their environment, but they also have memory resources to store previous experiences. These entities can exhibit complex delayed responses and critical incidence behavior.

*Inferential* entities have a small inference engine which they can apply to their accumulated database of experiences and internal rules.

*Coordinated* entities share rules with other entities. A typical use of coordination is to share synchronized clocks, permitting time sequencing of group behavior.

*Autonomous* entities provide exploratory tools for emergent behavior. Autonomous behavior is generated by meta-rules, rules which change the entity's behavioral rules.

The most complex entity is one that is inhabited by a human participant. In *inhabited* entities, dynamic behavior is slaved to physical transducers attached to the participant. These signals are standardized to the participant's physiological body model. The physiological model is then mapped onto a virtual body, which can be an arbitrary representation.