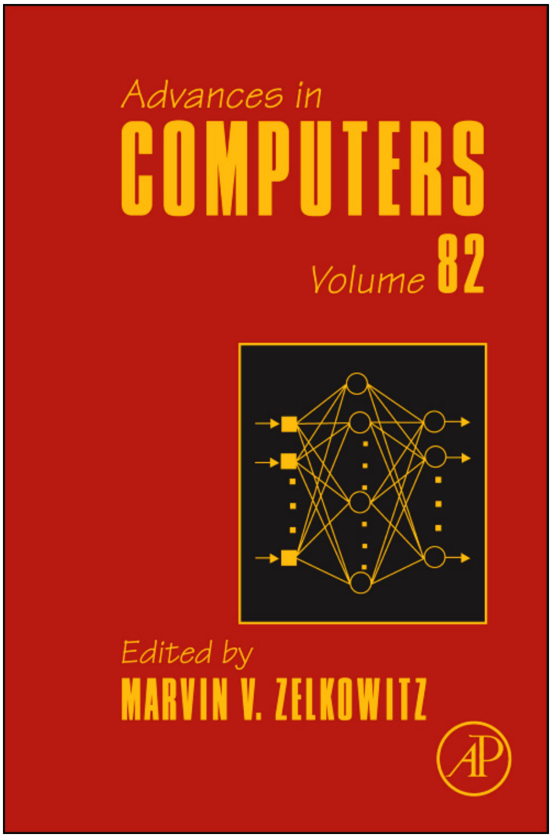


From: Jürgen P. Schulze, Han Suk Kim, Philip Weber, Andrew Prudhomme, Roger E. Bohn, Maurizio Seracini, Thomas A. Defanti, Advanced Applications of Virtual Reality. In Marvin V. Zelkowitz, Editor: *Advances in Computers,* Vol. 82, Burlington: Academic Press, 2011, pp. 217-260.
ISBN: 978-0-12-385512-1
© Copyright 2011 Elsevier Inc.
Academic Press

Advanced Applications of Virtual Reality

JÜRGEN P. SCHULZE

Calit2, UC San Diego, La Jolla, California, USA

HAN SUK KIM

Department of Computer Science and Engineering, UC San Diego, La Jolla, California, USA

PHILIP WEBER

Calit2, UC San Diego, La Jolla, California, USA

ANDREW PRUDHOMME

Calit2, UC San Diego, La Jolla, California, USA

ROGER E. BOHN

IR/PS, UC San Diego, La Jolla, California, USA

MAURIZIO SERACINI

CISA3, UC San Diego, La Jolla, California, USA

THOMAS A. DEFANTI

Calit2, UC San Diego, La Jolla, California, USA

ADVANCES IN COMPUTERS, VOL. 82
ISSN: 0065-2458/DOI: 10.1016/B978-0-12-385512-1.00006-2

Abstract

In the first 5 years of virtual reality application research at the California Institute for Telecommunications and Information Technology (Calit2), we created numerous software applications for virtual environments. Calit2 has one of the most advanced virtual reality laboratories with the five-walled StarCAVE and the world's first passive stereo, LCD panel-based immersive virtual reality system, the NexCAVE. The combination of cutting edge hardware, direct access to world class researchers on the campus of UCSD, and Calit2's mission to bring the first two together to make new advances at the intersection of these disciplines enabled us to research the future of scientific virtual reality applications. This chapter reports on some of the most notable applications we developed.

1. Introduction

The term virtual reality (VR) first made it to the mainstream about 25 years ago, when Scott Fisher's group at the NASA Ames Research Center presented the virtual interface environment workstation (VIEW) system, a head-mounted display with headphones and gloves [1]. However, at the time, technology was not advanced enough for VR to be economical for real-world applications, and even the video game industry gave up on it after several attempts in the 1990s at bringing the technology into video game arcades, for instance, the Trocadero in London.

Today's VR systems consist, at a minimum, of the following components:

- Graphics rendering units: The computer hardware to compute the virtual scene and render it to a frame buffer, ready to be sent to a display device. This is typically a high-end graphics PC.
- 3D stereo display units: Serve as the interface from the computer to the user. These used to be projectors and screens, but with the advent of 3D flat panel LCD or plasma displays these are more and more common.
- Tracking system: Serves as the interface from the user to the computer.

VR made a comeback in the past decade, when consumer graphics computers became powerful enough to compete with high-end, specialized graphics mainframes. This development brought the cost for graphics rendering down by more than one order of magnitude, while increasing image fidelity at the same time. Today, graphics mainframes are no longer being manufactured; they have been replaced by professional versions of computer gaming hardware, which are based on their consumer counterparts and thus only marginally more expensive than consumer solutions.

Similar developments have been happening with display and tracking technology. Tracking systems used to be either wireless and very expensive, or tethered and still expensive. Today, wireless optical tracking systems are available at a cost similar to a high-end PC. And with the advent of consumer 3D TVs, VR displays have finally made it to the consumer market. It remains to be seen if 3D is going to survive in the consumer market, and if the trend toward making VR feasible in the home is going to continue.

1.1 VR Hardware at Calit2

At Calit2, we built a number of novel 3D VR display systems over the past 5 years. The most notable ones are the StarCAVE, the NexCAVE, and the AESOP wall. The StarCAVE [2], as seen in Fig. 1, is a room-sized immersive VR system with about 10 ft diameter. The user wears polarized glasses and stands in the center of an array of 15 screens, each driven by two projectors for passive stereo. A cluster of 18 high-end graphics PCs renders 3D images on 34 HD (high definition) projectors (1920×1080 pixels each) with Nvidia Quadro 5600 graphics cards. We use passive stereo, so the user has to wear glasses with polarizing filters. In order to give

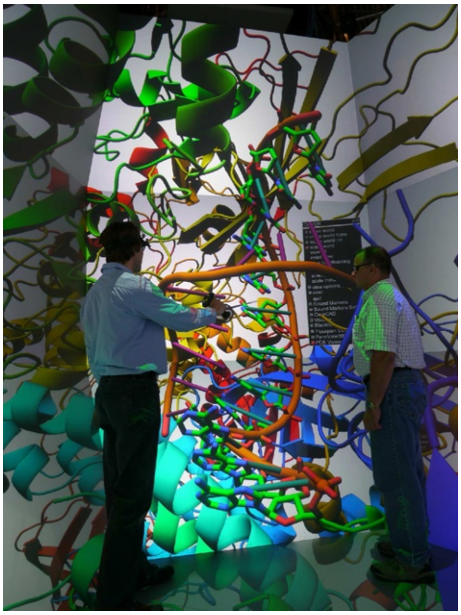


FIG. 1. The StarCAVE with two users looking at a protein structure.

a 360 viewing angle, the screens are rear projected. For head and hand tracking, we use a wireless, optical tracking system with four infrared cameras, mounted at the top of the StarCAVE. For surround sound output, we have a 5.1 channel surround sound system.

The NexCAVE [3], shown in Fig. 2, is the first tiled, immersive VR system based on flat panel displays. The LCD displays use micropolarization to create a stereo image viewable with polarizing glasses. The system consists of ten 46 in. HD displays, six high-end graphics PCs, and a two-camera optical tracking system. A Yamaha 5.1 channel digital sound bar can deliver spatialized sound. The displays are mounted with overlapping bezels, to minimize the space the bezels cover.

The AESOP wall, shown in Fig. 3, is a monoscopic tiled display wall, consisting of a 4×4 array of LCD displays, driven by a cluster of five high-end graphics PCs. The screens have ultrathin bezels, with only about 7 mm bezel space between two displays. This allows the system to run applications which do not need to worry about significant parts of the screen not permitting to display pixels, as it is the case with previous tiled display walls. The AESOP wall also features a two-camera optical tracking system and a 5.1 channel digital sound bar.

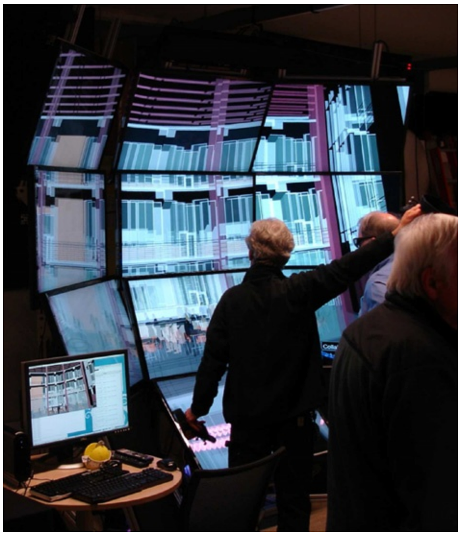


FIG. 2. NexCAVE: 10 overlapping passive stereo LCD displays.

1.2 VR Software Applications

This chapter is not going to focus on VR hardware, but instead at our recent developments in VR software. At the Calit2 center of UCSD, we have been researching and developing VR software since 2005. The VR center at Calit2 is in a unique position, as it was created, like many other laboratories at Calit2, to be a shared resource for researchers on the entire UCSD campus to allow them to utilize VR approaches without having to make the investment in expensive VR hardware themselves.

The software framework we use in all our VR systems is called COVISE [4]; its VR renderer is called OpenCOVER [5]. VR applications are written in C++ as COVISE plugins, multiple of which can run at the same time. COVISE abstracts the fact that the application is running in parallel over many machines and multiple OpenGL contexts, and automatically handles things such as stereo perspective calculations and OpenGL context management so that the application developer does not need to worry about them.

The Calit2 VR group has been working with researchers from a variety of disciplines at UCSD, but it has also conducted its own independent research into VR software applications, especially real-time immersive rendering and 3D user interfaces. In this line of work, we have been focusing on getting away from the

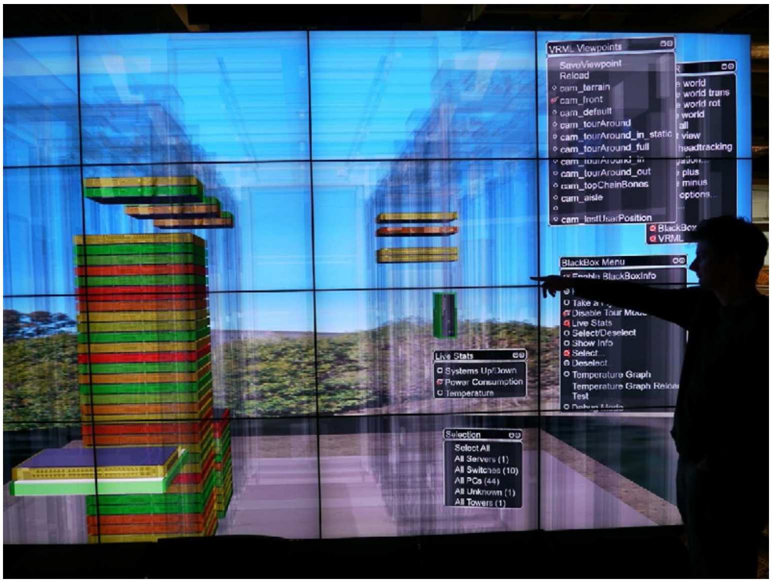


FIG. 3. AESOP Wall: 16 large, monoscopic narrow-bezel LCD displays with tracking and sound.

traditional mouse and keyboard interface, toward intuitive, immersive 3D interfaces. To us, good VR software applications use surround visualization and immerse the user in the data; 3D objects can be found at all possible distances from the user, from an arm's length to infinity; and user interaction does not just mean navigation of a static scene, but it means to interact directly with individual objects in the 3D scene with as much direct, object-oriented interaction as possible, without using menus or similar abstract constructs.

