Citywalk: A Second Generation Walkthrough System

Richard Bukowski, Laura Downs, Maryann Simmons, Carlo Séquin, and Seth Teller[†]

U.C. Berkeley, [†]MIT

[bukowski,laura,simmons,sequin]@cs.berkeley.edu, [†]seth@graphics.lcs.mit.edu

Abstract

The architectural framework of an advanced virtual walkthrough environment is described and placed in perspective with first generation systems built during the last two decades. This framework integrates support for scalable, distributed, interactive models with plug-in physical simulation to provide a large and rich environment suitable for architectural evaluation and training applications. An outlook is also given to a possible third generation of virtual environment architectures that are capable of integrating different heterogeneous walkthrough models.

1. Introduction

Over the last few decades, the evolution of powerful personal workstations equipped with advanced graphics hardware has given us low-cost systems on which fairly complex virtual worlds can be explored at interactive speeds. The earliest such “walkthrough” systems date back to the 1970’s and were designed to provide real-time flight simulation. These early systems were struggling to address basic problems in virtual environment visualization, and they were often pushing the bounds of contemporary hardware or algorithms to achieve their goals. We refer to these systems as *first generation*. In the last few years, advances in performance and reductions in cost are finally providing enough system resources to generalize and merge these varied techniques into a coherent whole. This creates a new set of structural and theoretical problems, as these disparate techniques all have their own requirements and may interact with each other in complex ways. The promise of these *second generation* systems is to provide greatly enriched interactivity and realism with large numbers of users on very large, distributed models.

1.1. First generation systems

Existing first-generation walkthrough systems can be roughly split into two general types: indoor and outdoor environments. While their application domains differ, all of these systems achieve scalable performance by partitioning the model such that only a relatively small portion of the database needs to be resident in memory at one time (visibility culling) and applying level of detail (LOD) abstractions to those elements that are visible to reduce rendering time.

The first outdoor virtual world applications were flight simulators. Their development resulted in pioneering work in levels of detail and abstraction that allowed a large, complex landscape to be loaded and simplified in real-time such that it was renderable on contemporary graphics hardware. The large scale of the environment in these systems was handled by combining locality-based databases (i.e. the world is “tiled”, and users see only the tile they are over, plus the adjacent tiles) with LOD techniques to simplify distant geometry and objects. Recently, virtual city walkthrough projects such as the Virtual Los Angeles Project [14] explored more advanced database techniques for streaming large models to clients, as well as new simplification techniques such

as impostors [15]. Because the user is in close proximity to the buildings, these systems must use much more complex world models than flight simulators to provide adequate visual detail. NPSNet [26] is an example of an outdoor environment that focuses on a large number of users in a world with high interactivity. These high-performance interactive systems provide a high-speed communication layer using IP multi-cast to provide rapid distribution of the state of users and other entities within localized cells. Recent commercial multi-player on-line computer games involving thousands of simultaneous users such as Ultima Online or Everquest attempt similar interactions.

Indoor environments typically found in architectural walkthroughs are treated separately because they have a densely occluded structure that lends itself to various forms of strong portal culling [12, 1]. These environments pose conceptually similar problems in database management and rendering complexity but rely less on LOD and instead on a more powerful set of culling techniques. Systems such as the Berkeley Walkthru program [12] and the University of North Carolina architectural walkthrough systems [2] have proven that systems using on-disk or on-network object databases, combined with integrated LOD abstractions, prefetching, and user motion prediction, can provide interactive visualization of very large, complex architectural models. Funkhouser's RING system [10] provides distributed multi-user functionality in the indoor domain, using a system of central servers with both high-speed interconnections and higher level geometric information about world structure. This allows the system to distribute the same information more intelligently, and limit the amount of data that is transferred to individual client machines based on client regions of interest. The approach also improves on the multi-cast techniques in that it works better for clients with slow and/or nonlocal network links, which is a problem for the IP multi-cast systems used in outdoor databases.

