# Interactive Simulation of Fire in Virtual Building Environments

Richard Bukowski* Carlo Séquin†

Computer Science Department ‡
University of California, Berkeley

## Abstract

This paper describes the integration of the Berkeley Architectural Walkthrough Program with the National Institute of Standards and Technology's CFAST fire simulator. The integrated system creates a simulation based design environment for building fire safety systems; it also allows fire safety engineers to evaluate the performance of building designs, and helps make performance-based fire codes possible. We demonstrate that the visibility preprocessing and spatial decomposition used in the Walkthru also allow optimization of the data transfer between the simulator and visualizer. This optimization improves the ability to use available communication bandwidth to get needed simulation data to the Walkthru in the best order to visualize results in real time; an appropriate communication model and data structures are presented. General issues arising in the integration of environmental simulations and virtual worlds are discussed, as well as the specifics of the Walkthru-CFAST system, including relevant aspects of the user interface and of the visualization and simulation programming interfaces. A recommendation is made to structure future simulators in such a way that they can selectively direct their computational efforts toward specified spacetime regions of interest and thereby support real-time, interactive virtual environment visualization more effectively.

**CR Categories:** I.3.2 [Computer Graphics]: Graphics Systems—Distributed/Network Graphics; I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics Data Structures and Data Types; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual Reality; I.6.7 [Simulation and Modeling]: Simulation Support Systems—Environments; J.6 [Computer-Aided Engineering]: Computer-Aided Design.

**Keywords:** Virtual/Interactive Environments, Scientific Visualization, Simulation, Virtual Reality, Interactive Techniques, Information Visualization

*bukowski@cs.berkeley.edu
†sequin@cs.berkeley.edu
‡Soda Hall, Berkeley, CA 94720

## 1 INTRODUCTION

Virtual environments are of major interest to computer graphics researchers; this is due, in part, to their ability to immerse the user in a computer-generated alternate reality in which we can easily recreate scenarios which are too dangerous, difficult, or expensive to play out in real life. One application domain with a particularly high expected payoff is building design evaluation, where scientists, engineers, architects, and other professionals can enter a virtual space and evaluate its physical structure without actually building or affecting a real instance of that structure. With such a system, users could preview architectural designs, evaluate their performance with various metrics, and do simulations and potentially destructive "what-if" experiments (such as fire safety studies; see figure 1) cheaply and with no risk. To obtain useful answers to such experiments, we need to integrate good physical simulations with virtual environment interfaces. Integration of powerful simulation technology with virtual reality visualization systems affords the possibility of intuitive interpretation and visualization of the results of complex and powerful simulations via 3D computer graphics.

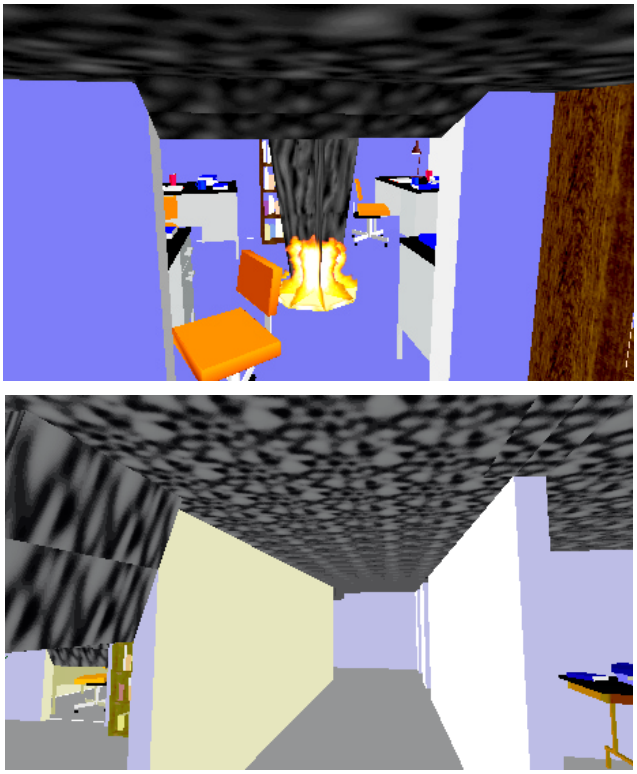

Figure 1: *Above, a pool fire, three-quarters of a meter in diameter, rages in a student office in Soda Hall; 100 seconds after ignition, smoke nearly fills the room. Below, smoke has spread into the hallways.*

We are attempting to realize some of these advantages for the benefit of fire safety in architectural environments. We are in the process of integrating the National Institute of Standards and Technology's (NIST) Consolidated Model of Fire and Smoke Transport (CFAST) [15] into the Berkeley Architectural Walkthrough (Walkthru) system [17, 10]. CFAST currently provides the world's most accurate simulation of the impact of fire and its byproducts on a building environment. Integrated into the Walkthru, it provides real-time, intuitive, realistic and scientific visualization of building conditions in a fire hazard situation from the perspective of a person walking through a burning building. The viewer can observe the natural visual effects of flame and smoke in fire hazard conditions; alternatively, scientific visualization techniques allow the user to "observe" the concentrations of toxic compounds such as carbon monoxide and hydrogen cyanide in the air, as well as the temperatures of the atmosphere, walls, and floor. Warning and suppression systems such as smoke detectors and sprinkler heads can be observed in action to help determine their effectiveness. This technology can be used to improve fire safety by helping engineers and architects evaluate a building's potential safety and survivability through performance-based standards (i.e. how well the building protects its occupants from the fire). With more development, it could also be used to help train personnel in firefighting techniques and rescue operations by presenting them with practice situations that are too risky to be simulated in the real world.

While the combination of virtual reality and environmental simulation constitutes a framework for very powerful tools, it also raises many implementation challenges. Among these challenges are interaction with the virtual world, setting up and dynamically changing simulation conditions from within the virtual world to a simulator, designing "visualization-oriented" simulators, transporting simulation results to the visualizer, integrating the simulator's results with the virtual environment, and visualizing those results in a way that is useful to the user; either descriptively, in the case of scientific visualization applications, or realistically, in the case of training or entertainment applications. These problems are compounded by an additional desire to distribute both the virtual environment and the simulation over multiple computers – potentially connected by relatively high-latency, low-bandwidth networks such as the Internet – when attempting to simulate and visualize large buildings with hundreds of rooms.

In this paper, we present an approach to the problem of distributed simulation-visualization data management that is optimized for densely occluded polyhedral environments (i.e. buildings) based on the Walkthru and CFAST programs. Walkthru has already addressed some of the problems of distributed visualization and of the interaction between the user and the virtual world [16, 11, 4]. We show that the basic virtual environment structure used in the Walkthru, a spatial subdivision of the world into densely occluded cells with connecting portals, can be put to good use for simulation data management. In addition to optimizing the visualization task, it is also useful for optimizing bandwidth requirements between a visualizer and simulator, both for communicating scenario information to the simulator and for communicating simulated states back to the visualizer. Using this structure, we can minimize bandwidth requirements for arbitrarily large visualizations and simulations, and relieve the visualization and simulation designers of the complexity of the data management problem. The solution is extensible to multiple distributed visualizers and simulators operating on one virtual world. It also suggests an important attribute of future simulation design for simulation developers who wish to make "virtual reality-oriented" real-time simulators: the ability to partition a simulation effort so as to concentrate computation on parts of the environment of immediate interest to the observer (i.e. those parts that affect the areas which are currently being viewed). This issue is also being studied by other groups at Berkeley [5].