This chapter reports on some of the most innovative VR software applications we have created at Calit2. We are going to present one application from each of the following five topic areas:

- VR technology: Orthogonal developments of algorithms which do not stand alone, but are used in conjunction with other applications.
- Scientific visualization: The visualization of data sets which have inherent 3D structure, so that the mapping of data to the three spatial dimensions and sometimes also time is given.
- Real-time data visualization: The real-time visualization of data acquired by sensors and sent over networks.

- Information visualization: The visualization of data sets which do not have inherent 3D structure, where the mapping of data to the three spatial dimensions and time has to be created artificially and sometimes dynamically by the user.
- Cultural heritage: Applications aiming at preserving or investigating objects and places of a country's cultural heritage.

2. Virtual Reality Technology

In this section, we report on a project in which we created software infrastructure which can be used by other VR applications. The project enables high-resolution video display in multiscreen VR environments. The video can coexist with other 3D data, so that it can be mapped on other geometry such as virtual screens or buildings.

2.1 High-Resolution Video Playback in Immersive Virtual Environments

Today, most new feature films, TV shows and documentaries, and a rapidly growing number of home and professional videos are shot in HD. The most popular HD resolutions are 720 pixel (1280×720 pixels) and 1080 pixel (1920×1080 pixels). In addition to HD, there is the new digital cinema standard, called "4K," which describes resolutions from exactly four times HD (3840×2160 pixels) up to 4096×2400 pixels. While no cameras exist for even higher resolution video, it can be created by stitching together video from lower resolution cameras, or it can be rendered in the computer.

Real-time graphics applications, such as computer games and VR applications, often embed video in their virtual worlds. Examples could be a computer game which intends to realistically visualize Times Square in New York City with its video screens, or an architectural walk-through of a 3D model of a movie theater with a movie playing (see Fig. 4), or a surveillance system with many camera feeds. Integrating video into 3D applications can increase the level of realism, and it can bring important information into the virtual world.

Because of the large data rate of high-resolution video, easily exceeding hard disk throughput rates, it is not straightforward to display high-resolution video in real time. Playing back HD video is already CPU and GPU intensive for today's computers, even when it is displayed on a 2D monitor with software like Windows Media Player or Apple QuickTime. While these video players benefit from hard-wired, optimized circuits on the graphics card and special operating system routines,

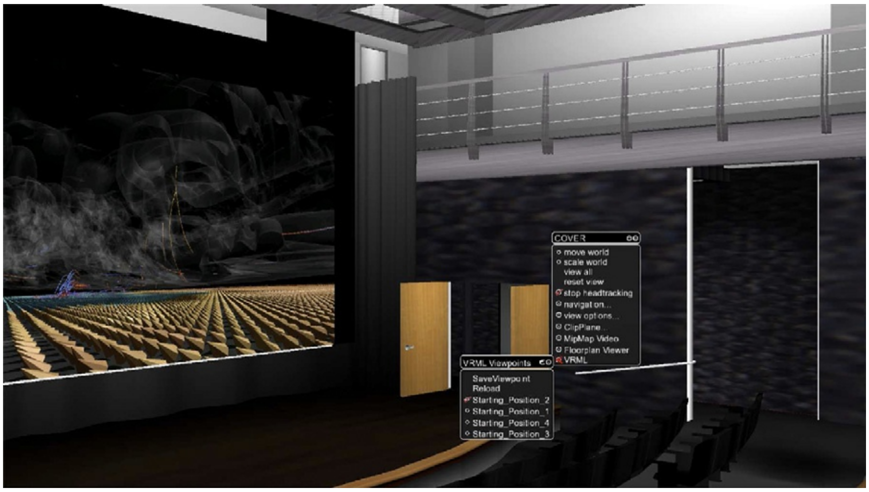


FIG. 4. The video playback plugin embedded into a virtual theater VR application. Users can navigate into the theater and watch high-resolution videos. The virtual theater application renders 201,688 polygons.

3D applications which display video in a virtual 3D world cannot benefit from these optimizations because the video is not aligned with the screen.

Our approach limits the video source to prerecorded material, because we apply a non-real-time preprocessing step to the data. Our algorithm can display HD and 4K video, and even higher resolutions as well in a 3D virtual environment, where the video's projection on the screen is not rectangular, but a general quadrangle whose shape depends on the viewing angle and the orientation of the virtual screen plane with respect to the physical display. Shape and location of this quadrangle change as the tracked user moves around. Our algorithm is based on mipmapping [6] and tiling (clip mapping [7]) of the video frames, on top of which we add several optimizations to maintain a constant frame rate. Mipmapping means that we render the video at as low a resolution as possible, matching or slightly exceeding the physical display resolution. This minimizes the data bandwidth during playback. Predictive prefetching of data further enhances the performance of our system. Our approach is entirely software-based and only assumes a more recent graphics card. Our demonstration algorithm is based on the VR framework COVISE and OpenSceneGraph [8]. Besides high resolutions, our algorithm supports the simultaneous rendering of multiple video streams, each of which can be independently positioned within the virtual world.

2.1.1 Related Work

Tiling and mipmapping have often been combined [9–14]. An important variable in mipmapping is how the appropriate mipmap level is selected. Two approaches are commonly used: one that matches the screen resolution as closely as possible across the entire screen and one that uses a point of interest to use higher resolution around the point of interest. LaMar et al. [9] constructed a spatial data structure, a quadtree, to store multiresolution data sets. The generation of a proper level-of-detail is determined by two factors: (1) the distance from view point p to a tile and (2) the area a tile covers in projected space. Given point p, a tile is selected if the distance from the center of the tile to p is greater than the length of the diagonal of the tile. As the projection transformation matrix transforms objects closer to p to appear larger and those further from p smaller, data points closest to the view point have the highest resolution.

Blockbuster [15] is a movie player for high-resolution videos, which runs on tiled display walls under the DMX (Distributed Xinerama) window manager. It plays movies in Lawrence Livermore National Labs' SM format, which supports tiled images, multiple levels of detail, and several types of intraframe image compression. The main difference to our approach is that Blockbuster assumes a flat display wall and renders the video parallel to the screens.

2.1.2 System Overview

DVDs are meant to be played at a constant frame rate: 60 interlaced frames per second in the NTSC standard. The frame rate for digital cinema is typically 24 frames per second. We refer to these frame rates as the *video frame rate*, with every frame being a *video frame*. We assume the video frame rate to be constant and only to vary between different video clips.

In interactive computer graphics applications, each frame, which we call an *image frame*, typically takes differently long to render, depending on how complex the scene is. This *image frame rate* is not variable.

Our video display algorithm needs to solve the problem of rendering a video at a variable image frame rate. Our strategy is that if rendering an image frame takes longer than the duration of a video frame, we skip video frames as needed to play the video at a constant pace, which is a requirement for synchronization with audio, and it keeps natural motions at their natural pace. If rendering an image is faster than the duration of a video frame, we display the same video frame again in the next image frame.

2.1.2.1 Bandwidth Considerations.

An uncompressed 4K video clip of 10 min at 24 frames per second uses more than 400 GB of disk space. The bandwidth required to load the entire video easily exceeds the performance limit of even today's high-end PCs. However, our VR display systems rarely need to render the entire video at full resolution, because the video extends across multiple screens and the physical resolution of an individual screen is often much lower than 8 megapixels. Our approach optimizes the loading bandwidth by loading only those mipmap tiles which are actually going to be rendered.

2.1.2.2 *Multiple Instances.*

One goal we had for our algorithm is the ability to display multiple video streams concurrently and also to display 3D geometry along with the videos. For instance, we want to be able to display a virtual surveillance center with a control room which displays a multitude of videos. This requires that our algorithm uses a reduced amount of memory for each video stream, depending on how many streams there are and how much bandwidth they require, so that it can coexist with the other video streams, as well as the rendering of the 3D geometry.

2.1.3 *Software Design*

Figure 5 shows the main components of our system. The *video playback renderer* cooperates with three other components: *frame manager*, *mipmapped tile manager*, and *LOD mesh generator*. *Frame manager* controls which image frame has to be rendered at a given time to synchronize with the video frame rate. *Mipmapped tile manager* manages a large number of tiles. It first loads metainformation for all tiles, and whenever the renderer requests a tile for metadata or texture data, it returns all the necessary data to the renderer. Due to the typically large size of these videos, it is

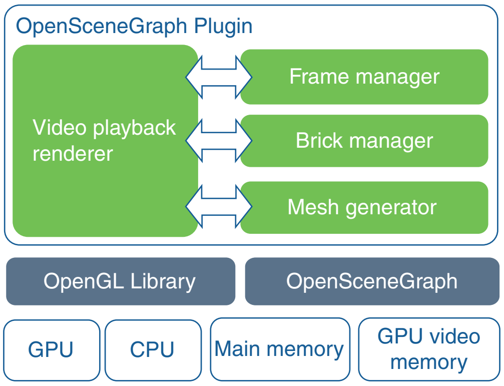


FIG. 5. System overview.

impossible to load all data into main memory at once. Thus, the *mipmapped tile manager* swaps requested tiles into main memory and then texture memory and removes the expired tiles. The decision about cache eviction is also made here. *Mesh generator* computes the best possible LOD for each region of the playback screen so that the smallest possible amount of data is copied into the texture, which utilizes memory resources and bandwidth more efficiently. We integrated the renderer and its accompanying three components into an OpenSceneGraph plugin for the COV-ISE software.

2.1.4 Rendering Algorithm

The rendering algorithm for a video stream consists of four steps: (1) mesh generation, (2) data loading, (3) prefetching, and (4) tile rendering. This routine is called once for every image frame, and it gets as its input the image frame to render from the frame manager. The mesh generation algorithm is presented in Section 2.1.6, and data loading and prefetching are discussed in Section 2.1.7. Once the algorithm completes mesh generation and data loading, it is ready for rendering the tiles. The final step is to iterate over the tiles which have to be rendered in the current image frame to draw them with their corresponding texture data.

2.1.5 Mipmap Generation and Tiling

In our approach, we preprocess the video frames in an off-line step. First, we extract the frames from the video clip. Then we downsample these frames to a set of consecutively smaller images by downsizing by 50% at every step, until the size is smaller or equal to an empirically determined tile size.

Figure 6 shows the layout of the tiles, which are stored in separate TIFF files. An image is divided into a 2D grid, and the origin of the grid is shown at the bottom left. Tiles at the rightmost column and at the topmost row are padded with zeros so that all tiles have a uniform size. Using a uniform size simplifies the rendering process.

2.1.6 Mesh Generation

The first step of rendering is to subdivide the playback screen into a set of tiles, which we call the mesh. The mesh comprises multiple tiles of different mipmap levels. The goal of subdividing the screen is to allocate the best possible mipmap level to each region with a limited number of tiles overall, because the number of tiles determines the amount of data to be loaded from disk or network.

