EUROGRAPHICS 2001 / A. Chalmers and T.-M. Rhyne
(Guest Editors)

Volume 20 (2001), Number 3, pp. C-8–C-16

JAPE: A Prototyping System for Collaborative Virtual Environments

Oliver G. Staadt, Martin Näf, Edouard Lamboray, Stephan Würmlin

Computer Graphics Group
Computer Science Department
ETH Zurich, Switzerland

E-mail: {staadt, naef, lamboray, wuermlin}@inf.ethz.ch
http://graphics.ethz.ch

Abstract

We present JAPE, a flexible prototyping system to support the design of a new advanced collaborative virtual environment. We describe the utilization of different hard- and software components to quickly build a flexible, yet powerful test bed for application and algorithm development. These components include a 3-D rendering toolkit, live video acquisition, speech transmission, and the control of tracking and interaction devices. To facilitate the simultaneous design of applications and algorithms that take advantage of unique features of new collaborative virtual environments, we provide the developer with a flexible prototyping toolkit which emulates the functionality of the final system. The applicability of JAPE is demonstrated with several prototype applications and algorithms.

1. Introduction

Advanced virtual environments, such as *spatially immersive displays* (SIDs) [4, 9], make great demands on application programming interfaces, often referred to as virtual reality toolkits. This is due to special and often unique features provided by those environments that make it necessary to either extend existing multi-purpose toolkits or to design new toolkits which are tailored to specific requirements of the environment.

We are currently developing a novel *collaborative virtual environment* (CVE), the *blue-c,* which combines immersive projection with real-time video acquisition and advanced multimedia communication [22]. Eventually, multiple blue-c portals, connected via high-speed networks, will allow remotely located users to meet, communicate and collaborate in a shared virtual space.

The development of large immersive environments over the past decade has revealed that good applications are crucial for the success of a VR system. Hence, instead of adapting existing applications to a mere technology-focused environment once it is completed, the application designer should be able to influence the development of the VR system right from the beginning [21].

Unfortunately, design and implementation of a full-featured *application programming interface* (API) and the actual VR environment are often carried out simultaneously. Thus, it is necessary to provide application designers with an alternative design and prototyping environment that emulates important features and concepts that will eventually be available in the final API. Moreover, it will provide developers of core system components with a flexible test bed for novel algorithms and methods.

Obviously, the provision of the prototyping environment should not be delayed until important design and implementation decisions of the VR system have already been taken. Therefore, reuse of immediately available hard- and software components, which constitute the prototyping environment, allows for application design at an early stage of the project.

In this paper, we present *JAPE*, a prototyping environment which has been built to support application design and algorithm development for the blue-c project. Our goal is not to introduce a novel system, but to reuse and combine suitable components in short time. Note further that we do not intend to build our blue-c API directly on top of this prototyping system. Some of the JAPE components, will

eventually be integrated into the final API. This will ensure a smooth transition from JAPE to the blue-c API.

The remainder of this paper is organized as follows. After a brief summary of the blue-c project in Section 2, we discuss related work in Section 3. Section 4 outlines the different components comprising the prototyping environment, followed by a detailed description of the core system and individual components. The point-based rendering method introduced in Section 7 is an example for the integration of a new component into JAPE. Finally, we present some prototype applications in Section 8.

2. The *blue-c* Project

The blue-c is a joint research project between several institutes at ETH Zurich, including the Computer Graphics Group, the Center of Product Development, the Computer Aided Architectural Design Group, and the Computer Vision Group.

Our research aims at investigating a new generation of virtual design, modeling and collaboration environments. The blue-c system foresees simultaneous acquisition of live video streams and projection of virtual reality scenes [22]. Color representations with depth information of the users will be generated using real-time image analysis. These *human inlay* representations will be integrated into a computer generated virtual scene, that will be projected in an immersive display environment. Thus, unprecedented interaction and collaboration techniques among humans and virtual models will become feasible.

Several applications that take advantage of the novel features of the blue-c are currently being designed. Developers from different application fields, including architecture and mechanical engineering, utilize JAPE for the development of prototype applications and for the study of new interaction techniques. The final applications, however, will be designed for the blue-c platform. The blue-c project has started in April 2000 and its first phase is expected to be completed by spring 2003.