In section 2, we discuss the simulation/visualization data management problem in the context of other related work in virtual environment simulation. In section 3, we present an overview of the two components of the system, Walkthru and CFAST, the issues involved in combining these two programs, and more generally, issues in combining visualization software with simulation software in a densely occluded building environment. In section 4, we present the most important abstract representations for the exchange of simulation data and the corresponding communication system. Section 5 explains the APIs and functionality provided to the visualization front end, the user, and the simulator. Finally, in section 6, we discuss some of the details of the internal workings of the simulation data management system.

## 2 RELATED WORK

The most frequent application of virtual reality technology so far has been visualization of static spatial environments. The majority of current virtual worlds are nearly static environments with a few movable objects and avatars inside. The most common applications of these systems are either peer-to-peer simulation of the user's interaction with other users or simulated entities, or systems that use physics to make the world seem more "real" to an immersed user. Some more famous examples of the former include the Iowa driving simulator [7], where the user's vehicle interacts with other independently-simulated road vehicles, and the department of defense's NPSNET [13, 19], where "units" of military vehicles engage in simulated combat on static terrain. Each simulated unit (or vehicle) communicates its status to each other unit, but since the environment (i.e. the terrain) is fixed, the communication requirements are bounded by the number of simulation entities, not the size of the environment. Though these systems may be doing some actual physical simulations, because only a few "detail objects" in the world are actually changing, the amount of data being transferred is relatively small. Other systems are typically concerned with the physics of everyday object interaction, such as impenetrability and collisions [9, 6, 14]; they have been used to evaluate the ergonomics of environments like kitchens, automobiles, or work spaces. In these systems, simulations are typically limited to objects being directly manipulated, and the computations are simplified so that they can be done directly in the visualization environment without seriously loading down the computer.

On the other hand, many virtual-reality visualization systems have been built to allow the user to perform and interact with complex physical simulations, but they tend not to involve what we would consider "interactive simulation;" that is, the user is simply exploring precomputed data, without being able to interactively change the conditions under which that data was derived, and observe the results of their tampering. NASA's virtual windtunnel [2], in which airflow around a particular object is calculated, is a well documented example of this approach. An observer can enter a "black void" in which the object is suspended, insert "ink" sources to produce streamers along flow lines, and view the airflow computations from within the air space around the object. This system visualizes a precomputed computational fluid dynamics solution, and only allows the user to explore the space of the computed solution, without the ability to interactively modify the object or wind conditions for which the solution was generated.

The architectural community is very interested in full-scale interactive environmental simulation of planned environments from the point of view of an immersed human observer. Parameters of interest include lighting, temperature, and airflow throughout an entire building, and the computations can become very complex. Some architectural firms have constructed non-interactive, predefined video-tape visualizations comprising many moving people [18]. Realistic world simulation, where the environment itself is

changing based on a reasonable subset of physical and chemical laws, and under the possible influence of user-initiated changes to the scenario set-up, is a much more difficult task. Combining such simulations with immersive visualization by one or more active observers adds particular challenges with respect to synchronization and data management.

For systems that do offer interactive, real-time scientific visualization of complex simulations, the data transmission problem is well documented [3, 8, 10]. As the simulated system grows more complex, the amount of data needed to describe the full simulation state of the system in each time step can easily exceed the available bandwidth between simulator and visualizer. Efficient encodings, even lossy compression, have been employed to alleviate this communications bottleneck [8]. Another approach is to run the visualizer on the same (super)computer that performs the simulation, thereby hopefully gaining access to any needed data for visualization on demand in less than a frame time. However, this requires that the observer be physically close to the simulation engine, or that there exist a fast video link between the visualizer and the display screen used by the observer [9]. The video link approach also requires an extremely low-latency command line from the observer to the simulator to make the user's normal movements and interactions with the environment reasonably responsive. In such a set-up it might be more difficult to realize a collaborative environment in which individual observers can sign on at will from anywhere in the country at any time.

Densely occluded interior environments such as buildings, boats, planes, or caves offer certain advantages for immersive environmental simulation. They can take advantage of the same kind of preprocessing that has already been demonstrated in the context of visualization of static models [17]. Only those simulation results that affect the currently visible set of spaces need to be transmitted to the visualizer. A cell-based decomposition of the densely occluded world allows an effective estimation of a tight yet still conservative superset of the data which is absolutely necessary for visualization at any moment in time. As long as the number and complexity of the cells visible at any time remains bounded, the size of the whole world model can be, in principle, arbitrarily large – as long as there is sufficient (super)computer power to keep the ongoing simulation up-to-date.

# 3 PROBLEM FORMULATION

The first problem we faced was to combine two existing large and relatively well-developed programs into an integrated system that leaves room for growth and experimentation. We will now briefly introduce the two preexisting systems and define the key integration issues.

## 3.1 Walkthru and CFAST

The Berkeley Walkthru program was designed to support real-time interactive visualization of large (several million polygons), densely occluded building models at interactive frame rates (greater than 10 frames per second). To accomplish this goal, the Walkthru subdivides the "world" into rectilinear *cells*, connected by *portals*. In a preprocessing step, the system associates with each cell the set of all other cells that can be seen by an observer from any point within that cell. From this information, plus constraints on how quickly the observer can move through the database, the Walkthru can compute a set of cells for each frame that tightly, but conservatively, bound the set of cells visible in the next few frames. There are only two types of object in the Walkthru: "major occluders," which are two-dimensional wall, ceiling, or floor polygons, whose planes define cell boundaries; and "detail objects," which are 3D models of building contents (such as furniture and light fixtures), and which are associated with the cells that intersect the object's bounding box. During each frame, the detail objects and major occluders incident to any visible cell are drawn, and visibility is reevaluated from the new position. If the user wishes to voluntarily disallow changes in major occluders by the database editor and any in-use simulators during a visualization run, many visibility relationships can be precomputed for the database. Otherwise, the update rate of the visibility computations is easily quick enough to support relatively small-scale changes in the visibility structure of the world (i.e. punching some new holes in walls, or opening a new shaft in the floor or ceiling). In the last few years, Walkthru has provided a testbed for several applications including database construction [4], large scale radiosity computation [16], and scalable distributed walkthroughs with up to thousands of simultaneous users [11]. This technology can now be leveraged into support for distributed virtual environment simulation.

NIST's CFAST is the world's premier "zone model" fire chemistry and physics simulator. Similar to the Walkthru, it assumes an environment composed of rectilinear 3D regions (called "volumes") which are interconnected by portals (called "vents"). Within each volume, physical quantities such as gas species concentrations, raw fuel density, combustion byproducts, atmospheric pressure and temperature, and wall, ceiling, and floor temperature are tracked. A system of differential equations monitors the flow and exchange of these quantities through vents into adjoining volumes. Although CFAST's building partition concept is analogous to the Walkthru's cell structure, CFAST does not require similarly precise geometry. Volumes have a floor and ceiling height as well as length and width, but only the area of the volume (length times width) is relevant. Volumes are also not positioned in 3D space; only their size and height matters, and their connectivity through vents. Similarly, the exact X and Y location of the vents is irrelevant to the physics and is not represented; only orientation (horizontal or vertical) and cross-sectional area of the vent are needed, as well as the height at which it connects to the two prismatic volumes. As in the Walkthru, walls and floors are differentiated from "detail objects" such as furniture. Wall specifications include material and thickness information. The furniture database contains no geometry, but does include mass, materials, chemistry, and ignition and combustion detail curves for each type of object. Objects will ignite at predefined temperatures and burn as separate fires, producing appropriate physical and chemical effects on the environment. Other fire-related objects, such as sprinklers and HVAC ducts, affect the physics of the situation in realistic ways, but their only geometric component is positional information. Thus, the geometry of the CFAST situation can be derived from a Walkthru model, but the Walkthru model contains much more geometric information than CFAST represents; likewise, the unadorned Walkthru database contains none of the chemical, material, or "building systems" information (i.e. in-wall ductwork, piping, and wiring) needed by CFAST.