We render areas closer to the viewer at higher resolution than those farther away. Rendering at lower resolution does not hurt the overall image quality because, after

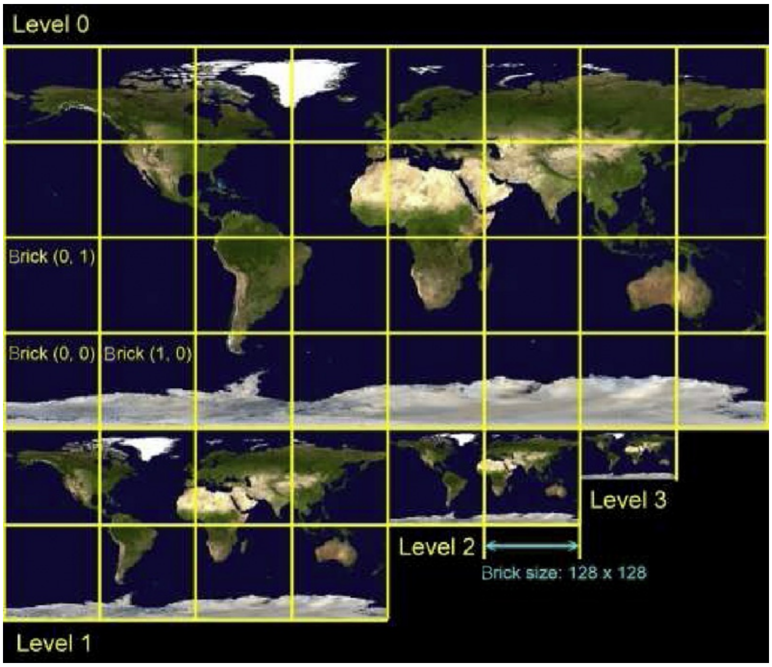


FIG. 6. Layout of tiles at multiple mipmap levels. The image is from NASA's Blue Marble data set [16].

perspective projection, the tiles farther from the viewer occupy fewer pixels on the screen, and the downsampled mipmap texture is still detailed enough to render this tile correctly without a noticeable change of the image quality. Our algorithm is based on quadtree traversal. Starting from the root node, which is the maximum mipmap level of the image, the algorithm checks whether or not the tile visited can be subdivided further. The area, *area(b)*, of tile *b* after transformations, that is, model view, perspective projection and viewport transformation, is used in the decision rule for the subdivision. Let *tileSize* denote the size of a tile. Then, if one tile of a certain mipmap level occupies about $tileSize \times tileSize$ pixels on viewport screen, the subdivision of this tile cannot further improve the image quality of the region. In virtual environments, the decision rule can be relaxed by adding a constant value α as follows:

$$area(b) > \alpha \times tileSize \times tileSize \tag{1}$$

where α can be any float value larger than 1. The algorithm subdivides a tile if Predicate 1 is true and stops if false. The constant α controls how detailed the image is rendered. If α is 1, the texel to pixel ratio of the rendered tiles is near 1. However, a large α makes the mesh algorithm stop the subdivision even if 1 texel of each tile maps to more than 1 pixel, which creates an image of lower resolution. α is introduced to control the system between high frame rate and the best image quality.

Another variable, *tileLimit*, controls the number of tiles to be rendered on a physical display screen. Tiles in the output list grow exponentially along the traversal of the quadtree. However, *tileLimit* guarantees that the rendering system does not have excessively many tiles on the rendering list. The ideal number for *tileLimit* is different from hardware configurations, and a realistic number often used is around 40×128^2 tiles. That is, 40 tiles on one display screen correspond to $40 \times 128 \times 128$ texels, which is about 640K texels.

With *tileLimit*, not all tiles can have the most desired mipmap level. Some tiles still can be subdivided into four smaller tiles to have higher resolution. Our algorithm, therefore, has to rank all tiles so that it can choose one tile among multiple possible choices of tiles given the bounded *tileLimit* value. In order to give priorities to each tile, a cost function is employed as follows:

$$cost(b) = \frac{area(b)}{distance(e, b)} \tag{2}$$

cost(b) denotes the cost for tile *b* and *distance(e, b)* measures the distance between the viewer's location and the center of tile *b*. The viewer's location is given by the tracking position in the VE. Intuitively, tiles occupying a large area on screen have higher priorities so that no large tiles of low resolution are left on the list. The denominator, *distance(e, b)*, gives higher priority to tiles closer to the viewer. Namely, this term provides a point of interest mechanism; as the viewer walks toward a specific part of the playback screen, the region around the viewer is set to higher resolution.

Figure 7 shows an example of a mesh generated by our rendering algorithm. The image plane is tilted in such a way that the bottom right corner of the plane is closer to the viewer. Tiles around the bottom right corner have a smaller size, which results in a higher resolution.

Due to the viewer's constant movement registered by the head tracker, *distance (e, b)* returns updated and normally different values than in the last image frame. This generally does not allow reusing the mesh generated for the previous frame, so we have to recompute the mesh for every image frame. Our view frustum culling test reduces the cost of the quadtree traversal by removing offscreen mesh cells. The quadtree optimizes this process by culling a culled parent's child nodes along with it.

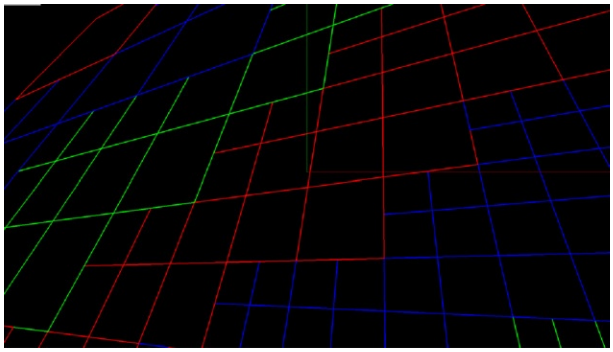


FIG. 7. Dynamically generated multiresolution tiled 2D volume.

Therefore, when the video is very large and spans multiple physical displays, a large portion of the quadtree is getting culled. As the video gets smaller, the traversal does not need to go down the quadtree as far anymore. This approach keeps the number of tiles rendered relatively constant.

2.1.7 Data Loading and Prefetching

The next step is to load the texture data from disk. Loading several megabytes of video data from disk as well as transferring them to texture memory for every frame slows down the rendering process as the bandwidth for data reads from hard disk is much lower than others in the system. We implemented three optimization methods to mitigate the disk bottleneck: prefetching, asynchronous disk I/O, and DXT compression.

2.1.7.1 Prefetching.

When videos are displayed in the StarCAVE, users either walk around the scene without paying particular attention to the videos, or they stop to watch a video clip. Even if they stop, there is always a slight change of the viewer position due to head tracking, but it is much smaller than when the user walks around. Therefore, our algorithm optimizes for a stationary viewer, for whom we found that mipmap meshes differ only by about four tiles.

Another issue is to predict the video frame from which the tiles are to be prefetched. After rendering video frame n, with n being the index of the video frame on disk, we calculate the next video frame to be displayed to be $(n+k)$. This means that we skip video frames $(n+1)$ to $(n+k-1)$. At every rendering step, k has to be estimated as correctly as possible, or the system will prefetch unnecessary tiles.

Again, we adopted a simple, computationally light scheme based on reinforcement learning [17]. We estimate the next frame by looking at the history of image frame durations. If the system has been skipping, for instance, every other video frame, we estimate that in the next image frame we are going to skip a video frame again. More formally, let A_n denote the current estimate of how many frames the system will skip and a_n be the current observation of the skip. Then, the next estimation of A_{n+1} is the weighted average between A_n and a_n.

$$A_{n+1} = \alpha a_n + (1 - \alpha)A_n$$

where α is a parameter representing how fast the algorithm adapts to new information a_n as opposed to the history A_n. We use the rounded values of A_n for the estimation of how many steps to skip. In order to further improve the accuracy, the $(n+k-1)$th and $(n+k+1)$th frames are also prefetched. The number of tiles prefetched is conservatively kept low, from one to four tiles, to prevent prefetching from generating too much load for the entire system and to utilize only the idle time of the I/O thread without delaying immediate requests from the rendering process even in the case of misprediction.

2.1.7.2 Asynchronous Disk I/O.

In order to accelerate data transfers between main memory and texture memory, a separate thread is dedicated to asynchronous disk read operations. Every disk read request is sent to the I/O thread via a message queue and the I/O thread reads data whenever it finds a message in the queue. There are two queues: a tile request queue and a prefetch request queue. The tile request queue contains the request from the main thread, which is for texture data of a tile that is needed to render the current frame. The prefetch request queue contains requests for texture data of a tile which will be needed in the near future. The messages from the tile request queue always have a priority over the messages from the prefetch request queue. In addition, the request for data loading is made as soon as the main thread finds a tile which will be needed for rendering. By posting disk read requests as early as possible, the parallelism between the rendering process and disk operations can be maximized. Another message used for communication between the main thread and the disk read thread forwards the current frame number.

2.1.7.3 DXT Compression.

DXT is a lossy compression standard which allows us to reduce the data rate by a factor of four, without a noticeable loss of image quality. This compression method works well for video because, due to the quick frame updates, it is hard for the eye to perceive the compression artifacts.

2.1.8 Synchronization

The time for rendering an image frame varies between frames, mostly depending on the number of tiles loaded for each frame. This causes two types of synchronization problems: synchronization (1) between frames and (2) between StarCAVE nodes. The first problem is that, without a synchronization scheme, the video frame rate changes depending on how many tiles are rendered for the frame, which varies depending on the viewer's location.

The second synchronization problem occurs because in a multinode virtual environment all nodes generally have different workloads and cause an imbalance in rendering times. For those display nodes that do not render much data, the rendering time is short, whereas other nodes might need more time for an image frame update than the video frame rate allows for, so that video frames have to be skipped. In our StarCAVE system, which consists of 17 rendering nodes, we update the images on all nodes at the same time, so that the update rate is equal to the image frame rate of the slowest node.

Our software provides a synchronized time which is the same on all nodes. Using this clock, we measure the time passed since the start of rendering the first frame, t_{elapsed}. Then, the desired video frame number, d, can be easily computed with the following formula for a 24-frames per second video clip:

$$d = d_{\text{base}} + \left[\frac{t_{\text{elapsed}}}{1/24}\right]$$

d_{base} denotes the frame number of the first frame. d_{base} will change when a video stream is paused and later continued. This approach solves the two problems because the above formula enforces frames to change neither too fast nor too slow, which solves the first problem, and because t_{elapsed} is measured from the globally synchronized clock, which is the solution for the second synchronization problem.

2.1.9 Results

We tested three different videos in the StarCAVE. We distributed the entire video to the local hard disks of the rendering nodes to avoid network bottlenecks. We used three different video clips: (1) A 1200 frame 4K clip showing a tornado simulation created by the National Center for Supercomputing Applications (NCSA); (2) The same tornado clip at a quarter of its original resolution; (3) A set of 24 microscopy images (14914×10341 pixels) from the National Center for Microscopy and Imaging Research (NCMIR). We preprocessed each of the video clips with our tiling and mipmapping tool and used a tile size of 512×512 pixels. Each display panel has full HD 1080 pixel resolution. Figure 8 shows the tornado clip in the StarCAVE.

FIG. 8. The VR video playback application in the StarCAVE with creator Han Suk Kim.

Table I shows the frame rates for various settings. Because the StarCAVE displays stereoscopic images, the frame rate here is defined as the time to render two images. Note that right eye images are rendered faster than left eye images, because they are rendered second and can thus benefit from the cached tiles loaded for the left eye. All measurements were averaged over a full playback cycle of each clip.

2.1.10 Conclusion

We showed and discussed the design and implementation of high-resolution video textures in virtual environments. In order to achieve a constant video frame rate, we created multiple levels of detail and dynamically subdivide the video into a set of tiles with different levels of detail. For efficient disk read operations, we assume that the plane will not change too much between image frames and prefetch tiles for the next frame. This helps overlap rendering with texture copying. In addition, synchronization was considered to sync the speed of rendering image frames and the video frame rate. Our experiments showed that our system provides constant frame rates and usable video playback performance.