3. Related Work

During the past decade, many different architectures for building virtual reality applications have evolved. We have analyzed a variety of commercial toolkits as well as research systems.

Commercially available toolkits include Sense8 (http://www.sense8.com/), which provides an easy to use interface for non technical users, and MultiGen-Paradigm Vega (http://www.multigen.com/), which offers powerful tools to create realistic worlds for real-time simulations, including sensor simulation. Examples for non-commercial architectures are the VR-Juggler [1] and the MR Toolkit [20].

Recent research efforts concentrate on building distributed virtual reality environments. Although various distributed military simulations have been developed during the past decade [3, 23], solutions for computer supported collaborative work (CSCW) in virtual reality environments is still an area of very active research. A basic toolkit with focus on networking primitives, avatar handling and basic database functions in the CAVE [4] environment is CAVERNsoft [10, 14]. In addition, CAVERNsoft provides a set of Performer-based high-level modules, which can accelerate the construction of collaborative virtual reality applications. In the future however, we plan to use advanced communication technology and protocols with QoS support. CAVERNsoft, concentrating on TCP, UDP and HTTP, does not fulfill all the requirements we established for the blue-c system.

Avango [24] (a.k.a. Avocado) takes a different approach at distributed virtual reality. It replicates a Performer scene graph across all sites over the network. In addition to this base functionality, it supports a new scripting language for rapid application development. TELEPORT [2] is a tele-presence system, which allows for the integration of a real person into a virtual office environment. A similar approach is taken with the *Office of the Future* project [17]. Both systems, however, are currently restricted to uni-directional operation.

The Distributed Interactive Virtual Environment (DIVE) [5] is a toolkit covering several important aspects of virtual environments, including graphics, audio and networking. The MASSIVE System [6], developed at the University of Nottingham, also provides a framework for distributed virtual environments and puts special emphasize on a spatial model of interaction for awareness management. MASSIVE and DIVE were both designed for Internet-based virtual environments with hundreds, or even thousands of users. Hence their target applications are different from those of the blue-c system, where only a few sites will be interconnected via a high-speed network, and where the applications will run on high-end graphics hardware.

Ongoing research at the University of Manchester lead to the development of novel techniques for display and interaction management (Maverik) [8] as well as to an interesting paradigm for large-scale multi-user virtual environments (Deva) [15]. The Deva implementation, which covers the communication aspects of the distributed virtual environment, has not yet been released.

We have come to the conclusion that none of the above toolkits were immediately suitable for the blue-c without significant extensions and modifications. The final blue-c application building toolkit will thus be built directly on top of OpenGL Performer. Consequently, we have decided not to employ any of those toolkits for the purpose of application design and prototyping.

4. System Overview

The JAPE environment comprises the *system core* and several interchangeable *components* for remote data acquisition. See Figure 1 for a schematic overview of the JAPE architecture.

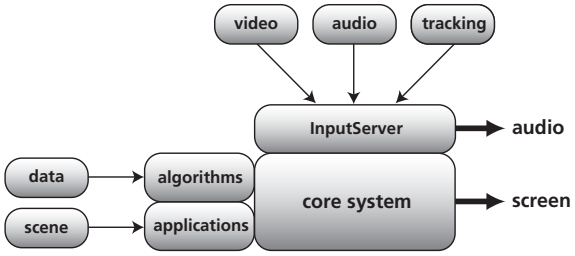

Figure 1: *Overview of the JAPE architecture.*

The system core is implemented using the OpenGL Performer library, which provides basic functionality such as scene graph management and rendering. Currently, Performer is the only library that efficiently supports multi-pipe rendering on an SGI Onyx, which is the target platform of the blue-c environment.

Three major components currently integrated with JAPE are video, audio and tracking. We have built a separate interface, the *InputServer,* that connects these components with the core system. This enables us to hide implementation details of the remote acquisition from the application.

A prototype application is built on top of the core system through which it communicates with individual components. When the application is started, the core system initiates the start of an InputServer instance on a number of remote client hosts, which are responsible for data acquisition and rendering (the rendering concept on remote clients will be introduced in Section 5.2). Note that the application is responsible for importing scene information.

The core system functionality can further be extended by adding additional algorithms. One example is the integration of a new point-based object as described in Section 5.4.