CFAST's main engine is a differential equation solver, computing flows of physical quantities and chemical species over time in the upper and lower parts of each volume. The formulation of the problem as a set of differential equations makes it feasible to create a parallelized version of CFAST, but this has not been done yet. CFAST provides large quantities of physical and chemical information, including concentrations of each of 10 chemical species, combustion products, temperatures of atmosphere, walls, floors, and ceilings, ignition times of objects, toxicology results, and many other physical and chemical quantities for each volume per time point.

While our system was designed specifically to integrate the Walkthru with CFAST, we attempted to make the combining framework sufficiently general to be useful for any environmental simulation one might want to do in a densely occluded world. Throughout this paper, we will refer to a generic "visualizer" and a generic "simulator;" for this project, the reader may infer "Walkthru" for visualizer

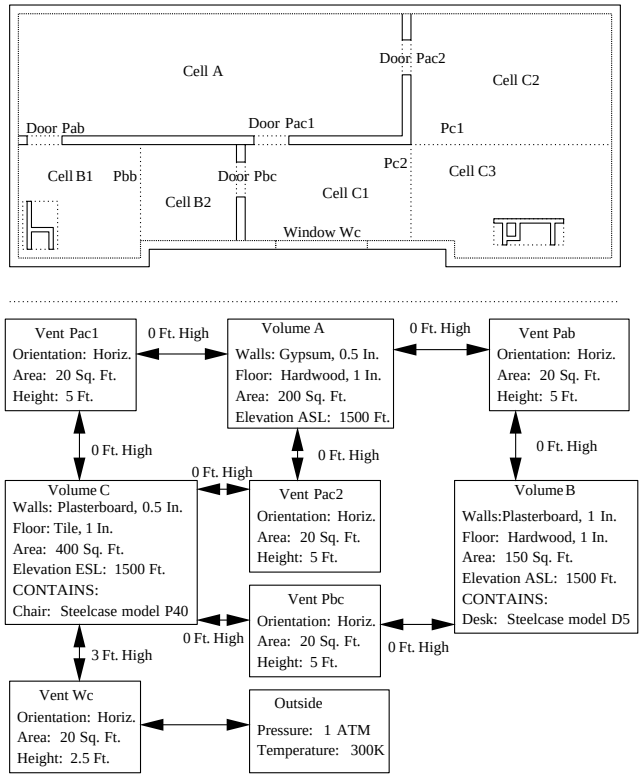


Figure 2: *How Walkthru (top) and CFAST (bottom) would "see" the same model. The Walkthru model contains detailed geometric information, but little else; the CFAST model is geometrically much simpler, but contains chemical and materials information that Walkthru lacks.*

and "CFAST" for simulator, but should keep in mind that the described framework is designed to be useful for other visualization and simulation engines.

### 3.2 Integration of Visualization and Simulation

Given CFAST and Walkthru, and with consideration to future visualization/simulation integration efforts, we are assuming the following model for our system:

We have a simulator and a visualizer, each of which operates on a cell-and-portal style environmental database. This database may be arbitrarily large, i.e., we could be operating on a building that will not fit into memory, and each of the two component systems can deal with the paging problem in its own way. However, due to occlusion, the visible "working set" of volumes will be tractable for any observer position. There is a mapping between the volumes of the visualization database and the simulation database, but the two are not be expected to be the same (i.e. a simulator "cell" might cover multiple visualizer "cells", or vice versa). Presently, we do not support arbitrarily complex geometric mappings between the two databases; we assume that one or more visualizer cells correspond to one simulator cell. We assume that the visualizer will transmit any setup information needed to begin simulation before issuing the start command. Furthermore, the visualizer may provide a front end by which the scenario being simulated may be changed on-the-fly. For example, the user may start a wastebasket fire in some room and then explore how the spread of the fire is influenced by opening or closing various doors or windows in the visualizer, thus repeatedly changing the situation being simulated. In such a case, the visualizer must transmit an update to the simulator in real time, and the simulator should recalculate previously computed simulation results that are affected by the change, as well as alter the course of the simulation in progress. CFAST explicitly supports opening and closing of vents at certain times in the model; however, we can make other interactive modifications by "restarting" CFAST in the middle of a run. We store the internal state of the solver at each time point, and, if necessary, "roll back" the simulator to the time of the modification by resetting the appropriate internal state if a change is made to the simulation conditions at a previously-computed simulation time. The solution is rather brute-force, as it requires complete recomputation of all conditions from that point forward. Hopefully in the future CFAST will directly support interactive modification without requiring discarding all simulation results past the time of the change.

Either one or both of the two component systems may be distributed, and may be operating on computers connected by anything from a LAN to a potentially high-latency, low-bandwidth network such as the Internet. We would also like to be able to attach and detach visualizers to a simulation in progress, to allow multiple observers to independently observe different portions of the data from the ongoing simulation. Each component system maintains its own world database during operation. The simulator generates data about subsequent world states observing relevant dependencies. CFAST operates with a fixed time step and produces its results in time slices that span all volumes in the database; these contain the current values of all the variables that are being tracked, and some derived quantities such as aggregate toxicity. Only a subset of that information will be of relevance to the visualizer at any particular time. We refer to a discrete piece of simulated information that is associated with one time slice and one spatial cell, a simulation "chunk." These chunks might be generated in different order depending on the demands of the visualizer.

The bottleneck in getting simulation data to the visualizer for rendering in real time may be in one of two places: either the simulator is too slow to generate data in real time, or the communication process between the simulator and visualizer has insufficient bandwidth to transmit the necessary chunks in a timely fashion. The simulation speed bottleneck is likely to hold for single-CPU simulations of reasonably sized databases; CFAST on a single 150MHz R4400 can only simulate about 16 cells (depending on degree of interconnection and density of furniture) in real time. Our goal in this situation is to increase the simulator's potential effectiveness by letting it know what areas of the world are of current interest to the visualizer. Specifically, the visualizer will inform the simulator of the currently visible cells and of the cells that may become visible in the very near future. The simulator can then concentrate on calculating and shipping the corresponding chunks with priority. In the near future, we expect simulator technology to improve; simulators will become faster, and their designs will evolve to provide better support for interactive visualization. Recent work has shown that this can be a promising approach for modeling the dynamics of physical structures [5]. In the specific case of CFAST, NIST is working on a version that will be able to concentrate its computational efforts on critical areas of the simulation, improving the speed and potential size of the simulation. We are also considering parallelizing the CFAST core for the Berkeley Network of Workstations (NOW) [1].

For the case where communication bandwidth is the bottleneck, the framework provides mechanisms that are easy to use and that optimally exploit the available bandwidth, while hiding communications concerns from the simulation designer. Of course, it is not possible to guarantee that all needed simulation chunks will be at the visualizer in time: the user might jump to a different part of the building or suddenly advance the time slider far into the future. To minimize the visible discontinuities associated with such a switch, we use a "just-in-time" chunk transmission scheme. Our scheme keeps

the communication channel in a state of near-starvation, allowing unanticipated "emergency" chunks to be sent through a nearly-empty transmission queue. This approach minimizes latency in the emergency case while still transmitting chunks at the highest possible rate for the channel.