TABLE I
FRAME RATES FROM THREE DIFFERENT VIDEO SOURCES

Video clip	2K Video 1920 × 1080			4K Video 3840 × 2160			Microscopy 12,941 × 10,341		
Configuration	Opt	No opt	LOD	Opt	No opt	LOD	Opt	No opt	LOD
2 × 2 panels	23.7	7.5	44.2	9.4	2.7	26.0	8.9	2.8	26.7
Single panels	20.4	5.0	59.6	18.0	4.9	45.1	21.8	6.1	58.4

Frame rate (frames per second) is the reciprocal of the time to render two images (stereoscopic image from left and right eye) and is averaged over a full playback cycle of each clip. We tested video playback on four (2 × 2) panels and on a single panel.

3. Scientific Visualization

Scientific visualization is the general term for almost all of the visualization projects we do at Calit2. This section reports on a very typical, albeit particularly sophisticated application we developed in close collaboration with scientists at UCSD. It is also our most successful application in a sense of how many scientists have used it for real work.

3.1 ImmersivePDB: A Protein Browser

To offer a new way of looking at molecular structure, we created a VR application to view data sets of the Protein Data Bank (PDB) [18] from the Research Collaboratory for Structural Bioinformatics (RCSB). The application can display the 3D macromolecular structures from the PDB in any of our virtual environments. Using the program ImmersivePDB, the viewer can move through and around a structure projected in VR. Structures can be compared to one another, they can be automatically aligned, and a variety of visualization modes can be selected from. The software has an interface that makes a connection to the RCSB PDB Web site to download and display files. Both single user and collaborative modes are supported.

To our knowledge, ImmersivePDB is the most fully featured protein browser for virtual environments, which is fully controllable from within the 3D world. In other approaches, for instance, PyMOL's [19] CAVE module, the virtual environment is only used to view and navigate the protein structures, but all other interaction like selecting the visualization mode, etc. is done at the head node in a 2D mouse/keyboard-controlled application.

3.1.1 Data Bank Access

The PDB currently contains over 60,000 protein structures, each of which has a unique, four-letter identifier. For hemoglobin, for instance, the identifier is 4HHB. The user can specify the protein ID in one of two ways: either during a VR session with the Loader window, or in the COVISE configuration file to add a preset menu item for the structure (Fig. 9).

Once the user selects a PDB ID, the system first checks the local cache for the PDB file. If the file is already in the cache, it will not be downloaded or converted to VRML again. If the file is not in the cache, it will be downloaded from the PDB server to the local visualization system with the following command line for hemoglobin:

```
wget www.pdb.org/pdb/files/4HHB.pdb
```

After the download, the file will automatically be converted to three visual representations: cartoon view, stick view, and surface view. All three representations will be stored on disk as VRML files. The conversion happens using a Python script which calls functions of the molecular visualization toolkit PyMOL [19]. The script is called with the following command:

```
pymol.exe -qcr batch.py -- 4HHB.pdb
```

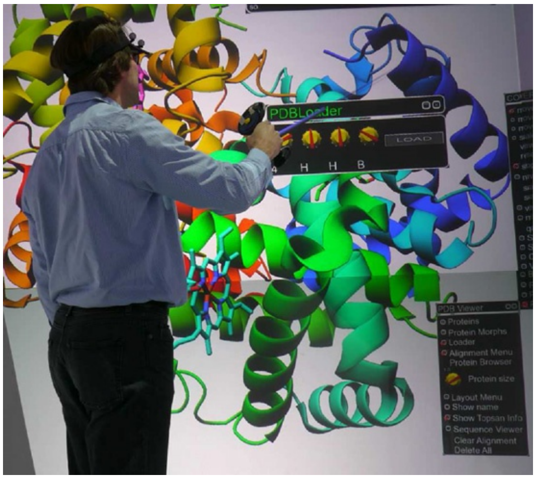


FIG. 9. A StarCAVE user dialing the PDB ID for hemoglobin into the ImmersivePDB loader window.

This command calls the following Python script to create three VRML files for the three supported visual representations.

```
from pymol import cmd
from pymol import preset
from pymol import util
from glob import glob
import sys
import os
path = ""
params = []
basepath = os. getcwd()
try:
index = sys. argv. index ("--")
params = sys. argv [index:]
if (len(params) == 2):
path = params [1]
cmd. cd (path)
else:
print "No Path specified"
except ValueError:
print "No Path specified"
for file in glob("*.pdb"):
listname = file. split(".")
name = listname[0];
cmd. load(file, name)
cmd. hide("all")
cmd. show("sticks")
cmd. save(name + "stix.wrl")
cmd. hide ("all")
preset. pretty(name)
cmd. save (name + "cart. wrl")
cmd. hide ("all")
cmd. show ("surface")
cmd. save (name +"surf. wrl")
cmd. delete ("all")
cmd. system ("rm -f"+ file)
print "Created " + name + " models"
cmd. cd(basepath)
```

3.1.2 Protein Morphs

In addition to proteins from the PDB, our system can visualize protein morphs from the Morph Server of the Database of Macromolecular Movements (Molmovdb) at Yale University. For instance, the calcium pump will be downloaded with the following command line:

```
wget -np www.molmovdb.org/tmp/396506-12995.tar.gz
```

Once the tar file has been downloaded, the time steps will be extracted as PDB files. The PDB files will be processed by the same PyMOL script as individual structures. The resulting VRML files will be loaded into an OpenSceneGraph Switch node, so that by switching through the time step models, the system can animate the morph.

3.1.3 Visualization Modes

Our PDB viewer supports three visualization modes for protein structures: cartoon, stick and surface. These can be selected from a VR menu which comes up when the user right clicks on a protein.

Our system is not limited to displaying one protein at a time. Instead, whenever the user loads a protein, it will be loaded in addition to what has already been loaded and displayed. Each protein can be moved around in 3D space independently, so the user can arrange many proteins around him to compare them or explore their differences.

When it comes to selecting a protein with the input device, we use a special selection mode for the proteins. Normally, in our VR system, users select objects with a virtual laser pointer or stick, by intersecting the pointer with an object. However, in the case of proteins, this is more difficult because the proteins are not solid objects, but they have rather large open areas in between the carbon chains. Hence, the user will often point at the protein without actually intersecting its geometry. We solved this problem by not intersecting with the geometry, but instead intersect with the protein's bounding box. We call the data structure, we use for this selection, the PickBox. Every protein is loaded into its own PickBox, so that the user can select individual proteins when multiple are loaded.

Once multiple proteins are loaded into the system, the user can choose to either lay them out manually by moving them to where the user wants them, or an automatic layout manager can be selected from a VR menu. We offer two different layouts: grid layout, which arranges the proteins in an array, or cylinder layout, where the proteins are arranged around the user on a virtual cylinder (e.g., Fig. 10). Radius and density of proteins on the cylinder are configurable.

3.1.4 Gradual Fading Between Visualization Modes

In some situations, the user wants to switch between two or all three protein visualization modes we support (cartoon, surface, stick). This can either be done manually, but checking selecting the respective boxes in the property sheet of the protein. Or, this can be done automatically as follows. If automatic mode is selected, the software will switch automatically between the selected visualization modes, based on the user's distance from the protein. Switches between modes do not occur suddenly, but the modes are faded in and out gradually, as the user moves. This, for instance, will allow the user to select surface and cartoon mode, and then gradually fade between them by moving closer to the protein, in which case it fades to cartoon mode, or farther away, and it will fade to surface mode. The parameters for this mode, for instance, the starting distance for the fade and the distance by which the visualization mode has fully switched to the next, are user configurable.

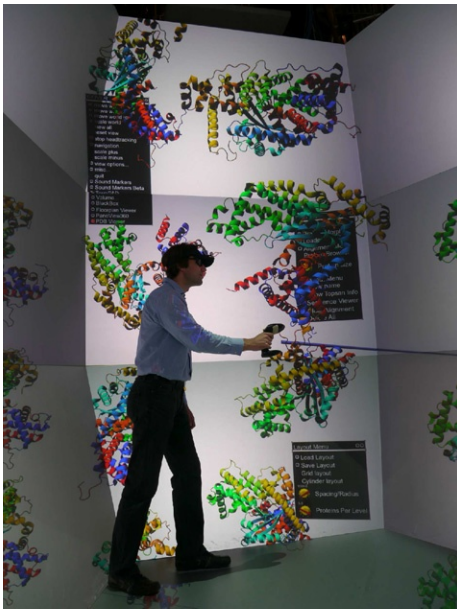


FIG. 10. Cylinder layout mode: all proteins are equally spaced on the surface of an invisible cylinder around the user.

3.1.5 Collaborative Mode

The software framework we use for the development of our application prototypes, COVISE, natively supports collaboration between multiple sites. Custom applications written for COVISE can utilize the collaboration API to support collaborative modes within the application. We used this API to make our ImmersivePDB application collaborative, which means that we can run instances of it at multiple sites (there is no theoretical limit to the number of supported sites) and collaboratively view protein structures.

COVISE will automatically offer three collaboration modes: loose, tight, and master/slave mode. In loose mode, the collaborators can view a data set independently, which means that they all look at the same data set, but their camera positions are independent from one another. In tight mode, all users share the same camera view. In master/slave mode, they also share the same camera view, but one user's camera motion dominates over the others. This is useful in training situations where the trainees should not be able to modify the instructor's view. In loose coupling mode, each user can see the other user positions indicated by a set of 3D glasses, a checkerboard pattern with an optional institutional logo where the user's feet are, and a 3D model of a hand with a pointer. This simple indication of an avatar for the collaborators shows what they look at and point to, which is useful when an audio connection is available to the collaborators as well.

Collaborative mode does not require particularly fast network connections. Even standard internet is normally fast enough for smooth collaboration. The bandwidth requirements are minimal, because only the user locations are sent over the network. Protein data sets need to be stored locally in each collaborator's cache, or downloaded from the PDB when a protein is selected.

Figure 11 shows a collaborative session between our booth at the International Supercomputer Conference (ISC) 2010 in Hamburg, Germany and our collaborators at King Abdullah University of Science and Technology (KAUST) in Saudi Arabia.

3.1.6 Amino Acid Sequence Browser

The amino acid browser is a dialog window in the virtual world, which lists all the amino acid chains of the selected protein structure in textual form. The user can interact with it in two ways: one is that the user can select a specific amino acid in the dialog window, and upon selection see where in the protein, the corresponding molecule is located, indicated by a cone-shaped marker. This also works the other way around: the user can move the marker to a place in the protein to find the place in the amino acid sequence it corresponds to.

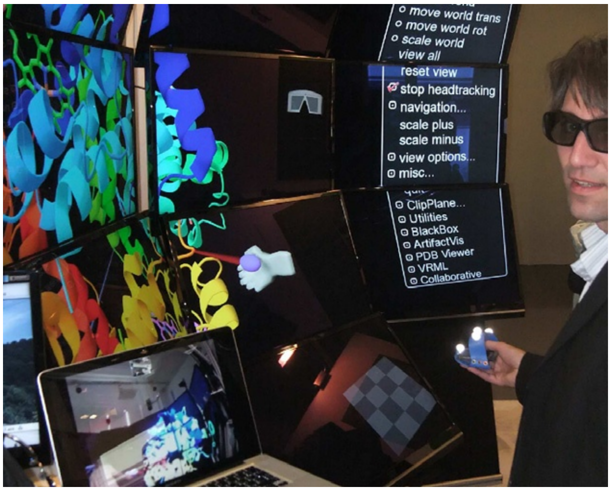


FIG. 11. Collaboration between two sites. Monitor in front shows video of collaborator's virtual environment at KAUST University. Stereo goggles, hand with pointer, and checkerboard pattern indicate collaborator's location in virtual world.