5. System Core

5.1. Shared scene graph

In our prototype environment, we use a scene graph based on the OpenGL Performer 2.4 library for storing and rendering the virtual scene. The development library includes thin wrapper classes for the scene graph, window management, and user interface elements such as mouse, keyboard and 3-D tracking devices. These are designed for rapid application prototyping while still allowing for direct access to the underlying Performer interfaces for advanced developers.

The prototype implements a pseudo-multiuser mode. Each user has his own view of the scene and user interface devices such as keyboard, mouse and trackers or 3-D mice. The application code, however, runs within a single process. This application process maintains a single copy of the scene graph. User interface input is multiplexed sequentially, allowing for consistent scene graph updates without additional synchronization.

5.2. Multi-head rendering

We exploit the multi-pipe rendering capabilities of Performer for rendering different views of the scene for each user, who is assigned a dedicated virtual graphics pipe which connects to any Xserver over the network. This does not only allow us to open multiple independent views on the same scene, but also to retrieve user interface input from standard X11 devices. This mechanism results in excellent rendering performance on our SGI Onyx 3200 system configured with two local consoles. A different configuration using SGI O2 workstations as remote terminals for scenes with reduced geometry and texture complexity still delivers adequate performance for prototype applications. Figure 2 depicts the concept for multi-head rendering with JAPE, illustrating the associated cull and draw processes of each virtual pipe.

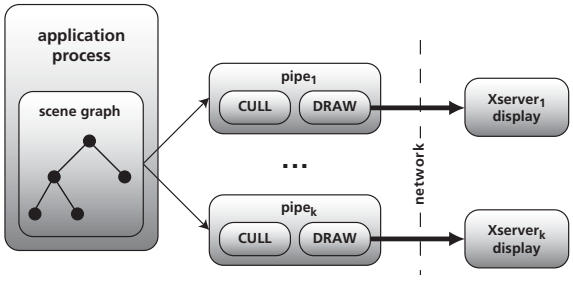

Figure 2: *The multi-head rendering concept.*

Note that the system does not impose a hard limit on the number of users participating in the virtual environment. In practice, however, the available network bandwidth for remote rendering limits scalability to a small number of users. Our application examples described in Section 8 have all been carried out with a two–user setup.

5.3. Video billboards

The visual representation of each user in JAPE is integrated into the scene graph. Each representation is derived from a Performer group node which hosts a textured billboard. The alpha channel in the texture is used for masking the person. This construction allows to treat the user representation just like any other standard geometry object.

Live-video data is acquired by the InputServer described in Section 6.

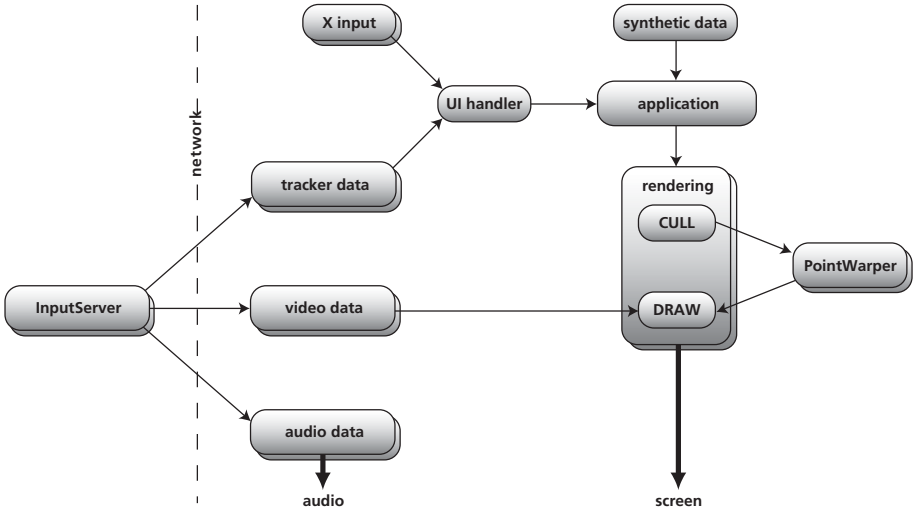

Figure 3: *Dataflow of a typical JAPE application.*

5.4. Point-based objects

In addition to application design, JAPE serves as a test bed for new core algorithms. A first example is a point-based rendering algorithm, which can be separated into two main components:

- A new pseudo-geometry object which is added to the scene graph. From the perspective of the rendering system, the point-object is a unit cube around the origin, which can be translated, scaled and rotated using regular dynamic coordinate system (DCS) group transformation nodes. Runtime optimization such as view frustum culling can be applied safely.
- Calculation engines for each point-based object are instantiated for each rendering channel. These engines continuously calculate new point sets depending on current camera and viewport parameters as well as object transformation. The parameters are updated during the culling phase for each frame. Callback routines render the most recent result in the drawing phase.