1.2. Shortcomings of first generation systems

While first generation systems provided basic tools and solutions to many of the fundamental problems facing specific virtual environment applications, they were usually unconcerned with the way in which these tools and solutions inter-operated. For example, none of the aforementioned systems allows the model to be changed by the observer in any meaningful way at run-time. Such changes would result in a need to recompute sections of the database structure; this often involves complex precomputations that would be slow and difficult to distribute to the other affected clients. Indeed, the model environment itself is typically not even centrally served to the users; almost all of these systems use a replicated world database that must be present in its entirety at each client when the simulation starts. Only a few systems have also distributed world state [23, 14] and those have not considered scaling to large models of the size of cities or buildings, nor to many hundreds or thousands of users sharing and modifying the space.

Also, for many applications, the environment will only be truly useful if it supports physically realistic behavior. Research systems have not generally addressed this area; production systems have been confined to small, localized models and focus on the interaction of a single user with a small but complex model. Where multiple users are concerned, they are generally observers rather than actors.

2. Citywalk: a second generation architecture

We have now reached the point where networks and workstation hardware can support systems that combine these techniques to provide a richer, more useful virtual world experience. To support a functional and robust fusion of these first generation approaches, we need a system that combines

Figure 1. A second generation Citywalk model.

aspects of a distributed persistent database, which can provide model storage and support intelligent model loading and unloading, with a high-performance network layer that provides interactive speeds for time-critical aspects of agent-agent and agent-world interactions. This leads naturally to a two-tiered architecture comprising a tightly linked object database paired with a data distribution layer that can rapidly propagate critical information between clients over limited-bandwidth links. Practical applications involving realistic physics also demand that the system provide a framework for integrating physical simulations that can be “plugged in” to the system and act as agents that can modify the world concurrently with the users.

2.1. Database layer

The foundation of the Citywalk second generation system is a database layer that provides services that support a distributed, highly interactive, and scalable world model. First generation experience with large scale models suggests that the critical aspects of the database are the ability to incrementally load and unload sections of the model and culling structures. In order to support second generation functions, it also needs to be distributable, with multiple servers and clients in a single system. It must provide multi-client consistency functions (i.e. distributed locks and transactions) as well as efficient update mechanisms to allow clients to process model changes in real-time. This functionality is largely provided by modern object-oriented database systems (OODBMS). While off-the-shelf databases such as POET, ObjectStore, and Objectivity are good candidates for such a layer, practical considerations lead us to believe that a custom database layer is still a better choice right now. There are a number of extensions that can be made to the basic database protocols that enhance performance, such as augmenting the server with knowledge of geometric information [10]. These extensions would have been difficult to implement using a “black box” database.

2.2. Simulation support layer

The object database interface alone cannot provide sufficient performance to distribute simulation information to clients in real-time. Research systems that address this problem typically combine region culling with IP multi-cast [26] or intelligent servers [10]. These systems show that interactive performance on a limited bandwidth network requires a layer that is dedicated to distributing real-time data to clients efficiently based on individual clients’ regions of interest in the model.

It typically takes many years, a large budget, and a great deal of specialized expertise to author a simulation engine that provides high-quality, accurate output. We would like to take advantage of the huge body of robust, verified simulation codes. Unfortunately, these simulators were not designed to work as interactive agents within a virtual world. We feel that the best answer to this problem is to provide a framework into which simulations can be tightly integrated into a virtual world model with minimal effort and rewriting of code. This approach combines the work already done by physicists and engineers in designing the simulators with the latest approaches in computer graphics to provide a rich and productive user experience. Designing a general-purpose framework that allows these disparate systems to work well together is the challenge we have addressed with Citywalk.

Our interactive simulation layer provides a plug-in interface that allows a legacy simulator to be incorporated into the virtual world. The interface allows the simulator to interact with the virtual world database and manages clients that connect to the simulator. As the simulator generates data, the framework automatically sorts, prioritizes, and distributes the data according to the real-time interests of the client visualizers. This distribution system takes bandwidth availability into account and provides the highest quality real-time rendering that the client can support given its link to the rest of the system. It also provides efficient use of inter-server bandwidth by grouping interest regions for sections of the network when it replicates data between nodes in the server cluster. The interface is designed to make it as easy as possible to integrate new third-party simulators, and allow them to take advantage of the interactive, persistent world model as well as the real-time position feedback from client visualizers.