3.1.7 Alignment

The alignment of protein structures helps scientists understand the differences between them and is an important tool for them. The VR environment is very useful for the presentation of aligned proteins because it is easier to see the differences between two aligned proteins in 3D than in 2D, since the aligned offsets between the proteins generally occur in all three dimensions.

We integrated the multiple structural alignment program MAMMOTH-mult algorithm [20] into ImmersivePDB. In order to align two proteins, the user first selects Alignment Mode and loads the two proteins he wants to align. Their IDs will then be listed in a window, and copies of them are created and put on the alignment point interactor, a small cube the user can move to a convenient place to put the aligned proteins. The proteins copied to the alignment point are being colored in solid colors, with different colors for each protein, so that it is easy to distinguish them. These colors are user selectable. The alignment itself happens almost instantaneously and, in our experience, never takes longer than a few seconds.

3.1.8 TOPSAN Integration

The Open Protein Structure Annotation Network (TOPSAN) is an open annotation platform, created to provide a way for scientists to share their knowledge about functions and roles of proteins in their respective organisms. Researchers can add comments to the data bank through a Deki Wiki interface.

Whenever a structure is loaded by the user, ImmersivePDB connects to the TOPSAN server to see if information about the structure is available. If so, this information will be downloaded to the VR viewer and displayed in a floating window (see Fig. 12).

3.1.9 Conclusions

The ImmersivePDB application has been our oldest, but also our most "polished" and most used VR application. Researchers from the campus of UCSD, but also from other universities in Southern California, come to our laboratory to explore their protein structures interactively in 3D. They have told us time and again that they often see features of protein structures which they did not see at the desktop, and they develop a better understanding of the way these proteins function than by using monitor and mouse.

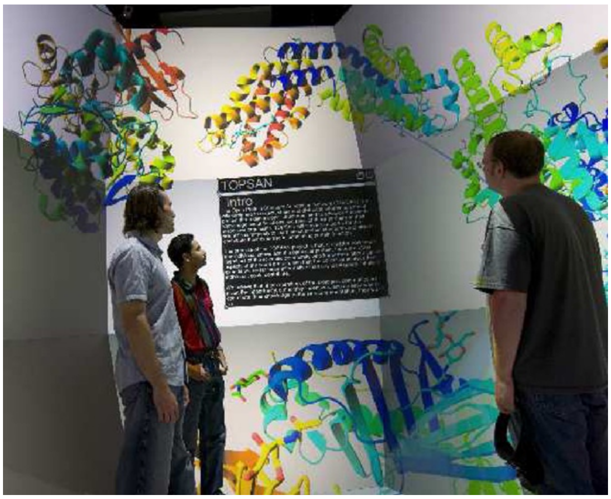


FIG. 12. TOPSAN creator Sri Krishna Subramanian with the ImmersivePDB team in the StarCAVE.

4. Real-Time Data Visualization

In this section, we are going to present an application which is based on real-time sensor input. Real-time sensor data is more and more available globally over the internet, so that monitoring and data processing tools based on these sensors do not have to be colocated with the sensors, but can be located where the users are.

4.1 The Virtual Data Center

In project GreenLight [21], which has been funded by the National Science Foundation (NSF) with $2 million over 3 years, researchers at Calit2 purchased and then populated a Sun Modular Datacenter (Sun MD [22]) with computing, storage, and networking hardware to measure the energy usage of computing systems under real-world conditions. The container can accommodate up to 280 servers, with an eco-friendly design that can reduce cooling costs by up to 40% when compared to traditional server rooms. The Sun MD's closed-loop water-cooling system uses built-in heat exchanges between equipment racks to channel air flow. This allows the unit to cool 25 kW per rack, roughly five times the cooling capacity of typical data centers.

The power consumption of each component in the Sun MD container is constantly being measured by networked Avocent power strips, and the data is sent to a central server. The measurements include data from temperature sensors in 40 locations around the container, to study the energy flow through the system. The goal of project GreenLight is to learn how computer hardware and software can be made more energy efficient. Early results for computer graphics hardware and software are already available [23]. In the following sections, we are going to report on the technical details of the implementation of our virtual reality monitoring application in greater detail.

4.1.1 3D Model

To help researchers understand power consumption and temperature distribution in the Sun MD container, we developed a 3D model of the Sun MD to visualize these measurements spatially. The 3D model, which is depicted in Fig. 13, is based on a CAD model of the data center from Sun Microsystems. To this, we added 3D models of the installed computer systems, such as servers, network switches, and storage systems. The model of the data center is a complete replica of the container, which allows the user to open the doors, enter the container, and pull out the computer racks, all by directly interacting with the 3D objects in VR. We even included a few

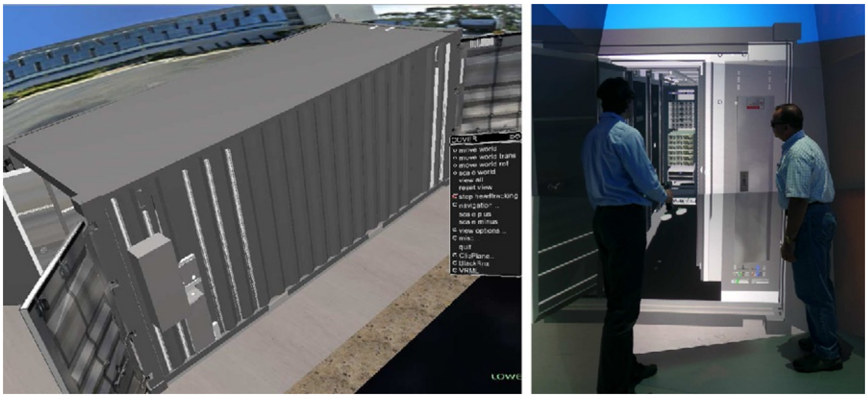


FIG. 13. Our 3D model of the Sun Mobile Data Center. Left: Bird's eye view. Right: View from the front in the StarCAVE.

dynamic elements like spinning fans and conduits which move with the racks when they get pulled out, to make the model as realistic as possible. We embedded the 3D model of the container in an aerial image of the campus of UCSD, where we placed it in the location it was physically installed. Around the container, we display a cylindrical view of the surrounding area, so that when the user is located at the container, the view around resembles what it is in reality. This is similar to the panoramic view "bubbles" Google Earth's Streetview mode uses.

We created the 3D model of the container and its content with Autodesk 3ds Max, which we also used for many of the interactive components. 3ds Max allows to connect VRML TouchSensors to geometry, which provide a way to trigger JavaScript code when the user clicks on them. JavaScript controls all interactions which cause geometry displacement in the model, such as doors which open or racks which get pulled out. Other interactions, such as the selection of individual components, were implemented in C++, referencing geometry in the container. In order to reference this geometry correctly, we use a common scheme of unique names for the geometry of the scene between the 3ds Max model and the VR application.

The VR application was created as a plugin for the VR software framework COVISE [4], which we use for the development of most of our VR applications. COVISE uses the OpenSceneGraph library [8] for its graphics subsystem. We exported the 3ds Max model with COVISE's own VRML exporter to a .wrl file. Our COVISE plugin loads that VRML file and adds its geometry to the scene graph. In addition to the static model of the container, this file also contains the touch sensor nodes with their associated JavaScript functions to control animation.

Our VR application connects via Web services to the central data server [24], which collects and stores the real-time information from the different types of sensors in the container and retrieves the status of the container in real time. Once received, this data is stored in XML files on our visualization server. We then parse these files to extract the sensor data to display in the VR model. Figure 14 shows two examples of how the instrumented components in the container are displayed using different colors depending on the type of measurement selected. For instance, when looking at temperature data, the components in the container may be depicted in red to indicate a high temperature, and green when they are running cool. This gives the user a quick overview of the state of the devices in the container. We implemented this functionality with an OpenSceneGraph node visitor, which traverses only the part of the scene graph with the computer components and changes their material properties depending on the measured data values. The connection between the data values and the geometry is established by the unique naming scheme mentioned above.

Our 3D application allows a user to visit the data center in VR, without having to physically go to the container. This saves on transportation cost and time but also has further reaching consequences—by allowing technicians and researchers to view the status of the container without having to go there, open it and thus allowing the cool air to escape from it, the measurements will not be interrupted. Many routine maintenance jobs, such as checks for available space, the status of a component, or repairs, can be avoided or at least optimized this way.

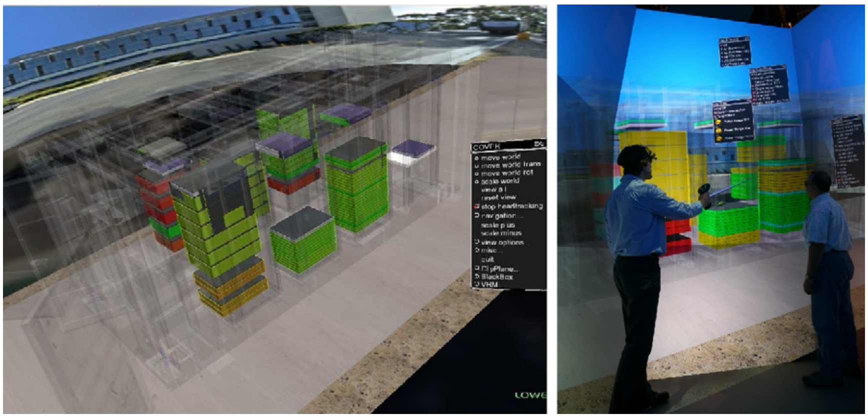


FIG. 14. Left: Bird's eye view of the container with translucent walls, using the same camera perspective as in Fig. 13. Right: The container displayed in the StarCAVE.

The main purpose of the virtual container is to allow scientists who measure the power consumption and performance of their algorithms, to view these measurements spatially. The virtual environment can give clues on whether one hardware component affects another by, for instance, blowing warm air into the other machine's air intake, in which case that machine might run warmer than it would without the hot machine. These side effects can be much more easily detected in a spatial model of the hardware components than in the Web interface. The 3D visualization application can also be used for educational purposes or to reduce maintenance times, by indicating to technicians, the hardware that exhibits problems.

4.1.2 Data Visualization

One result of our research on how to effectively display the data from the GreenLight instrument is about the display of this data in the 3D model. Whereas we first implemented a method to display the data as 2D graphs, which we displayed in the 3D environment, we found that this method made it hard to associate the graph with a certain component in the container, even if the component was highlighted. Also, only a limited number of graphs can be displayed at the same time. Therefore, we decided to put the information about the devices directly on their visual representation by coloring their surface with an appropriate color scheme. This approach makes it harder to see small differences in the data because it is hard to distinguish small differences in color; however, it allows the state of all the hardware to be displayed in the container at once.

Our visualization method of coloring the components in the container led to implementation of the "X-ray mode." Originally, all of the geometry of the container and its components was opaque, so the user had to pull out the computer racks to see the components; this was cumbersome and did not allow simultaneous viewing of all devices. Making the noninstrumented geometry in the container translucent, allowed for simultaneous viewing of all the instrumented geometry, without the need for moving racks. The user is allowed to switch between opaque and X-ray mode because it is sometimes useful to see the container the way it looks in reality, for example, technicians can train for where to find defective components and how to get to them. The user of the interactive system can choose which types of IT components should be selected: all components in the container; or only a subset: switches, storage systems, rack PCs; PC towers; or other components.