The algorithm developer can easily plug different types of image warping or point rendering algorithms into this framework without knowing the innards of Performer. This model allows for easy compositing including correct occlusion handling as long as the object remains within the limits of the unit cube.

5.5. Application integration

All user application code runs in a single process to relieve the application programmer from tedious and error prone synchronization tasks. The application is in charge of loading its data files, e.g., the virtual scene description. The overall organization of an application running concurrently with the JAPE processes is illustrated in Figure 3. The Performer rendering system spawns a cull and a draw process for each user. The virtual pipes are synchronized for a consistent view of the scene among all users.

A set of additional processes are spawned to decouple the achievable frame rate from acquisition and calculation tasks and to fully exploit the available processing power:

- There is one video receiver process for each video source. It manages two image buffers including status information in a Performer shared memory arena. One buffer always represents the current image, the other is the target for network transmission. The billboard texture is updated for each rendered frame using subloading in a draw process callback.
- Each point-based object has its own calculation process for each pipe. The calculation is triggered by a position and camera update from the cull process. The resulting vertices are again managed in a double-buffer. The draw process renders the most recent point set. This approach may lead to small inconsistencies between the calculated data and the actual view if the data takes longer than a single frame to calculate. This disadvantage is fully compensated by the improved visual impression gained with a higher frame rate, though.
- User interface input is handled asynchronously by separate acquisition processes. These processes either receive input information from the distributed InputServer components or poll the locally attached devices. The input handler manages queues and state objects to pass events and current state to the application process.
- Additional helper processes start and control the distributed InputServers.

The asynchronous data acquisition adds a small latency penalty because the available data is not always rendered immediately. At high frame rates, however, this latency penalty can be neglected, especially when considering the scene-update–to–render latency of three frames that is inherent to Performer in multi-processing mode.

5.6. Navigation and interaction

The development of the rapid prototype environment was initiated by the need for a test bed for new human-machine interaction paradigms. As a consequence, the system does not provide high-level interaction tools. The system provides only the raw input data such as mouse position, keyboard events and tracker data in a real world coordinate system. Additional services for object picking are provided by Performer. It is up to the application developer to interpret the user interface input in order to navigate in and to interact with the virtual world.

6. Remote Acquisition

The InputServer grabs live video images and tracker state information, and streams them to the application host. Moreover, it handles audio conferencing with its partner sites, i.e., another workstation running an InputServer (see Figure 3). The InputServer uses the *Real-time Transport Protocol* (RTP) [19] as its transport protocol above UDP/IP. We used Jori Liesenborgs' RTP implementation *jrtplib*, version 2.3 (http://lumumba.luc.ac.be/jori/jrtplib/jrtplib.html).

6.1. Live video

Live video acquisition is carried out on SGI O2 workstations with video option. The images of a user standing in front of a blue background are captured in RGBA format. A simple chroma-keying is performed to extract the user's texture. The RGB values of the background pixels can be set to the same value as their corresponding alpha value. This leads to long runs of identical byte values, which can be compressed efficiently with run-length encoding. After the acquisition and preprocessing steps, the image is streamed to the application host.

For run-length encoding, we used the implementation from the SGI digital media library. We analyzed three test cases with different amounts of background pixels:

- The user's body covers about 10–20 per cent of the input image. This is a typical situation where the user is allowed to move during the session.
- The user's body covers the complete image frame, i.e., the user is tracked by a camera, or is not allowed to move during the session.
- The image frame only covers the user's head, which would be typical in a "video-conferencing" style application.