3. Plug-in simulation agents

First generation walkthrough systems are extremely limited in the types of interaction they allow. The user can typically only walk through and examine the static contents of a precomputed environment. In order to support more interesting and compelling dynamic virtual environments, our second generation system provides management for launching, controlling, and receiving updates from autonomous simulation agents. A simulation agent is a generic process that can act upon the database and potentially change its state.

The system's simulation support allows the incorporation of a variety of plug-in modules that can run on the host or on an arbitrary client computer. The agent can live at any point in the networked system; communication is handled via the intelligent data distribution layer. The integrating programmer needs to create two key modules: a user interface module that gives the user the ability to set up and trigger simulations via whatever interaction method is appropriate (for example, "lighting a fire" for the fire simulator), and a rendering module that converts the transported simulation data into a visual representation in the 3D world view.

We have demonstrated our framework in a working system jointly built and operated between UCB and MIT. It runs over the Internet and operates with clients on many different platforms, ranging from powerful SGI workstations to low-cost PC's. Users from both research groups can explore this virtual environment simultaneously and interact with one another as well as with the objects (furniture) in the model. The database for this system contains the detailed model of Soda Hall (the home of the CS Division at Berkeley) and several buildings of the MIT campus (Fig. 1). We have integrated several simulation agents into this framework: a fire and smoke simulator, a collision and reaction-force calculator, a multi-user agent, an on-demand impostor generator, and a differential radiosity update module. All agents share the same abstract interfaces.

3.1. CFAST (The Consolidated Model of Fire and Smoke Transport)

CFAST [17, 6] is the world's premier "zone model" fire chemistry and physics simulator. It assumes an environment composed of rectilinear volumes interconnected by vents. Within each volume, physical quantities such as gas species concentrations, combustion byproducts, and atmospheric pressure and temperature are tracked. Furniture and sprinklers affect the flame spread in realistic ways based on their materials and construction. A system of differential equations monitors the flow and exchange of these quantities through vents into adjoining volumes.

CFAST is a *virtual-time* simulation agent; it generates data in both space and time, but the time axis of the simulation has nothing to do with "real" time. The user can play, rewind, and explore the simulation data much like a person can watch a VCR tape. All view modules for virtual-time simulations inherit a standard time controller that replicates a standard VCR button panel to control the "playback" of the simulation. The panel also displays the current progress of the simulator, as well as the virtual-time being displayed in the visualizer window.

Simulation clients use the data distribution framework to tell the simulator which areas of the model are of interest to the users. The framework prioritizes the data communicated from simulator to client based on this interest set, allowing the system to perform well even on very low bandwidth links and to guide the simulation towards particular regions of the model or types of simulated output. A complex simulation of fire in a large building can be visualized in real-time over links as slow as a 28kbit modem line; this would not be achievable using a naïve data transfer method.

Output can be visualized quantitatively, through the simulator output panel, or qualitatively, via the 3D virtual world view. The data panel can also display numeric or graphical information about the simulation in progress. The CFAST module adds a readout of the temperature, pressure, and gas concentrations where the user is standing, allowing the user to probe regions of the model simply by moving there. An integrated 2D graphing package can provide graphs of conditions over time. Qualitative views of smoke and fire in the virtual world viewport are rendered in each frame by the CFAST user interface plug-in. A plug-in may implement multiple methods of visualizing the data; for example, the CFAST module offers a *realistic* view mode with swirling smoke and flickering flames, a *symbolic* mode with transparent panes for smoke and geometric objects representing the fire (Fig. 2a,b), and a *thermal imaging* mode where it draws heat patterns on the walls to convey volume temperatures (Fig. 2c).

In practice, this complete integrated simulation has proven to be an intuitive way to view CFAST output. Users can import a defined fire from a known input case, and place it with a mouse click. They can then walk through the building while the fire burns, using the time controls to speed up or slow down the progress of the fire until something interesting happens. When an interesting event is found, time can be stopped or reversed, and the scientific visualization modes and qualitative view panel can be used to help understand the implications.