# 4 KEY ABSTRACTIONS

The key primitives that define the interactions between simulator and visualizer are the *Simulation Data Set* and the *Real-Time Channel* over which this information gets exchanged. In this section we define these two abstractions.

## 4.1 The Simulation Data Set

In order to provide efficient data exchange between simulator and visualizer, we need a general structure for simulation data that can be easily managed and which is flexible enough to accommodate any information that a particular simulator may want to convey to the visualizer. This structure, called the *simulation data set*, which holds all simulation results, is organized in a three-level hierarchy as a set of sets. At the top level, it is indexed by simulation time. At the second level (i.e. within a particular timeslice) it is indexed by an identifier corresponding to one of the volumes into which the two databases are partitioned. At the third level, each spacetime volume contains a set of one or more integer-indexed subvolumes which together provide an arbitrarily sized data subspace for each volume. The leaf nodes of this hierarchy are the aforementioned "simulation chunks;" they are fixed-size data structures that represent part of the simulation output for a particular volume at a particular simulation time. The structure of a chunk is user-definable, so it can be easily modified to accommodate different simulator models. Because the system has a known mapping between simulator volume IDs and Walkthru cells, the visualizer can transmit desired simulation time and cell visibility information to the simulator, allowing the latter to determine exactly which chunks still need to be transmitted.

We do not currently manage distributed simulations, since the latest version of the CFAST code is unable to operate in parallel. However, assuming that any multicomputer simulator subsystem would be able to distribute the problem appropriately, the chunks generated by the separate simulators are easily recombined via simple set unions. Furthermore, since the subsystem controller knows how the problem is distributed, it should also be able to appropriately distribute the visibility lookahead data provided by the simulation manager.

## 4.2 The Basic Communication Mechanism

A simple and robust communications model is critical to both the performance and ease of use of a system that will be used to integrate a visualizer and a simulator for real time operation. Our communication model is based on a primitive we call a *real-time channel* (RTC). This is a 2-way, buffered, asynchronous mechanism that can operate in either a nonblocking polling or an interrupt-driven mode. Each channel has two separate 2-way byte streams: a data stream for most communication, and a command stream, intended to be used relatively infrequently, for user commands and simulator status packets that need to arrive quickly. The interface to a channel is independent of the specific low-level mechanism used (currently either Internet- or Unix-domain sockets), and provides the ability to send either single-integer "commands" or arbitrary-length "packets" across either of the two streams. A server mechanism is provided that allows a simulator to open a server port on a machine and wait for connections, which will launch an instance of a simulator connected to an instance of a channel. Channels can be opened locally or over a network; the appropriate low-level protocol is automatically selected by the system when it connects.

# 5 PROGRAMMING, INTEGRATION, AND USER INTERFACES

With the data format and the basic communication mechanism defined, we now look at the system's interface and functionality from the point of view of both visualization designer and simulation designer.

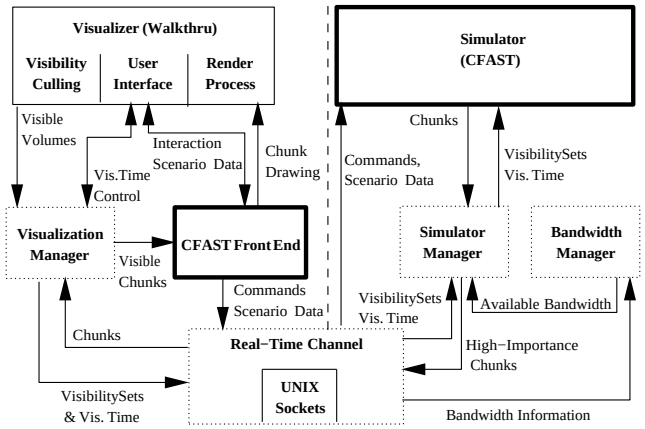


Figure 3: *A diagram of how the system components connect simulator to visualizer. Components in bold outline are created by the simulator designer; components in dotted outline are provided by the integration framework.*

## 5.1 Visualizer User Interface Constructs

The visualizer's user interface needs to allow the user to connect and disconnect the simulator, as well as to control the progress of "visualization time." The selection of which volumes are being visualized is determined simply by "walking" to the appropriate area. We provide a standardized simulator controller consisting of a panel with simulator connection and status controls, a time slider bar that covers the timespan of the currently running simulation, and a set of VCR-style controls (play, reverse play, fast forward, reverse, and pause) that allows the user to control the rate at which time passes. The slider bar may be directly manipulated to change the current viewing time to any desired value; the VCR controls alter the "time velocity" of the user in simulation time (Play is velocity 1, Fast Forward and Rewind are 10 and -10 respectively, Pause is velocity 0, etc.). The portion of the slider corresponding to data that has been computed by the running simulation is colored green; the portion corresponding to the as yet unsimulated timespan is colored red. This provides immediate feedback to the user about how far the simulation has progressed. The slider is prevented from entering the red region.

The controller also includes a tool intended to mitigate the inherent "burstiness" of most simulations, including CFAST. This tool, called the *autopause* mechanism, will automatically "pause" the visualization time in two cases. At the beginning of the simulation run, it allows the simulator to get a certain distance ahead of the current visualization time, in order to provide a buffer of data that allows the visualization to proceed smoothly if the simulation output becomes bursty. Second, at any time, if the visualization time

"catches up" to the simulator, the pause is engaged in the same fashion. In either case, when the simulation provides enough of a buffer, the pause will automatically be removed and visualization time will once again move forward.

### 5.2 Establishing a Connection

There are two mechanisms for establishing a connection between a simulator and visualizer. A local connection can be established by the visualizer forking a simulator process, which is connected via a local channel. Alternatively, the simulator can establish a server on either a local or remote machine, and the visualizer can connect to it. Once a connection is established, negotiation and setup are handled by the simulator-specific visualization front-end and the simulator being run. In the case of CFAST, the CFAST visualization front-end within the Walkthru transmits the simulation scenario across the channel in the form of a native CFAST data file, which it can create and/or interpret from the Walkthru model geometry, registration information, and chemistry data provided by the user. Negotiation is specific to the particulars of the simulation being run, so the only mechanism provided is the communication channel. When both sides are ready to begin, they each establish their respective "manager" (either a visualization manager or a simulation manager). With the managers attached to the channel, the simulator may begin generating data and submitting it to the simulation manager, and the visualizer can begin visualizing simulation data provided by the visualization manager, each side trusting the managers to handle the communication process. Attaching the managers does not restrict use of the channel; "out-of-band" information such as condition changes and commands from the visualizer can pass through the managers and across the channel, where the simulator can account for these changes by adjusting or re-simulating all or part of the current run.

### 5.3 Visualization Manager and API

The visualizer's access to simulation data is controlled by the visualization manager and the simulator's visualization "front-end;" the former is a standard component of our architecture, while the latter must be provided by the simulation designer for each simulator to be integrated into the system. When launched, the visualization manager establishes a connection to a simulation manager on the other end of a channel, as well as registering a callback with the visibility lookahead system of the Walkthru. The simulator's visualization front-end launches both the simulator and the visualization manager, and must provide the visualization manager with a starting visualization time and time velocity; these values are typically derived from the simulation scenario. The visualization manager keeps the simulation manager apprised of both the currently visible set of volumes (from the renderer's visibility system) and the current visualization time and velocity; this is done via corrections which are transmitted across the command stream of the channel whenever the time, time velocity, or lookahead set changes.