During our research and development of the virtual data center, we found that the configuration of the components in the container changes quite frequently, mostly when new components are added and old ones get replaced. Previously, our human 3D modeler updated the 3D model with a CAD tool to represent the new configuration of devices, and in addition, the sensor data was remapped to reflect these changes. To allow a system administrator without 3D modeling skills to make these changes, and to make them much faster, a database was implemented to describe the configuration of the devices in the container. This approach allows devices to be added, moved, and removed quickly and easily without the need for a 3D modeling tool. The 3D modeling tool is still needed when new types of devices are added to the container for which a 3D model has not yet been created.

4.1.3 Conclusions

We presented a visualization tool to view live data from a Web server, mapped onto geometry in a virtual 3D replica of Calit2's mobile data center. This approach allows users to easily view the data acquired by the power consumption sensors in the data center, and it shows clearly where exactly the hardware is installed. This can reduce the amount of in-person visits to the data center, which can play an important role if the data center is located far away from the scientists and engineers using it.

The current version of the interactive software application leaves various things to be desired. In the future, we plan to use the virtual environment not only to view the state of our data center, but also to actively control it. This will require the addition of a data path back to the data center, along with access control mechanisms, but it will be a very intuitive, yet powerful way to administer a data center. We also plan to install additional sensors to be able to obtain a more accurate spatial map of the temperature distribution in the Sun MD. This will help optimize the spatial arrangement of the IT devices in the container to minimize the HVAC requirements.

5. Information Visualization

This section showcases an application designed to display data in VR which does not have inherent 2D or 3D structure. This type of data visualization is often called "information visualization." The challenge of this type of data is to find effective mappings from the multidimensional data domain to the three spatial dimensions in VR systems, as well as time which can sometimes be used as a fourth, independent dimension. We also utilize the unique features of VR, such as 3D stereo, immersion, surround, high-resolution screens, head tracking, and 3D input, to make the higher-dimensional data set more accessible.

5.1 How Much Information

Data is being created at exponentially increasing rates, driven in part by the decreasing costs and increasing number of embedded processors sold each year. Stories abound of scientific data streams are not analyzed due to lack of time. Commercial enterprises have for decades collected operating data in manufacturing, sales, and elsewhere that they were unable to analyze further.

In an attempt to exploit this profusion of data, enterprises now invest in "Business Intelligence" (BI) capability. This typically includes a data warehouse with the ability to retrieve data very flexibly, and software to search for patterns and trends, both by machine learning and by assisting human analysts. This "data analytics" software, however, is primarily still based on old models of information presentation, such as spreadsheets.

The plunging cost of digital hardware now enables alternative ways of presenting and interacting with information. Office workers have access to hardware more powerful than engineering workstations a decade ago; megapixel color displays driven by powerful 3D graphics cards (hundreds of parallel processors running at speeds >1 GHz [25]) and attached to terabyte storage now add less than \$1000 to the cost of an office computer.

Our research project looks beyond the current desktop environment, to what will be available in a few years. Falling costs should enable dramatically new interfaces. However, application-level analytic software is moving only slowly to take advantage of these interfaces. We therefore built prototypes of an application for visual analytics for the StarCAVE, building on the following key features of it: 3d graphics, stereo vision, 360° surround projection, and user head tracking. Our hypothesis is that immersive VR systems can display complex nonspatial data more effectively than 2D monitors. While we have yet to do a formal user study on this hypothesis, others did similar studies [26] and found that 3D visualization can have measurable benefits over 2D visualization.

The StarCAVE costs approximately \$1 million when it was built in 2007. Commercial off-the-shelf versions of most of its components, albeit with significantly lower resolution and less immersion, are now accessible for a few thousand dollars per screen. An example of such a system is the NexCAVE.

5.1.1 Nonspatial Data

The StarCAVE and other VR systems have generally been used for scientific data, and particularly data with a natural spatial layout. Examples include the data sets used in the rest of this chapter, as well as architectural models, machine parts, medical CT and MRI data, or simulation results of blood flow in arteries. It is natural to show such data in 3D space, because the mapping from data values to 3D coordinates is inherent in the underlying problem.

However, with other kinds of scientific problems, and for most enterprise problems, the data does not have a natural arrangement in a 2D or 3D pattern. For example, each observation could be an individual in a company, a unit on a production line, or a different product. Each observation has multiple columns of quantitative data, again with no natural spatial organization. The question is then how to take advantage of enhanced spatial visualization in the StarCAVE to better understand the data.

5.1.2 Data Set

We analyzed a large data set with over 60,000 top-level and 5 billion low-level observations. It provided the hard drive structure of employees' computers at Microsoft over 5 years [27]. This data set gives the topology of the file systems of each computer, including number of files in each directory, parent directory, and children directories if any. For each file, it includes the file type, file size, and various time-stamp information, including creation date, last modification date, and last access date. This data set allowed us to analyze how employees organized, and to a limited extent how they used, their computers.

5.1.3 Related Work

Ware and Mitchell [28] showed that on interactive 3D displays, graphs can be an order of magnitude larger than in 2D and still be read with the same accuracy by users. We employ this idea by displaying the file system hierarchy as a 3D graph and add 6° of freedom navigational interaction and head tracking, far exceeding the rotational effect they used. Our system is based on the idea that large multidimensional databases are best queried by using an interactive visualization tool, as previously discovered by Stolte et al. [29].

A similar approach to our graph layout algorithm was published by Robertson et al., who created cone trees [30] for a similar purpose. Our approach differs in that our graphs are centered around a selected node and grow around it in all directions, more like Lamping et al.'s hyperbolic tree browser [31], but in 3D. Cone trees grow in a linear direction, not utilizing 3D space as equally balanced as our graphs. Our general idea of mapping nonspatial data into a 3D space, however, is not new. Russo Dos Santos et al. [32] did this for cone trees and other visualization metaphors, but their approach does not involve 3D interaction beyond navigation, whereas we support direct interaction with the data as well.

Parker et al. [33] suggested that using direct 3D interaction and 3D widgets can have benefits over more traditional visualization methods. Our system was built with the same motivation, but for a different type of data, and with a very different implementation of visualization methods.

We also implemented box graphs, which map multidimensional data into a lower dimensional space. In this case, we map 6D data into 3D space. This general idea is not new, a good overview of such mappings for 2D was given by Ward [34] in his XmdvTool. Our approach differs in that we map into 3D space instead of 2D. Our box graph is similar to an interactive 3D scatterplot [35,36], but it uses box glyphs instead of points, allowing six dimensions to be displayed for each observation.

5.1.4 Our Visualization System

Queries over more than 60,000 data records are best done by a database. We use a MySQL database with custom indices to allow for real-time queries from the StarCAVE. For the visualization of the directory trees, and their analysis, we created three novel visualization methods: a hyperbolic node graph, a stat box, and a box graph.

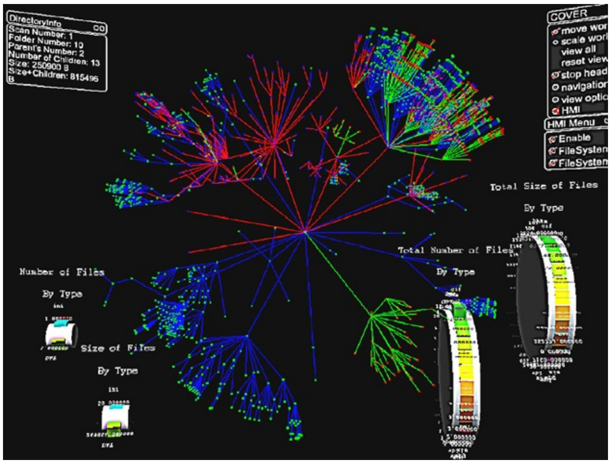
5.1.4.1 Hyperbolic Node Graph.

Similar to Lamping et al.'s hyperbolic tree browser [31], we created a 3D tree which at the beginning is centered around the root directory of the selected user's hard disk. Directories are displayed as spheres, with lines connecting parent and children directories. The user can then choose to add another user's directory tree, whose directories will be displayed in a different color than the first user's. Common directories will be displayed in a third color. Even though the data set we had was anonymized, it used consistent hash codes so that we were able to identify common directories and file extensions.

As opposed to a hyperbolic tree, we map our directory nodes onto invisible, concentric, and evenly spaced spheres around the root directory. Since we use a surround visualization system, there is no need to limit the virtual width, as opposed to 2D graphs which cannot extend beyond the edge of the screen or paper they are displayed on.

When the user clicks on a directory node, the graph automatically recenters on this directory. This allows the user to study a particular part of the tree, even one that is deep down in the tree, while still taking advantage of the 3D space around the user to spread out the nodes of interest.

As shown on the screenshot in Fig. 15, a click on a directory node with a different button brings up four wheel graphs, showing information about the file types in the selected directory, as well as in the entire subtree including and below the selected



DirectoryInfo

Scan Number: 1
Folder Number: 10
Parent's Number: 2
Number of Children: 13
Size: 250900 B
Size+Children: 815496
B

FIG. 15. Screenshot of our 3D hyperbolic graph with node and edge highlighting, as well as wheel graphs and menus.

directory. For each of the above, there is one graph showing the number of files, and one showing the accumulated file size. The wheel graphs are cylinders with bars extending out from the center, the length of the bars indicating file size or number of files of a certain type. We distinguish 11 different file types, based on file extensions: audio, binary, library, text, code, compressed, internet, office, image, video, and other. The user can select between a linear scale and a logarithmic scale for the length of the bars. We find that the wheel graphs occlude less of the scene than traditional bar graphs would, while still conveying the data similarly well. In addition to the wheel graphs, we display a text box which lists file sizes and other information about the selected directory in numeric form. Figure 16 shows what our hyperbolic graph looks like in the StarCAVE.

5.1.4.2 Stat Box.

In order to answer questions about how many of the files on disk were accessed more recently, we created a novel visualization widget, which we call the stat box. The stat box consists of a height field where the last file access date is mapped to the x-axis (starting on left with date of directory scan, older files to the right), the user's directories mapped to the y-axis, and the number of files per directory mapped to the z-axis (height). File age can be selected to be one of the three dates associated with each file: creation date, last modification date, or last access date.

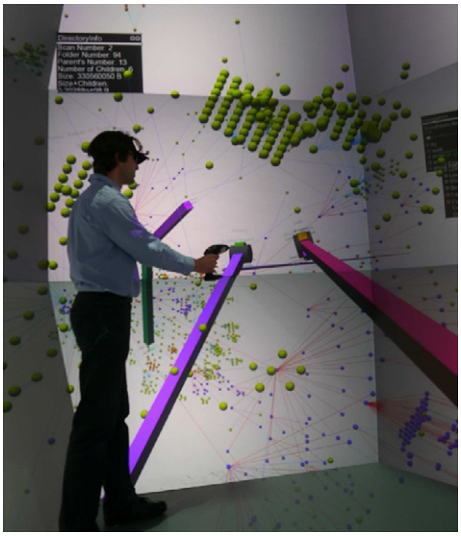


FIG. 16. Hyperbolic graph with wheel graphs in the StarCAVE.

In addition to viewing the stat box as it is, the user can choose to select a certain position on the *x*-axis to set a boundary for file age. Once the user selects this value, it is visually shown as a translucent plane parallel to the *y*/*z*-plane. This plane can be moved along the *x*-axis with the analog joystick on the 3D input device. Whenever this plane is displayed, all those directories containing files with less than the selected value will be highlighted in the corresponding hyperbolic graph; files with greater values will be dimmed.