Figure 2. CFAST: (a) Fire in a room, (b) Smoke in a hallway, (c) Thermal imaging mode.

3.2. IMPULSE (Impulse-based dynamics simulation)

IMPULSE [16] is an object-level simulation of physical dynamics in the presence of forces in the environment. It is a virtual-time agent like CFAST (i.e. it generates a velocity and position time line for a set of objects); it generates spatial paths over time for a set of objects that are interacting with each other within each volume.

Starting an IMPULSE simulation requires setting up the objects in the starting configuration with the interactive Citywalk editor [5], then pressing the start button. This brings up the standard VCR time control, which controls visualization of the objects tumbling and interacting dynamically. Again, the user can play, rewind, or pause the dynamic simulation. IMPULSE provides no additional quantitative information, so the VCR is the only control available. The user can also use a special “strike object” command to impart additional momentum to an object. This command is communicated to the simulator via the framework’s communication channel. IMPULSE adds a momentum vector to the object’s current state in the simulation. This allows the user to interact directly with the physical simulation.

The data set for IMPULSE output consists of paths and velocities of the objects in each space. Thus, the visualization module for the IMPULSE simulation agent simply overrides the regular object drawing routine to insert an additional transformation specific to the object’s position at the given virtual-time. To the user, the objects seem to tumble and fall in a physically realistic fashion while the IMPULSE simulation is running (Fig. 3).

3.3. Real-time multi-user walkthrough

The multi-user agent [10] coordinates the presentation of the “avatars” of walkthrough clients. Unlike CFAST or IMPULSE, which are virtual-time agents, the multi-user agent is considered to be a *real-time* agent. This means that the information it is maintaining (the current position and orientation of each client in the world) reflects the state of something that is happening *right now* in the real world. As such, it needs to propagate that data quickly to other clients in order for it to be of any use. If data gets delayed for more than a few seconds, it quickly becomes useless; it is only useful to transmit the most current state of a client to the other clients.

In the case of the multi-user service, the “simulator” is very simple; it collects the current user states from each client. The simulation state for each room contains the set of clients that are sitting in that room, plus their current visual state (i.e. their location, what their avatar looks like, and what it is doing). This information is then propagated intelligently back to the other clients as “simulation” data. At each client, the rendering plug-in for the multi-user simulator now has access to up-to-date positions and avatar conditions for all the other clients that are currently in the visible

Figure 3. (a),(b): IMPULSE dynamics simulation. (c): Multi-user simulation.

set of rooms for this particular client. The rendering plug-in simply draws graphical “avatars” at the positions of all the other clients that are “seen” by the given client.

The multi-user agent recalls the built-in, visibility-prioritized transmission of real-time simulation data that characterizes the RING system presented by Funkhouser [10]. It is important to note, however, that this functionality is effectively a special case of the Citywalk real-time visualization framework, even though it was encoded to propagate environmental data rather than specifically to perform multi-user visualization.

3.4. Radiosity on demand

To provide for realistic lighting conditions for walkthrough building models, radiosity solvers have been developed within a few first generation walkthrough systems [24, 11]. For large models, the computation time required for a complete global radiosity solution can be very large (on the order of days), so they are generally precomputed. Newer approaches [25] decrease the amount of time necessary to generate a solution, but the computational expense still makes it infeasible to employ such “global” solvers in an interactive, dynamic environment for interesting and highly complex models.

We assume that most changes made to a virtual environment are small and concern only a few objects or rooms. Thus, a previously calculated radiosity solution is a good starting point for calculating the new, adjusted radiosity solution [13, 9]. While any change may cause the illumination value to change on many polygons, we are most interested in reducing the shading error visible to a particular observer at a particular time. Past work in importance regions has demonstrated how to provide solutions with a bias for increased accuracy near a preferred viewpoint [4]. We have adapted a radiosity solver to be a simulator plug-in that operates incrementally, refining partial shading solutions on a surface-by-surface basis by focusing computational resources on areas of greatest visual importance.