The simulator visualization front-end provides the user with both direct and indirect control of the simulator and visual interpretation of the simulation results. It may include arbitrary user interface controls for the simulation scenario, and it is required to include a rendering function for simulation chunks. The rendering function accepts a Walkthru database cell and a set of chunks describing the current conditions in the cell, and renders the chunks' contents into the GL window. During each frame, it is called with all visible cells in the frame that have simulation chunks associated with them at the current visualization time. It is never called for a cell or simulation chunk that is not visible in the current frame; this provides efficient, rapid rendering of simulation conditions. The front-end is also provided with hooks into the visualizer's event processing system and is required to interpret any user interactions that might affect the ongoing simulation scenario. If such an interaction happens, the necessary changes to the scenario are transmitted to the simulator, and, by default, all simulation chunks from that simulation time forward are invalidated. The simulator then has the option to either re-validate (i.e. tag valid without regenerating) or regenerate any portion of that data.

CFAST simulation chunks come in two types. The first type contains temperature, energy output, location, and fuel conversion rate of one particular fire; there can be many of these in one spacetime volume, corresponding to active fires from individual fuel sources such as pieces of furniture. The second type, of which there is only one per spacetime volume, contains the chemical concentrations of nine different gases, fuel concentration, atmospheric pressure and toxicity level, and smoke interface height for the volume as a whole. The user can select a "natural" viewing mode or one of several "scientific visualization" modes. In the natural mode, the drawing function renders the smoke interface with texture-mapped smoke flowing in the appropriate direction. If there are active fires in the volume, a texture-mapped, animated flame is drawn at each flame location with a physically accurate height and base area. If the user is actually inside the smoke, hardware-enhanced fogging of the viewing volume is used to simulate visibility attenuation. A side panel indicates the chemical composition of the atmosphere; an alarm sounds when the user experiences toxic chemical levels. These panels and signals help the user determine what sort of conditions would be experienced by someone inside the burning building. Alternatively, one of the scientific visualization modes can be used. For example, if the user is interested in temperature, "infravision" can be activated; the observer can look at the walls, floor, or smoke, and the surfaces are pseudo-colored according to their temperature. Thermal distributions on the walls can be directly viewed, and the spread of heat can be observed. Selection of appropriate visualization modes is an ongoing research topic of this project. Good combinations of visualization and realistic rendering are being explored to provide maximum information transfer to the user.

### 5.4 Simulation Manager and API

The simulation manager allows the simulator to generate simulation data as rapidly as possible without worrying about how that data is being transmitted to the visualizer. Once the manager is engaged, the simulator simply generates data, and "submits" the data in the form of simulation sets to the simulation manager. Submissions may be made for any timeslice or volume ID; if a simulation chunk is submitted for the same time, volume ID, and subvolume ID as a previously submitted chunk, it supersedes the older chunk. In this way, a simulator may "retract" any subportion of the previously generated data that is incorrect or that was generated as a quick approximation to be improved later. This will generally happen when the user makes a change in the environment at a particular simulation time, rendering many or all of the chunks after that time invalid.

If the simulator has the necessary capabilities, it may request the current set of visible volumes and the current visualization time from the simulation manager, and selectively generate or improve the corresponding simulation data to ensure that the visualization can proceed without pausing. We believe that this will be an important feature of future simulators that intend to provide visualization data in real-time while operating on very large databases.

## 6 SIMULATION DATA MANAGEMENT

We have stated that the visualizer and simulator can rely on the visualization manager and simulation manager to get the simulation data to the visualizer in time to be viewed. In this section, we explain how this is implemented via "just-in-time" data management.

## 6.1 "Just-In-Time" Simulation Data Management

In order for the visualization manager to ensure that the appropriate simulation chunks are either already present or en route from the simulation manager, it has to provide the simulation manager with enough information to determine which chunks are most critically needed. To do this, we define an "importance function" over spacetime, in which the chunks associated with spacetime cells of higher importance will be transmitted to the visualizer earlier. Clearly, the spacetime cells that are visible to the user at the current visualization time are the most important ones, and are needed immediately by the visualizer. Given the user's location, maximum velocities in space and time, the current visualization time, the current visualization time velocity, and the preprocessed volume visibility information from the Walkthru's cull process, we can compute for each spacetime cell the earliest real time in the future in which the user might be able to see that cell. This defines the desired function; smaller "earliest-possible-time-to-visibility" values correspond to higher importance. The information needed to compute this function is available to the visualization manager, which is directly linked to the visualizer; one of the visualization manager's tasks is to transmit this information to the simulation manager, which evaluates the importance function over the set of chunks generated by the simulator, and thereby determines which unsent chunks are most important at any given time.

Our current system does not support the full computation of this function. We implement a heuristic approximation by maintaining a *visibility set* and an up-to-date visualization time at the simulation manager. The visibility set contains the set of Walkthru cells that are either currently visible to the observer, or may be visible in the next several frames. This information is normally computed as part of a Walkthru frame. The visualization manager monitors the visibility set and transmits an update to the simulation manager if the set changes from one frame to the next. Similarly, the visualization time and time velocity are updated if the user alters the time velocity or moves the time slider. Note that, though the visualization time changes as real time passes, the simulation manager can keep accurate track of the current visualization time without continuous updates from the visualization manager; updates are only necessary if the user manipulates a control setting.

The simulation manager then assigns highest importance to the transmission of chunks that are in the visible set and whose time is closest to the current visualization time in the direction of the current time velocity. The next highest importance is assigned to chunks in the visibility set in the *opposite* direction of the current time velocity, since the user often wants to review preceding time slices in the current location to find out how the situation has evolved. All other chunks are of tertiary importance. This corresponds to an approximation of the "ideal" importance function discussed above for very high values of time velocity; it can be computed quickly and does not require the full visibility information of the Walkthru's visibility processing. The simulation manager uses the communications channel to transmit those chunks that have not already been sent and are of highest importance as denoted by the heuristic function.

A sudden change in the time, time velocity, or visibility set can result in a need to get a new set of chunks to the visualizer as quickly as possible. If the user has been visualizing simulation time $t_s = 10$, for example, and the time slider is moved to $t_s = 200$, the simulation manager may have this data, but it is unlikely that the data has been transmitted already. In this case, the simulation manager *immediately* evaluates the most critical chunks to be sent to the visualizer, and transmits those chunks as soon as possible.

It is interesting to note that limiting the user's maximum "time acceleration" (i.e. disallowing direct manipulation of the time slider, allowing the user to move in time only with the VCR buttons) has the effect of allowing us to compute a "time lookahead" to go along with the visibility lookahead. This means that we can establish a tight superset of the number of spacetime chunks that might be visible in the next few seconds of real time. Without such a bound, the potentially visible set from one frame to the next includes the set of potentially visible cells for *all* timeslices of simulation data available, because the user can drag the time slider from any point to any point within one frame time. With such a bound, and a bound on the number of chunks that will be submitted per spacetime volume (which is easy to derive for most simulators, including CFAST), we can compute a minimum required bandwidth so that we can *guarantee* that all of the needed chunks will be available if there has been at least enough time since the chunk's submission to overcome the latency of the communication channel.