5.1.4.3 Box Graph. The box graph (see Fig. 17) is similar to a scatter plot, but with rectangular boxes instead of points. The boxes are large enough for the user to distinguish differences in shape and color. This graph type allows us to visualize seven dimensions of data records in one graph: three dimensions for the location of the box; three for width, height, and depth of the box; and one for the color. We designed our box graph to map data dimensions to visual parameters as follows. *x*-axis = file type, *y*-axis = user bin, *z*-axis = average file age, width = average number of files per directory, height = number of directories, depth = average file size, color = file type. We map file type both to the *x*-axis coordinate and box color, so that it is easier to distinguish different rows in the graph from one another. We choose the depth of the boxes to be the average file size, so that the box volume indicates the total number of bytes a user has of the respective

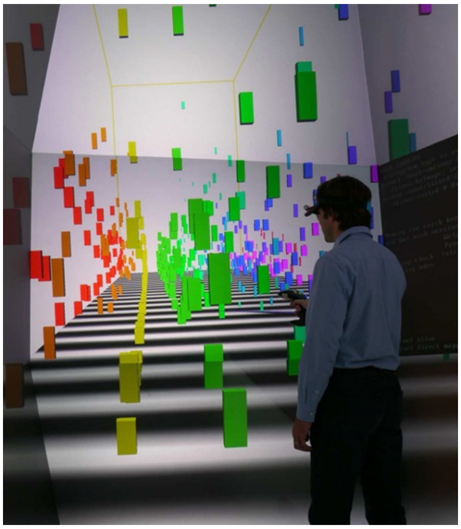


FIG. 17. Box graph in the StarCAVE.

file type. (Total bytes = number of directories × average number of files per directory × average number of bytes per file.)

Note that we do not draw a box for the file types of individual users, but groups of users. For example, we sort the users of each year's survey by the total number of bytes they store on their hard disk. We then bin a number of users together and map them to one unit on the *y*-axis. This way we can view the data of an entire year in one graph.

We find that when looking at the box graph in the StarCAVE, we can visualize a much larger amount of data than we can at the desktop. We believe that this is because of the surround nature of the StarCAVE, which allows us to get close to one part of the graph, but still see all the rest of it in the distance. Also, head tracking allows us to "look around" boxes which would otherwise occlude other boxes behind them.

In the menu, the user can interactively scale the box size, or any of the grid coordinate axes to find a good balance between size of the graph and its density. In addition, the user can choose to use a logarithmic scale for these values. The user can also choose to display a striped grid at the bottom of the graph, which helps estimate distance from the viewer. Another option are base lines, which are vertical lines connecting each box to the bottom of the graph. Rulers can be selected to help judge the size of the boxes in each dimension. However, they also considerably impact the frame rate because every box gets a set of lines attached to it.

5.1.5 Conclusions

We created an interactive visualization system for file systems, with the ability to compare many users' file systems to one another and to display up to 6.5 variables simultaneously for each observation. Through a combination of direct interaction with data points and menus for more abstract functionality, we are able to use the system entirely from within our virtual environment, without the need to resort to keyboard and mouse. Our visualization system makes good use of the immersive quality of surround VR systems, as well as direct 3D interaction.

6. Cultural Heritage Visualization

In this section, we describe a project we did with Calit2's cultural heritage researcher Maurizio Seracini. It exemplifies how this science can benefit from modern visualization technology, even if the objects studied are hundreds of years old.

6.1 Walking into a da Vinci Masterpiece

The Adoration of the Magi is an early painting by Leonardo da Vinci 18. Leonardo was given the commission by the Augustinian monks of San Donato a Scopeto in Florence, but departed for Milan the following year, leaving the painting unfinished. It has been in the Uffizi Gallery in Florence since 1670. Three hundred and thirty three years later, in 2003, cultural heritage researcher Maurizio Seracini, sponsored by the Kalpa Group of Loel Guinness, got exclusive access to this masterpiece and took very high-resolution photographs of it under different wave lengths. Under each wave length, he took hundreds of overlapping close-up pictures of the artwork and carefully stitched them together to high-resolution image files with up to $25{,}267 \times 11{,}581$ pixels ($\sim$292 megapixels; Fig. 18).

In order to display them in our VR environments, we use OpenSceneGraph's terrain rendering engine VirtualPlanetBuilder [37] and treat the images as if they were terrains with an elevation of zero. VirtualPlanetBuilder uses the GDAL [38] library to subdivide the large image into smaller sections, called tiles, and stores these at a number of different resolutions, so that the rendering engine can use mipmapping to render them. This engine is designed to dynamically load the tiles in as needed and automatically select an appropriate mipmap level, while strictly sustaining an interactive frame rate of at least 20 frames per second. The tiled and mipmapped images are created with OpenSceneGraph's osgdem tool [37], which creates .osga files as its output, which encapsulate all tiles and mipmap levels in one

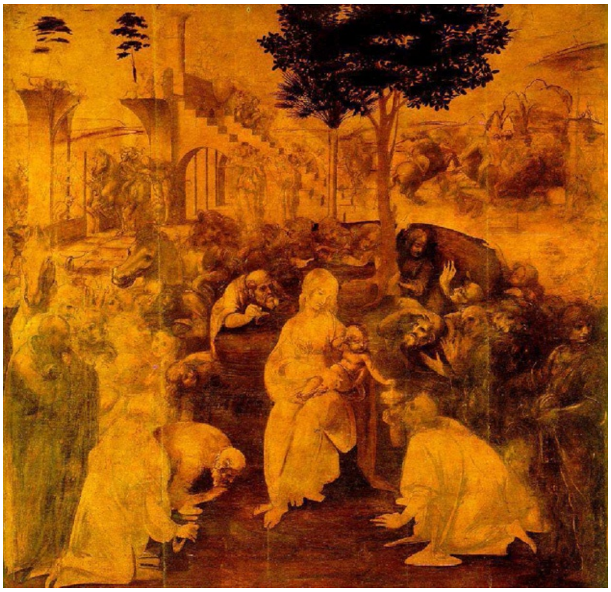


FIG. 18. Leonardo da Vinci: Adoration of the Magi. As seen at Uffizi Gallery, Florence, Italy.

file. The osgdem application is a command line program. A sample call to convert TIFF file Adoration-IR.tif looks like this:

```
osgdem -t Adoration-IR.tif -l 6
-o Adoration-IR.ive -a Adoration-IR.osga
```

6.1.1 The Application in a 3D Virtual Environment

Our COVISE plugin can load and display multiple such high-resolution spectral image at the same time and still operate at interactive frame rates. Each image can be separately positioned in the 3D world, and the user can click on them to move them around, as seen in Fig. 19A. Alternatively, the user can choose to display the images as a set, which means they will be stacked up and aligned with one another (Fig. 19B). The distance between the images can be adjusted with a dial and can be reduced to practically zero (exactly zero would introduce z-buffer fighting).

If in stack mode the images do not align well, the user can get into manual alignment mode. In this mode, one can place a cone-shaped marker on a feature point which occurs in each spectral image, as seen in Fig. 20. Once all cones are

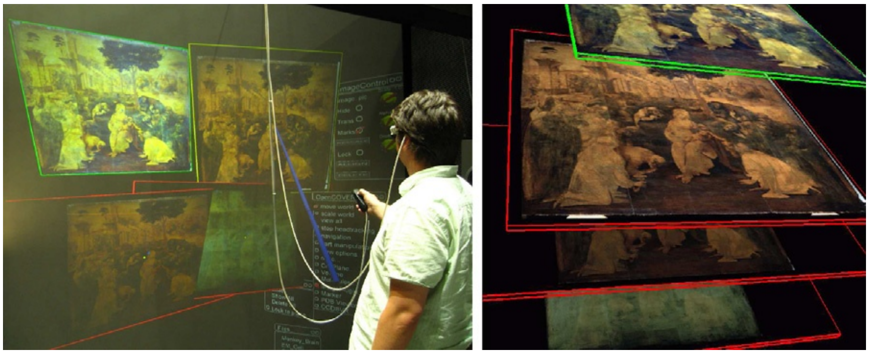


FIG. 19. Philip Weber at Calit2's stereo display wall, viewing four spectral versions of da Vinci's "The Adoration of the Magi" (visible light, ultraviolet, infrared, X-ray). On the left (A), they are being viewed individually, on the right (B), as a set.

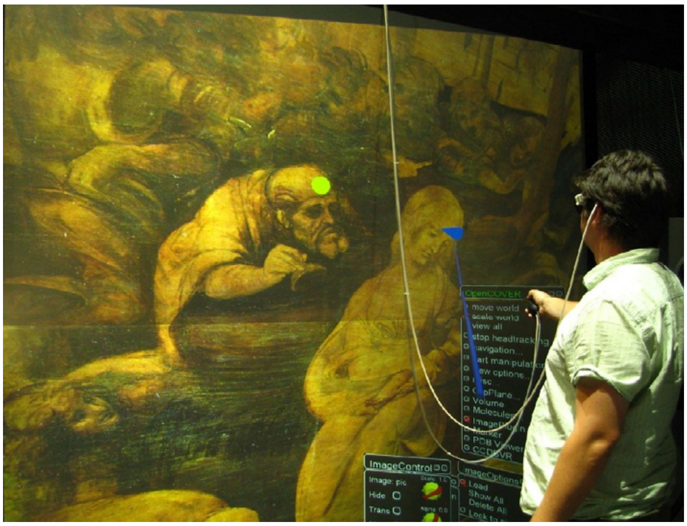


FIG. 20. Philip Weber placing alignment markers on "The Adoration of the Magi."

placed, the images are shifted so that the selected feature points align. With this method, we can only compensate for translational misalignment. Scale, rotation, and potential intraimage warping cannot be compensated for with our system but would be feasible to implement; we simply did not encounter such misalignment with our images.

When the images are all in one plane, the user can switch to a mode in which they can be made translucent, depending on the viewer position, in the following way: the physical space the viewer operates in is subdivided into as many zones as there are images. Each zone corresponds to one of the spectral images and is characterized by its distance from the screen. The software then dynamically adjusts the opacity of the images such that the images which corresponds to the zone the user is in will be displayed fully opaque when the user is in the center of the zone, and it will be more translucent the further the user is away from the center. At the edge of the zone, the image will be at 50% of its opacity, and in the next zone's center, the image will be entirely transparent, making it invisible. This allows the user to select one of the images by adjusting their distance from the screen, while the transitions in between are smooth. In many demonstrations where we let the visitors try out the application, this showed to be a very intuitive way to explore the various spectral images. Figure 21 shows a detail view of the Adoration of the Magi in visible light and infrared.

6.1.2 The Application on a 2D Display with 3D Input

We first created this application for 3D display systems, which make it an immersive experience, especially in free-floating image mode where all spectral images can be moved separately, so that the user can put them side by side to compare them with one another, or even arrange them around the user. However, in stack mode when the user looks at the stack perpendicularly, 3D plays no longer a role, a 2D display system would do just as well at displaying the images.

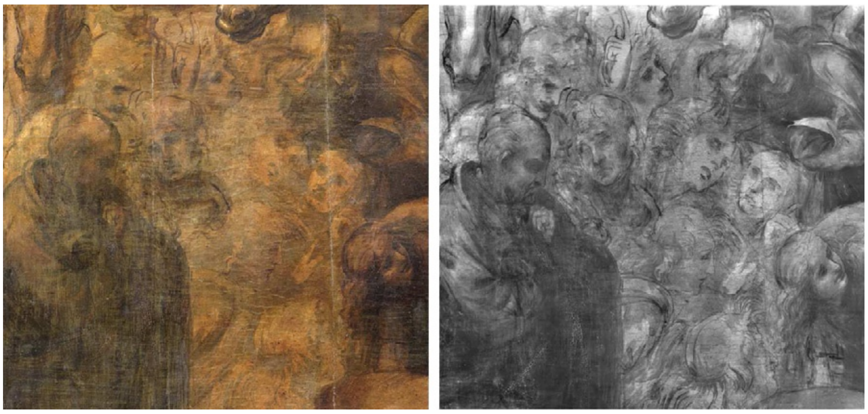


FIG. 21. Detail view of the "Adoration of the Magi" under different spectra of light. Left: visible light. Right: infrared. Images provided by Maurizio Seracini, copyright by Loel Guinness.