When the radiosity process is started on a model for which no previous solution has been computed, the location of the observer can be used to guide the global solution dynamically to provide better immediate results to the observers. The simulator interface provides each simulator with a visible and potentially visible set for each attached client. The union of these sets is the global visible set that represents the areas of interest for all users within a model. Each cell is given a priority for the radiosity solver based on the proximity of the closest viewer and the number of viewers of the cell. While we would like to concentrate as much computation as possible on the objects in the visible set, it is important to assign some priority to objects in the lookahead set so that the observer will find other parts of the model reasonably well lit when they move around.

An existing radiosity solution, or a solution in progress, may need to be updated when an object moves or when its material properties are changed. Using the database watch mechanism, the radiosity simulator can monitor all objects for which a solution has been computed. If any of these objects should change, the simulator will analyze the object to determine an appropriate dynamic response. When an object’s material properties have changed, we “shoot” a correction for the changed material properties back into the environment [21]. For each object receiving correction, we may further shoot a new correction to the objects in that object’s visible set, and so on until the correction factor becomes negligible. When an object’s geometry has changed, we can also shoot corrections into the environment, this time based on the change in form factors between the objects and all objects in its old and new visible sets.

While we did not expect to get real-time performance due to the fairly loose coupling of the radiosity simulator to each client’s rendering system, we did see performance that was acceptable

for interactive purposes in many instances. With the described techniques in their current state of implementation, we see radiosity updates every few seconds, with the final solution taking on the order of a few minutes. A reasonable looking initial solution takes about one minute.

4. Tapestries: On-line impostor generation

In order to achieve fast, interactive frame rates, first generation walkthrough systems utilize a combination of model-based visibility culling, prediction of user behavior, and suitably chosen impostors or lower levels of detail (LOD) for some parts of the model. In scenes of high depth complexity, many objects, or portions of objects, that are not visible to the current view may be sent through the rendering pipeline. In order to achieve interactive frame rates and visual quality in such environments, it is imperative to render only those portions of the scene that are actually visible. One approach to address this problem is to generate view dependent image-based LOD representations for large collections of objects [15], as well as for individual objects. Another strategy is to utilize 2.5D textured depth meshes either as the primary rendering primitive [7] or to provide the background behind other parts that are displayed with full 3D geometry [20]. Of course, any precomputation of such representations will become invalid if the underlying environment changes. Some image-based techniques generate image impostors in real time by caching results from the frame buffer [18, 19]. Our tapestry approach [22] extends existing impostor generation techniques to incorporate samples from multiple views and to support automated on-line generation and update in the Citywalk environment.

4.1. Tapestry construction

A *tapestry* is a triangle mesh constructed from an on-line sampling of the environment. The sampling is done from a collection of adjacent views, resulting in a representation of the surfaces in the environment visible from those views. With a relatively dense sampling, a subset of the samples corresponding to important visual features is chosen as vertices. In addition, explicit edges are specified at apparent discontinuities in the sample image and incorporated into a constrained Delaunay triangulation. The set of sample color values is stored as a texture and mapped onto the triangle mesh. This part of the approach is similar to an approach used to generate impostors for urban environments [20]; we improve on the mesh quality by considering normal as well as depth discontinuities when constructing constraint edges, and by incorporating samples from multiple views.

To generate a tapestry for a given view, the environment is first rendered using OpenGL and the resulting image is stored as a texture. Each pixel is then processed and labeled as a depth or normal discontinuity. Groups of such “discontinuity pixels” are then approximated by an edge if the line segment formed by the end points of the pixel chain is a reasonable approximation to the

Figure 4. Tapestry construction: a) input, b) discontinuity map, c) mesh, d) tapestry.

set of pixels. These edges and vertices are then incorporated into a triangle mesh. The vertices store the world-space location of the sample, resulting in a 2.5D representation. Figure 4 illustrates an example of constructing a tapestry for a portion of a building environment.