If memory is limited on the visualization machine, it is possible for our system to run the visualization manager as a *cache*, rather than as an *accumulator* of the entire simulation data set. In this case, the visualization manager is allowed to "throw out" old or not recently used chunks. The visualization manager reports to the simulation manager which chunks have been discarded, so that they may be retransmitted if they need to be viewed again,. In the case of very large precomputed data sets, the simulation manager can also be run on a local machine, managing access to a huge disk file instead of an active simulation, while the visualization manager manages the set of simulation data being cached in memory.

## 6.2 Bandwidth Management

Given only the importance function on the set of simulation chunks that have been submitted, there is no indication of *how much* data should be sent by the simulation manager per unit time. Because the channel is buffered, if no bandwidth usage constraint is enforced, then every time some conditions are submitted by the simulator, all of that data could be queued for transmission through a channel which will not be able to actually finish transmitting that data for quite some time. In a priority situation, when the importance function has changed due to user input, and a different set of chunks are needed *immediately*, queued "old" chunks would delay the transmission of urgent data until those older chunks had drained through the pipe. This "clogging" reduces or eliminates the system's ability to respond to sudden changes in visibility or time. Unfortunately, with most physical simulators, this situation would occur fairly often; physical simulators, including CFAST, tend to exhibit "bursty" output corresponding to sets of solutions for conditions across a slice of time for the entire model. If we use our importance function to determine which chunks are to be sent, the situation becomes even worse; sudden changes of the user's time or position generate even larger spikes, as new, potentially huge sets of chunks become highly important when the user walks into a new region of the database.

An early solution we tried for this problem is to include a priority bypass which provides the ability to interrupt the channel's normal input queue with a second queue of chunks which are to be transmitted first. In an interactive system, this priority bypass often proves ineffective, due to the fact that *two* of the aforementioned sudden changes in the importance function could cause the system to send priority data down an already busy priority channel, and the more recent priority packets, which are now more critical, are delayed in the same fashion that the one-channel strategy delays the first set of priority packets. The situation is made worse in larger databases; in the unmanaged condition the size of these spikes grows with the size of the database. Adding bypasses on top of bypasses quickly becomes unwieldy and inefficient; once all of the data is sitting in multiply-bypassed queues, control of transmission order becomes impossible, the amount of storage needed for redundant queues quickly becomes prohibitive, and the work needed to override a chunk which has been regenerated by the simulator grows without bound.

The core of the problem is the inherent buffering of data in the communication channel. This buffering is unavoidable due to its

ubiquity in the low-level communication structures provided by the operating system and the network itself, which use buffering to optimize throughput. Unfortunately, the more buffering there is in the channel, the larger the potential latency for a high-priority packet to be transmitted through the channel; since guaranteed-receipt network protocols guarantee arrival in order of transmission, every bit of buffered data in the channel must clear the channel before our high-priority packet can get through. Thus, we would like to operate the channel in a near-starvation mode, which simultaneously minimizes buffering while using all or nearly all of the bandwidth to transmit useful chunks as quickly as possible. This job is handled by our *bandwidth manager*, which closely controls the speed at which the simulation manager is allowed to transmit chunks to the visualization manager. Available bandwidth is currently specified to the bandwidth manager in total kilobytes (kb) per second. The bandwidth number should be selected to closely approximate real bandwidth (i.e. on two machines on an Ethernet, bandwidth might be on the order of 1,000 kb per second, whereas two machines connected by 28.8 kilobaud [kbps] modem would only be able to manage about 3 kb per second). Several times a second, the bandwidth manager "wakes up" and gives the simulation manager permission to transmit another $x$ kb worth of simulation chunks on the data stream, where $x$ is the given bandwidth divided by the manager's wakeup frequency. When this happens, the simulation manager selects $x$ kb worth of chunks from the unsent chunk pool in order of importance, and gives those chunks to the channel for immediate transmission. By the time the manager wakes up again, all of the submitted chunks should have cleared or nearly cleared the channel; thus, if an emergency situation happens while the manager is asleep, when the manager next wakes up, the most important chunks will be transmitted on a nearly empty channel, which minimizes the transmission latency for those chunks. At the same time, if no emergency occurs, the channel is still being utilized at nearly its maximum capacity, with the next most important set of chunks being sent "just-in-time" for the channel to have completed transmitting the last set; clogging cannot occur if the bandwidth estimate is accurate or conservative.

In our current system, the bandwidth manager's settings are provided by the user. In the future, we intend to have the bandwidth manager dynamically determine via feedback how much bandwidth is available in the pipe, and scale its notion of available bandwidth appropriately [12].

### 6.3 Performance

Figures 4 through 6 show a typical example and comparison of the performance of three strategies for data management. The most basic is the naive, *oldest-data-first* strategy (figure 4A) which simply queues timeslice data into the communication channel as it becomes available. The second is the *visibility-guided* strategy (figure 4B), in which simulation data is transmitted only for visible or almost-visible volumes (i.e. in order of the basic heuristic importance function), but with no bandwidth management, so that it queues *all* unsent available data for the visible volume set after a change in visualization time or the visible set. The third strategy is our full *bandwidth-managed-importance* strategy (figure 4C), which incorporates all of the subsystems mentioned in this paper. The data was gathered from our instrumented RTC package during identical prerecorded runs of both the visualizer and simulator, in which all simulation data generation, user motion, and manipulation of the time slider and VCR controls were recorded and reproduced in exactly the same way for each run. The communication bandwidth was artificially reduced to 3 kb/s for these runs in order to demonstrate the difference between the strategies; at present, our largest test case is insufficient to stress the switched Ethernet in our office. The reader may wish to note that this bandwidth was selected to correspond to that available from a 28.8 kbps modem link.