This led us to install the application in our auditorium, where it runs on a high-end graphics PC with dual Nvidia Quadro 4500 graphics cards. A 10,000 lumen Sony SRX-R110 projector displays the image on the 32×18 ft screen at full 4K resolution of 3840×2160 pixels. On the stage, we installed a tethered Ascension Flock of Birds tracking system with a Wanda input device to allow the presenter to interact with the high-resolution painting in real time, using the same 3D input method as in the initially used 3D VR environment. Only now, user and audience do not need to wear glasses, which helps given that the auditorium can hold up to 200 people. Because the auditorium environment requires us to use a fixed viewpoint when rendering the image, given the size of the audience, we do not use head tracking in it. Therefore, the user position cannot be derived from the head position. Instead, we use the wand position as the user position for our fading effect. We adjusted the width of the viewing zones to match the size of the stage, so that the user can "walk into the da Vinci masterpiece." The user can also use the 3D input device to pan the images left/right and up/down by clicking on a point in the painting with the virtual stick, and dragging it into the desired direction. Scaling is implemented by twisting one's hand. Clockwise rotation scales the image up, anticlockwise scales it down. Figure 22 shows a typical demonstration situation with a narrator and a separate system operator.

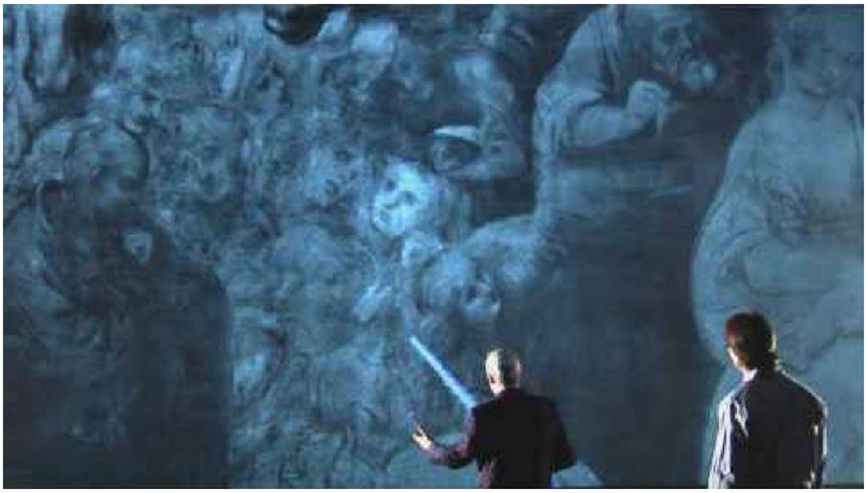


FIG. 22. "Walking into a da Vinci masterpiece" demonstration on the 4K screen in Calit2's auditorium.

This combination of a 2D display and 3D input has shown to be very effective in demonstrations and motivated us to consider it again for future applications. The main benefit is that the software can be shown to a much larger audience than in, say, the StarCAVE. The 3D input device with its six degrees of freedom allows for a much greater variety of input than a traditional mouse.

7. Conclusion

We presented five software applications developed at Calit2 over the past 5 years, which we consider best practice within five application categories. Each of these applications utilizes the unique features of VR and could not be used equally well at the desktop with keyboard and mouse. We believe that, especially with dropping cost for VR hardware, software will more and more be developed specifically for VR environments, as opposed to first for desktop systems, and then adapted to VR at a later point. This development is going to give researchers, engineers, and consumers powerful software applications to solve their day-to-day problems in new, more intuitive, and more efficient ways. We are excited to be part of this development at this important time when VR is no longer cost prohibitive.

Acknowledgments

This publication is based in part on work supported by Award No. US 2008-107, made by King Abdullah University of Science and Technology, in particular, the development and construction of the NexCAVE. UCSD, through Calit2, receives major funding from the State of California for the StarCAVE and 4K facilities. UCSD also receives major funding from the National Science Foundation (NSF), award CNS-0821155, which supported the construction of the Calit2 AESOP display, part of the GreenLight Instrument. Other parts of this work have been supported by the National Center for Microscopy and Imaging Research (NCMIR).

References

[1] S. Fisher, M. McGreevy, J. Humphries, W. Robinett, Virtual environment display system, in: Proceedings of ACM Workshop on Interactive 3D Graphics, 1986, Chapel Hill, NC.

[2] T. DeFanti, G. Dawe, D. Sandin, J. Schulze, P. Otto, J. Girado, et al., The StarCAVE, a third-generation CAVE and virtual reality OptIPortal, Future Gener. Comput. Syst. 25 (2) (2009) 169–178.

[3] T. DeFanti, D. Acevedo, R. Ainsworth, M. Brown, S. Cutchin, G. Dawe, et al., The Future of the CAVE, Cent. Eur. J. Eng. 1(1), 2011.

[4] D. Rantzau, U. Lang, R. Rühle, Collaborative and interactive visualization in a distributed high performance software environment, in: Proceedings of International Workshop on High Performance Computing for Graphics and Visualization, Swansea, Wales, '96, 1996.
[5] D. Rantzau, K. Frank, U. Lang, D. Rainer, U. Wössner, COVISE in the CUBE: an environment for analyzing large and complex simulation data, in: Proceedings of Second Workshop on Immersive Projection Technology (IPTW '98), Ames, Iowa, 1998.
[6] L. Williams, Pyramidal parametrics, in: ACM SIGGRAPH '83 Proceedings, 1983.
[7] C. Tanner, C. Migdal, M. Jones, The clipmap: a virtual mipmap, in: ACM SIGGRAPH '98 Proceedings, 1998, pp. 151–158.
[8] OpenSceneGraph, Scenegraph based graphics library. http://www.openscenegraph.org, 2004.
[9] E. LaMar, B. Hamann, K. Joy, Multiresolution techniques for interactive texture-based volume visualization, in: IEEE Visualization '99 Proceedings, 1999, pp. 355–361.
[10] P. Bhaniramka, Y. Demange, OpenGL volumizer: a toolkit for high quality volume rendering of large data sets, in: Proceedings of the 2002 IEEE symposium on Volume Visualization and Graphics, 2002.
[11] S. Guthe, M. Wand, J. Gonser, W. Straßer, Interactive rendering of large volume data sets, in: IEEE Visualization '02 Proceedings, 2002, pp. 53–60.
[12] M. Weiler, R. Westermann, C. Hansen, K. Zimmermann, T. Ertl, Level-of-detail volume rendering via 3D textures, in: IEEE Volume Visualization 2000 Proceedings, 2000.
[13] I. Boada, I. Navazo, R. Scopigno, Multiresolution volume visualization with a texture-based octree, Vis. Comput. 3 (17) (2001) 185–197.
[14] S. Prohaska, A. Hutanu, R. Kahler, H.-C. Hege, Interactive exploration of large remote micro-CT scans, in: Proceedings of IEEE Visualization, 2004, pp. 345–352.
[15] Lawrence Livermore National Laboratory, Blockbuster, high-resolution movie player. https://computing.llnl.gov/vis/blockbuster.shtml, 2009.
[16] R. Stockli, E. Vermote, N. Saleous, R. Simmon, D. Herring, The Blue Marble Next Generation—a true color Earth dataset including seasonal dynamics from MODIS, Technical Report, NASA Earth Observatory, October 2005.
[17] R. Sutton, A. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998.
[18] PDB, Protein Data Bank. http://www.pdb.org, 2010.
[19] W. DeLano, The PyMol Molecular Graphics System, DeLano Scientific, San Carlos, CA, 2002. http://www.pymol.org.
[20] D. Lupyan, A. Leo-Macias, A. Ortiz, A new progressive-iterative algorithm for multiple structure alignment, Bioinformatics 21 (15) (2005) 3255–3263, Oxford University Press.
[21] P. Greenlight, Home Page. http://greenlight.calit2.net, 2010.
[22] Oracle, Sun Modular Data Center. http://www.sun.com/service/sunmd/, 2010.
[23] J. Schulze, Advanced monitoring techniques for data centers using virtual reality. SMPTE Motion Imaging Journal, ISSN 0036-1682, July/August, 2010.
[24] C. Farcas, F. Seracini, GLIMPSE Home Page. http://glimpse.calit2.net.
[25] N. G. 480, High-end graphics card. URL: http://www.nvidia.com/object/product_geforce_gtx_480_us.html, 2010.
[26] M. Tavanti, M. Lind, 2D vs. 3D, implications on spatial memory, in: Proceedings of IEEE Symposium on Information Visualization, 2001, pp. 139–145.
[27] N. Agrawal, W. Bolosky, J. Douceur, J. Lorch, A five-year study of file-system metadata, ACM Trans. Storage 3 (3) (2007) 9:1–9:32. http://research.microsoft.com/apps/pubs/?id=72885.
[28] C. Ware, P. Mitchell, Reevaluating stereo and motion cues for visualizing graphs in three dimensions, in: Proceedings of Applied Perception in Graphics and Visualization, 2005, pp. 51–58, Vol. 95.

[29] C. Stolte, D. Tang, P. Hanrahan, Polaris: a system for query, analysis, and visualization of multidimensional relational databases, IEEE Trans. Vis. Comput. Graph. 8 (1) (2002) 52–65.

[30] G. Robertson, J. Mackinlay, S. Card, Cone trees: animated 3D visualizations of hierarchical information, in: Proceedings of the SIGCHI Conference CHI'91, 1991, pp. 189–194.

[31] J. Lamping, R. Rao, P. Pirolli, A focus+context technique based on hyperbolic geometry for visualizing large hierarchies, in: ACM SIGCHI Proceedings, 1995, pp. 401–408.

[32] C.R.D. Santos, P. Gros, P. Abel, D. Loisel, N. Trichaud, J. Paris, Mapping Information onto 3D Virtual Worlds, in: International Conference on Information Visualisation, 2000, pp. 379–386.

[33] G. Parker, G. Franck, C. Ware, Visualization of large nested graphs in 3D: navigation and interaction, J. Vis. Lang. Comput. 9 (1998) 299–317.

[34] M. Ward, XmdvTool: integrating multiple methods for visualizing multivariate data, in: IEEE Visualization Proceedings, 1994, pp. 326–333.

[35] B. Becker, Volume rendering for relational data, in: IEEE Symposium on Information Visualization, 1997, pp. 87–90.

[36] R. Kosara, G. Sahling, H. Hauser, Linking scientific and information visualization with interactive 3D scatterplots, in: Proceedings of WSCG, Plzen, Czech Republic, 2004.

[37] VirtualPlanetBuilder, Terrain Database Creation Tool. http://www.openscenegraph.org/projects/VirtualPlanetBuilder, 2010.

[38] GDAL, Geospatial Data Abstraction Library. http://www.gdal.org, 2010.

[39] osgdem, OpenSceneGraph's utility program for reading geospatial imagery and digital elevation maps. http://www.openscenegraph.org/projects/osg/wiki/Support/UserGuides/osgdem, 2010.