The resulting mesh contains enough geometric information to produce appropriate parallax effects when viewed from nearby viewpoints. In general, there will be areas that were not visible in the initial sampling. The mesh triangles in these locations (called “skin” triangles) incorrectly interpolate between two disjoint surfaces. The further the observer deviates from the generating view, the more apparent the artifacts arising from these undersampled areas will become. In order to minimize these visual artifacts, we perform sampling from additional nearby views and incorporate this information into the mesh. For each additional view, the current tapestry mesh (minus skin triangles) is rendered from the new view. Regions for which the current representation is missing geometric information are identified using the depth-buffer. Mesh sub-patches are then constructed for each region: resulting in a collection of depth meshes relative to the secondary view.

4.2. Tapestry simulation agent

We have implemented a tapestry-based simulation agent that automatically generates tapestries in batch mode and updates them on-line if the environment changes. A tapestry is associated with each relevant portal in the cell-portal visibility structure and represents the portion of the environment visible through that portal when viewed from a particular region of space.

In batch mode, the agent traverses the portal list and generates a tapestry for each portal. We have found three default views (on a semi-circle defined by the portal width, observer height, and portal normal, at 0, -30 and 30 degrees) to be sufficient to capture the geometry visible through the portal. The tapestry is attached to the portal and added to the database. At run-time, the portal tapestry is rendered when the portal is visible, but not close enough to require full geometry (the default distance is five times the portal width). A similar dynamic technique is presented in [3] using textured rectangles as portal impostors. We have also used portal textures as a lower LOD, but found the quality to be unsatisfactory except for very limited viewing conditions. Our approach significantly improves the range of the use of such impostors by incorporating sampled geometry into the representation.

The agent also supports dynamic update of tapestries. Each object that is represented by a tapestry impostor stores a reference to the tapestry. Each cell also maintains a list of adjacent tapestries. The simulation agent maintains a watch on all tapestry objects. If an object moves, the appropriate tapestries are regenerated. If an object’s surface appearance changes, then only the texture maps associated with the geometry need to be regenerated. The tapestry server can run on a remote system to avoid contention for rendering resources. We have demonstrated tapestry update with object insertion and editing, in conjunction with an on-line radiosity update.

For a walkthrough sequence in the Soda Hall model (approx. 1.5 million polygons) we achieved framerate speedups of 5-7 times. Even though the model has relatively low depth complexity (on average < 1.5 after culling), the use of the tapestry still reduced the number of pixel writes by tens of thousands for a single frame. We expect even greater speedups with increasing model complexity. For such environments, a fully automated, dynamic, scalable, output-sensitive display representation like the portal tapestry is essential. The color plates in Figure 5 provide an example, in images and a narrative, of a typical update task for a user editing a database in which both the tapestry and radiosity agents are active.

(a) We approach the alcove to be edited. At this distance the alcove is represented by a tapestry.

(b) We enter the alcove and deposit a brightly emissive sculpture on the pedestal. This triggers watches in both agents via the change to the cell's contents list.

(c) We stay in the area to observe the radiosity update. After a minute, the radiosity agent begins to commit changes to the surfaces in the cell, causing watches to fire in the viewer. The viewer reloads the changed surfaces and we see the bright lighting.

(d) Moving back again, we see the tapestry that was committed by the tapestry agent while the radiosity agent was computing the first gather. Note that the bright lighting is not present in the tapestry.

(e) After 3 gathers the radiosity solution has converged, and the final watch causes the last updated tapestry to be displayed in the viewer.

(f) A diagram illustrating the flow of information between the database server, viewer client, and simulation agents.

Figure 5. An interaction between a user, the radiosity agent, and the tapestry agent.

5. Towards third generation systems

The system that we have described and implemented allows us to navigate and run simulations in a collection of buildings that fit into a single homogeneous database. One clear avenue for future evolution of walkthrough systems is an extension to integrated systems that involve models of different kinds and an expansion to models of wider scope and larger scale. Many groups have independently built virtual worlds with very sophisticated machinery for visibility culling, LOD selection, efficient collision detection, and other simulation tasks. This machinery is often tied very closely to the internal structure of the particular walkthrough system. For example, NPSNet [26] uses different basic structures from the downtown LA model [14], the UNC coal-firing plant [2], or the Berkeley Soda-Hall Walkthru model [12].