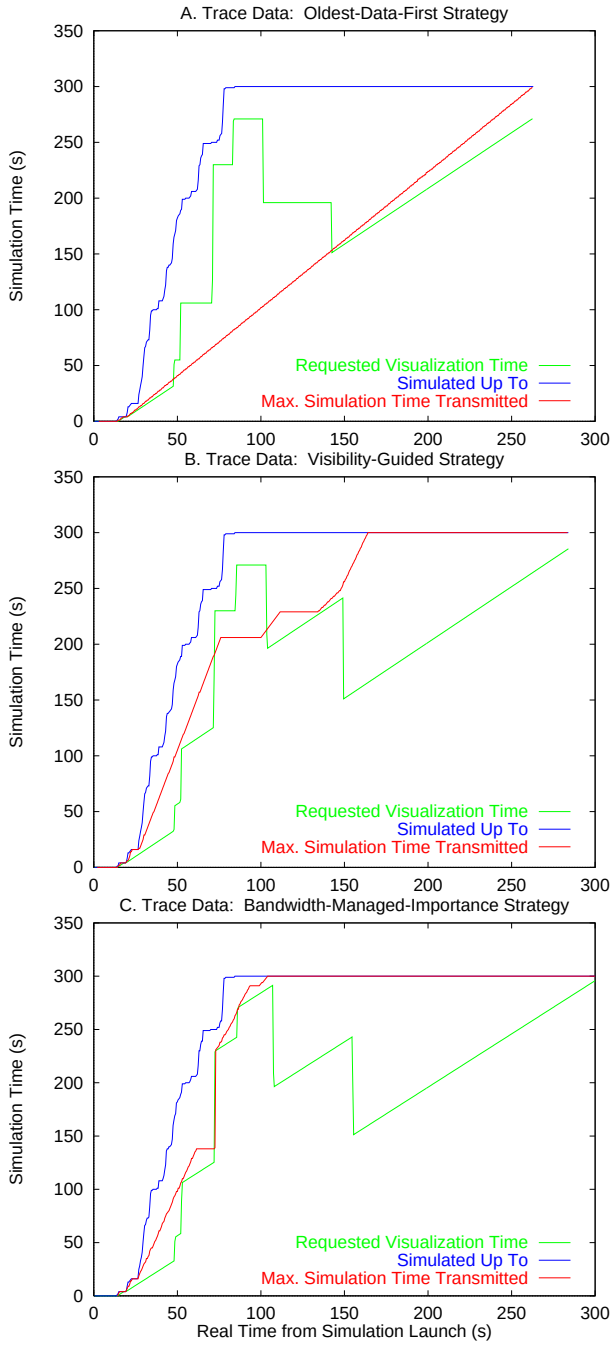


Figure 4: *Trace data of simulator-visualizer data transfer for three strategies: the oldest-data-first strategy (top), the visibility-guided strategy (middle), and the bandwidth-managed-importance strategy (bottom). The horizontal axis is real time; the vertical axis is simulation (i.e. virtual) time. Three functions are plotted for each strategy: the amount of simulation time completed by the simulator, the viewer's current visualization time, and the timestamp of the latest chunk that has been transmitted from simulator to visualizer. Note the vertical lines in the requested visualization time, which denote user-created time discontinuities, and the horizontal lines in the requested visualization time, which show regimes for which data is available from the simulator, but for which that data had not been transmitted in time to be viewed. The "maximum simulation time transmitted" curves give an indication of how responsive each strategy is to user movement in space and time.*

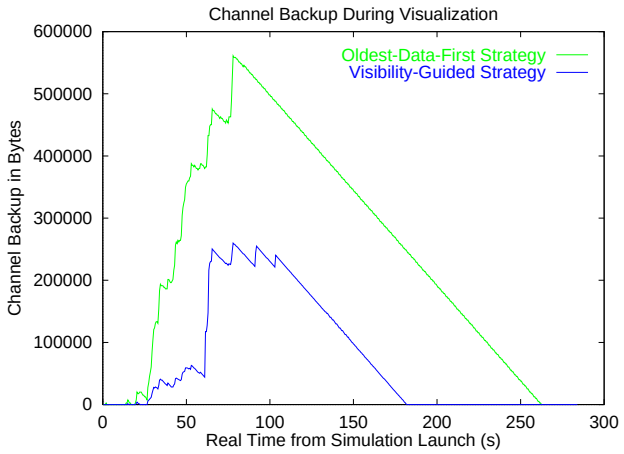


Figure 5: *Trace data of communication pipe backup (i.e. clogging) for the oldest-data-first and visibility-guided strategies. The former is much worse than the latter, although it is in the latter that it actually makes a difference. Pipe blockage in the bandwidth-managed-importance case is negligible (less than 1.5 KPS on this graph, where the others peak at about 550 KPS and 250 KPS respectively), and can in fact be reduced to an arbitrarily small amount on a fast computer by increasing the manager's callback frequency.*

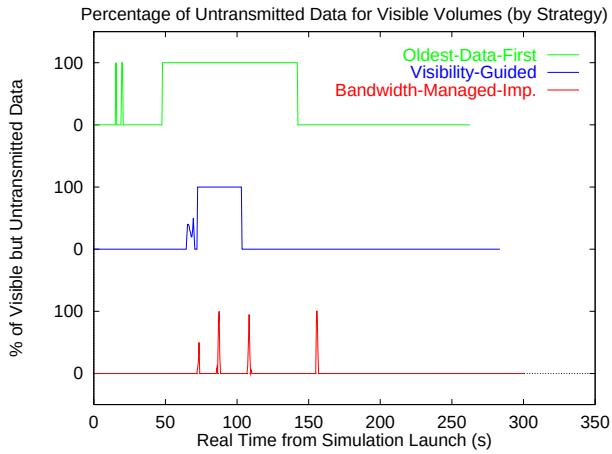


Figure 6: *Trace data of the percentage of spacetime volumes visible to the user which have simulation data available, but for which that data has not yet been transmitted to the visualizer. The oldest-data-first strategy exhibits massive gaps in viewable data; the visibility-guided strategy fares better, but there is still a 40-second period where the user should be seeing smoke and flame, but instead sees nothing. The bandwidth-managed-importance case shows only brief, 1- to 2-second gaps at time discontinuities (i.e. where the user unpredictably drags the time slider far into the untransmitted data).*

The three graphs in figure 4 compare three functions of real time for each strategy: how far the simulator has progressed through the simulation, labeled *Simulated Up To*; the latest simulation time for which simulation data has actually been sent to the visualizer (i.e. the maximum possible viewable simulation time at the visualizer), labeled *Max*. *Simulation Time Transmitted*, and the simulation time currently being requested by the user within the Walkthru, labeled *Requested Visualization Time*. If bandwidth were infinite, the user should be able to "see" simulation results whenever the requested visualization time is less than the simulated-up-to time, and the maximum simulation time transmitted would be identical to the simulated-up-to time (which is the most recent simulation time available from the simulator). Under bandwidth limitations, however, it may be that the requested visualization time is less than the simulated-up-to time, but the data is not yet available (i.e. the maximum simulation time transmitted is *less* than the requested visualization time) due to failure of the communication channel to transport the needed data. The most visible evidence of this in the graphs is where the requested visualization time becomes a horizontal line, indicating that the autopause mechanism has engaged due to the visualizer not having the needed data (resulting in a zero time velocity and unchanging visualization time). Many such flats are seen in the case of the *oldest-data-first* strategy; the channel is far too narrow at 3 kb/s to transmit the data in time. The *visibility-guided* strategy does better toward the beginning of the run, where the fact that few volumes are visible allows it to get more timesteps to the visualizer, since those steps contain fewer volumes. However, when the user walks out into the hallway at time 70, the new set of visible volumes results in a deluge of newly important data being queued into the channel. When the user proceeds to move the time slider at time 83, the channel is clogged with the (as yet unsent, but already obsolete) "priority" data from the hallway transition, and the system is unable to respond, resulting in a period of no data being visible. A graph of the communication channel blockage per unit real time (figure 5) shows that a large "clog" occurs at time 70 in the visibility guided case; this clog is what prevents the adaptation to the time discontinuity at time 83. The *oldest-data-first* strategy shows much more extensive clogging. Ironically, since the naive strategy has no ability to adapt to changing conditions anyway, the clogging is somewhat moot.

In the *bandwidth-managed-importance* case, the system adds the data to the pipeline a little at a time, never adding enough to clog it for more than a fraction of a second; when the user enters the hallway, the system immediately switches to transmitting the needed data for the hallway, and when the time slider is moved, a similar switch is performed that sends the data for the new time. Since the pipe is never clogged, the needed data can be transmitted quickly in subsequent emergency situations.