Conceptually, the simplest approach to combining such models into a virtual world would be to convert all data into a single walkthrough model format and to use one set of tools to navigate it. However, such an import task could be impractically large, and there might be primitives that translate poorly and structures that would be lost. It is thus preferable to use these models as they were designed, with their abstractions and machinery in place.

We propose to introduce another level of abstraction to the world model. Rather than merely considering scene graph objects, cells, and actors, we need to add the concept of a walkthrough space, which has its own visibility culling and rendering methods, and which may reside on a remote system. We will let the individual walkthrough systems handle the rendering of their model worlds, since they have the appropriate machinery, and we will create a new layer of integration via a general communication interface designed to handle rendering queries between these different heterogeneous systems.

The simplest form of a query will consist of a view frustum relative to the coordinate frame of the child space within the integrated heterogeneous model. A single view of a scene will result in a recursion into all spaces that are visible in the view. Each space will collect up visible geometry from its contained spaces, which will be returned in the query response to its parent space. The rendering program will gather up all of the geometry and impostors from all spaces that are visible on each frame and render them together.

The simplest interface offers occlusion culling only in the most rudimentary form: that derived from the view frustum. Almost every virtual reality system provides more advanced mechanisms for culling hidden objects. If we want to provide such mechanisms at the highest level of abstraction, our interface must be capable of transmitting occlusion information. In order to achieve this goal, we must devise a format to describe occlusion information in the form of a generic visibility structure, for example, as a portal tree plus occluders. Information in this format may then be transmitted for a particular view across multiple types of walkthrough models. For example, we may use a cell and portal visibility scheme within a building [1, 12], and a cull horizon for looking across a city [8]. A single view from within a building may include occlusion specifications from both of these mechanisms at once.

So far, we have only described an interface for *rendering* of heterogeneous models. Clearly, before too long people will demand all the same capabilities that are now available in second generation systems: simulation and interaction. This will necessitate transmittal of corresponding information (forces, temperatures, light flux, etc.) through the interface. This is an open grand challenge: how to deal with interactions that go across the seams of such a heterogeneous world. Opening up simulation across the boundaries of these models may require a very large bandwidth of communication, and interactive simulations may be difficult to achieve at interactive rates with the long feedback delay inherent in distributed systems.

6. Summary

We have presented the salient features of Citywalk that we believe to be representative of second generation virtual walkthrough environments. Our environment combines many of the techniques that were individually developed on several different first generation walkthrough systems. Its architectural framework is built on top of an object-oriented database management system and makes use of an intelligently buffered communication layer. These abstractions enable a fairly platform independent system and make it easier to distribute its functionality over many different computing sites. Different interaction and simulation engines can be added to the environment in a modular manner, as demonstrated by the five agents described in this paper.

We conclude with a vision of a third generation framework that would allow us to combine walkthrough systems with different organizations into a heterogeneous world model, in which the various model spaces communicate through a standardized interface that handles suitably abstracted and extended rendering requests.

References

- [1] John M. Airey. *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Sub-division and Potentially Visible Set Calculations*. PhD thesis, Dept. of CS, U. of North Carolina, July 1990. TR90-027.
- [2] D. Aliaga, J. Cohen, A. Wilson, H. Zhang, C. Erikson, K. Hoff, T. Hudson, W. Stuerzlinger, E. Baker, R. Bastos, M. Whitton, F. Brooks, and D. Manocha. MMR: An interactive massive model rendering system using geometric and image-based acceleration. In *Proceedings Symposium on Interactive 3D Graphics*, pages 199–206, April 1999.
- [3] D.G. Aliaga and A. Lastra. Architectural Walkthroughs Using Portal Textures. In *Proceedings IEEE Visualization*, 1997.
- [4] Philippe Bekaert and Yves D. Willems. Importance-driven progressive refinement radiosity. *Proceedings Eurographics Workshop on Rendering*, pages 316–325, June 1995.
- [5] Richard Bukowski and Carlo Séquin. Object associations: A simple and practical approach to virtual 3d manipulation. In *Proceedings of Symposium on Interactive 3D Graphics*, pages 131–138, April 1995.
- [6] Richard W. Bukowski and Carlo H. Séquin. Interactive simulation of fire in virtual building environments. In *Computer Graphics (Proceedings of SIGGRAPH 1996)*, August 1997.
- [7] L. Darsa, B. Costa, and A. Varshney. Navigating static environments using image-space simplification and morphing. In *ACM Symposium on Interactive 3D Graphics*, pages 25–34, 1997.
- [8] Laura Downs, Tomas Möller, and Carlo Séquin. Occlusion horizons for driving through urban scenes. In *Proceedings of Symposium on Interactive 3D Graphics*, pages 21–25, 2001.
- [9] David A. Forsyth, Chien Yang, and Kim Teo. Efficient radiosity in dynamic environments. *Proceedings Eurographics Workshop on Rendering*, pages 313–323, June 1994.
- [10] Thomas A. Funkhouser. Ring: A client-server system for multi-user virtual environments. *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, pages 95–92, April 1995.
- [11] Thomas A. Funkhouser. Coarse-grained parallelism for hierarchical radiosity using group iterative methods. In *Computer Graphics (Proceedings of SIGGRAPH 1996)*, pages 343–352, August 1996.
- [12] Thomas A. Funkhouser, Seth J. Teller, Carlo H. Séquin, and Delnaz Khorramabadi. UCB system for interactive visualization of large architectural models. *Presence: Special Issue on Teleoperators and Virtual Environments*, 5(1):13–44, Winter 1995.
- [13] David W. George, François X. Sillion, and Donald P. Greenberg. Radiosity redistribution for dynamic environments. *IEEE Computer Graphics & Applications*, 10(4):26–34, July 1990.
- [14] W. Jepson, R. Liggett, and S. Friedman. An environment for real-time urban simulation. In *Proceedings Symposium on Interactive 3D Graphics*, pages 165–166, 1995.
- [15] P. W. Maciel and P. Shirley. Visual Navigation of Large Environments Using Textured Clusters. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 95–102, 1995.

- [16] Brian Mirtich. *Impulse-Based Dynamic Simulation of Rigid Body Systems*. PhD thesis, University of California, Berkeley, 1996.
- [17] P. Reneke et. al. R.D. Peacock, G.P. Forney. CFAST, the consolidated model of fire and smoke transport. 1993.
- [18] G. Schaufler and W. Stürzlinger. A three-dimensional image cache for virtual reality. In *Computer Graphics Forum (Eurographics 96)*, pages 227–236. 1996.
- [19] J. Shade, D. Lischinski, D. Salesin, T. DeRose, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *Computer Graphics (Proceedings ACM SIGGRAPH)*, pages 75–82, August 1996.
- [20] F. Sillion, G. Drettakis, and B. Bodelet. Efficient impostor manipulation for real-time visualization of urban scenery. In *Computer Graphics Forum (Proceedings Eurographics)*, pages 207–218, 1997.
- [21] François X. Sillion and Claude Puech. Radiosity and global illumination. 1994.
- [22] Maryann Simmons. *Tapestry: An Efficient Display Representation for Interactive Rendering*. PhD thesis, Dept. of EECS, University of California at Berkeley, May 2001.
- [23] G. Singh. Bricknet: Sharing object behaviors on the net. In *Proceedings of the IEEE Virtual Reality Annual Symposium*, pages 19–25, 1995.
- [24] Seth Teller, Celeste Fowler, Thomas Funkhouser, and Pat Hanrahan. Partitioning and ordering large radiosity computations. In *Computer Graphics (Proceedings of SIGGRAPH 1994)*, pages 443–450, July 1994.
- [25] Andrew Willmott, Paul Heckbert, and Michael Garland. Face cluster radiosity. *Eurographics Rendering Workshop 1999*, 1999.
- [26] M. J. Zyda, D.R. Pratt, J.G. Monahan, and K.P. Wilson. NPSNET: Constructing a 3d virtual world. In *Proceedings Symposium on Interactive 3D Graphics*, 1992.