Figure 6 shows the crucial data of concern; that is, the percentage of volumes per unit real time that are visible to the user and for which simulation data has been generated, but for which that simulation data has not been transmitted yet. The top curve shows that the *oldest-data-first* strategy spends almost half of the simulation run in this state; the viewer is looking at blank volumes when they should be seeing smoke. The center curve shows that the *visibility-guided* strategy does well until the second discontinuity at $t = 83$, at which point it breaks down as well; however, it recovers just after $t = 100$, whereas the *oldest-data-first* strategy doesn't recover for another 20 seconds. Finally, the *bandwidth-managed-importance* strategy is shown at the bottom. The spikes are 1 to 2 seconds long, and are only present at gross visualization time discontinuities (compare the locations with the vertical jumps in visualization time in figure 4). This corresponds closely with the minimum response latency for an emergency; it takes about 1 second just to transmit the data for a timestep at 3 kb/s, and the bandwidth manager makes new bandwidth available once every 0.5 seconds in our system, so the total response latency in our test case has a lower bound of 1 second, and an expected time of 1.25 seconds if the channel had absolutely no latency (which is the ideal case). Thus, our bandwidth manager approaches ideal performance under these conditions. If we bound the user's time velocity, we can further reduce the frequency of these spikes; with a bandwidth of about 30 kb/s, we could eliminate them entirely for any user manipulation of the VCR controls (i.e. any time velocity under 10 virtual seconds per real second). This claim cannot be made for the other two strategies.

## 7 CONCLUSION

We have developed an example of a simulation based environment for the study of fire hazards and their environmental effects in buildings. An existing simulator, NIST's CFAST, and an existing visualizer, the Berkeley Building Walkthru program have been integrated via a new general framework.

To achieve the most effective simulation for real-time visualization, we have exploited the "cell-and-portal" data structures of both systems. The resulting spatial subdivision and localization of the overall world allows the visualizer to concentrate the expensive rendering task on only those cells that are possibly visible to the observer. With a suitably structured simulator, preference could also be given to simulating the environment in the neighborhood of the observer; but even with an unstructured simulator that produces data in a strictly time-ordered fashion, the cell-based subdivision permits an efficient grouping and prioritizing of the simulation data for its transmission to the visualizer over a potentially bandwidth-limited, high-latency network.

The integration of CFAST into the Walkthru visualizer makes the results of this powerful fire simulator much more understandable at several levels and allows the user to interfere with and redirect the ongoing simulation. Previously, CFAST operated in a batch mode, and provided limited, one- and two-dimensional graphs and data dumps that had to be pieced together with substantial cognitive effort. The use of 3D computer graphic techniques employing suitable symbolic visualizations permits scientists to perceive several variable values such as temperature, smoke levels, or air toxicity in a parallel and quantitative manner. On the other hand, using more natural rendering techniques showing flickering flames and drifting smoke clouds, gives even lay-persons an intuitive understanding of the environmental phenomena being simulated. Such realistic-looking virtual worlds offer the promise of practicing fire-fighting and rescue strategies without any physical danger to the trainee.

The abstractions, integration mechanisms, and high-level data structures that define the interface between the CFAST fire simulator and the Walkthru visualization program are also applicable to other virtual environments combining one or more simulators with first-person interactive visualizers in densely occluded worlds. The demonstrated programming interfaces will also allow rapid prototyping and integration of other environmental simulations into the Walkthru.

## Acknowledgments

The authors would like to thank Walter Jones of the Building Fire Research Laboratory for intellectual guidance and technical support for CFAST, and Professor Patrick Pagni of Berkeley, for encouraging us to undertake this project. This work was primarily supported by NIST under contract number 60NANB5D0082, and received further financial assistance from Lockheed-Martin Missles and Space, NEC, and ONR MURI grant N00014-96-1-1200.

## References

[1] Anderson, T.E., Culler, D.E., and Patterson, D. A Case for Networks of Workstations: NOW. *IEEE Micro* 15:1, Feb. 1995, pp. 54-64.

[2] Bryson, S. The Virtual Windtunnel: A High-Performance Virtual Reality Application. *IEEE Virtual Reality Annual International Symposium* (Seattle, WA, Sept. 1993), pp. 20-26.

[3] Bryson, S. Virtual Reality in Scientific Visualization. *Communications of the ACM* 38:5, May 1996, pp. 62-71.

[4] Bukowski, R.W. and Séquin, C.H. Object Associations: A Simple and Practical Approach to Virtual 3D Manipulation. *Proc. of the 1995 Symposium on Interactive 3D Graphics* (Monterey, CA, April 1995), pp. 131-138.

[5] Chenney, S., and Forsyth, D. View-Dependent Culling of Dynamic Systems in Virtual Environments. *Proc. of the 1997 Symposium on Interactive 3D Graphics* (Providence, RI, April 1997), pp. 55-58.

[6] Cohen, J., Lin, M., Manocha, D., and Ponamgi, M. I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments. *Proc. of the 1995 Symposium on Interactive 3D Graphics* (Monterey, CA, April 1995), pp. 189-196.

[7] Cremer, J., Kearney, J. and Ko, H. Simulation and Scenario Support for Virtual Environments. *Computers and Graphics* 20:2, March/April 1996, pp. 199-200.

[8] DeFanti, T.A., Sandin, D.J., Lindahl, G. et. al. High Bandwidth and High Resolution Immersive Interactivity. *Very High Resolution and Quality Imaging* (San Jose, CA, 1996), pp. 198-204.

[9] Doi, S., Takei, T., Akiba, Y., et. al. Real-Time Visualization System for Computational Fluid Dynamics. *NEC Research and Development* 37:1, Jan 1996, pp. 114-123.

[10] Funkhouser, T.A., Séquin, C.H. , and Teller, S.J. Management of Large Amounts of Data in Interactive Building Walkthroughs. *ACM SIGGRAPH Special Issue: 1992 Symposium on Interactive 3D Graphics*, March, 1992, pp. 11-20.

[11] Funkhouser, T. A. RING: A Client-Server System for Multi-User Virtual Environments. *Proc. of the 1995 Symp. on Interactive 3D Graphics* (Monterey, CA, April 1995), pp. 85-92.

[12] Gong, K. and Rowe, L.A. Parallel MPEG-1 Video Encoding. *Proc. 1994 Picture Coding Symposium* (Sacramento, CA, September 1994).

[13] Macedonia, M.R., Brutzman, D.P., Zyda, M.J. et. al. NPSNET: A Multi-Player 3D Virtual Environment over the Internet. *Proc. of the 1995 Symposium on Interactive 3D Graphics* (Monterey, CA, April 1995), pp. 93-94.

[14] Mirtich, B., and Canny, J. Impulse-Based Simulation of Rigid Bodies. *Proc. of the 1995 Symposium on Interactive 3D Graphics* (Monterey, CA, April 1995), pp. 181-188.

[15] Peacock, R.D., Forney, G.P., Reneke, P. et al. CFAST, the Consolidated Model of Fire Growth and Smoke Transport. NIST technical note 1299, U.S. Department of Commerce, Feb. 1993.

[16] Teller, S., Fowler, C., Funkhouser, T. and Hanrahan, P. Partitioning and Ordering Large Radiosity Computations. *Proc. of SIGGRAPH 94* (Orlando, FL, July 1994), pp. 443-450.

[17] Teller, S.J., and Séquin, C.H. Visibility Preprocessing for Interactive Walkthroughs. *Proc. of SIGGRAPH 91* (Las Vegas, Nevada, Jul. 28-Aug. 2, 1991). In *Computer Graphics*, 25, 4 (Jul. 1991), pp. 61-69.

[18] Taisei corporation. "Yebisu Garden Palace." Video, 1994.

[19] Zyda, M.J., Pratt, D.R., Monahan, J.G. and Wilson, K.P. NPSNET: Constructing a 3D virtual world. *ACM SIGGRAPH Special Issue: 1992 Symposium on Interactive 3D Graphics*, March, 1992.