Table 1 shows the results for the three typical test cases. As expected, the achievable compression ratio depends on the proportion of background information in the captured image. We measured approximate compression ratios of about 18:1, 3:1 and 1.6:1 for the three test cases respectively. Furthermore, the frame rate at which the InputServer operates depends on the image resolution. Already for half-frames, we risk having too small frame-rates for real-time interaction.

test case	image size	compression ratio	capture frequency
	288 x 360	19.3 : 1	14 Hz
	192 x 238	18.9 : 1	26 Hz
	144 x 192	16.8 : 1	27 Hz
	288 x 360	3.4 : 1	10 Hz
	192 x 238	3.3 : 1	23 Hz
	144 x 192	2.8 : 1	27 Hz
	288 x 360	1.7 : 1	7 Hz
	192 x 238	1.6 : 1	18 Hz
	144 x 192	1.6 : 1	26 Hz

Table 1: *Compression ratio and capture frequency for three test cases.*

Chroma-keying is very sensitive to the image quality of the input image, which in turn depends on lighting conditions and camera hardware. In order to obtain a good user extraction from the background, we used a consumer-level video camera. If correct background extraction is not important for the target application, inexpensive webcams can also be used. In order to increase the number of pixels covered by the user's body, we used the camera in an on-end configuration.

6.2. Tracking

We integrated a conventional magnetic motion tracking device, the *Flock of Birds* system by Ascension Technology Corporation. The tracker acquires position and orientation data up to 145 Hz per sensor, with an accuracy of 2 mm and $0.5°$, respectively. It is possible to operate multiple sensors simultaneously at high update rates.

We implemented a driver for the system, which allows us to choose the number of tracking sensors. In addition, we support sensors with integrated buttons like a space mouse. The InputServer collects tracker information which is streamed to the application host.

6.3. Audio

For audio communication, we integrated the open-source streaming application *Robust Audio Tool* (RAT, http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/) into the InputServer. Our choice of RAT was guided by the following features:

- Availability for all important platforms.

- Support of multi-party audio conferencing and use of RTP as transport protocol above UDP/IP.
- Large range of different speech coding algorithms from low-bandwidth linear prediction coding up to high-quality linear coding at 48 kHz sampling rate. For sample recovery or substitution in case of packet loss, some advanced techniques like redundant or interleaved coding as well as receiver-based repair strategies are supported.
- The open-source distribution allows for seamless integration of RAT into an application.

We linked RAT to the InputServer such that the audio conferencing application starts automatically without its own graphical user-interface. The necessary user operations that are normally required for starting listening using the workstation's loudspeakers and talking using a separate microphone are all performed in software. The choice of sampling frequency and of the audio codec as well as the playback configuration are specified in RAT's own configuration file.

Since we used the prototype only in a local-area-network, we chose linear encoding at 48 kHz without silence suppression. This produces a high quality audio stream at 768 kbps per user. For this prototype environment, synchronization of audio and video streams is not required.

7. Rendering of Point-based Objects

The use of video billboards as described in Section 5.3 is a simple way of integrating dynamic real-world objects into a three-dimensional synthetic environment. The blue-c system, however, will eventually integrate point-based objects that are reconstructed form multiple live video streams into the virtual environment. JAPE is used as a test bed for point-based rendering algorithms. We implemented a first approach to render static objects from multiple reference images with additional depth information using 3-D image warping [12]. With this method we want to give a proof of concept for compositing 3-D point-based objects into the geometry-based OpenGL Performer scene graph using the interface provided by JAPE (see Section 5.4).

7.1. 3-D image warping

In order to render images with depth information, we adapted the 3-D image warping equation [12]. Let $\dot{X}$ be a point in Euclidian space and (u_s, v_s) be its projection on the plane of image i_s. The projection of $\dot{X}$ into a new image plane i_t is given by

$$\bar{x}_t \doteq \delta(\bar{x}_s)\mathbf{P}_t^{-1}(\dot{C}_s - \dot{C}_t) + \mathbf{P}_t^{-1}\mathbf{P}_s\bar{x}_s ,$$

where $\doteq$ denotes projective equivalence. $\dot{C}_k$ is the camera's center-of-projection associated with image i_k and $\mathbf{P}_k$ is its 3×3 camera matrix. $\delta(\bar{x}_s)$ is the generalized disparity of a source image pixel (u_s, v_s), which is defined by the range value r of that point and its position to the center-of-projection:

$$\delta(\bar{x}_s) = \frac{\left|\mathbf{P}_s\bar{x}_s\right|}{r} .$$

The 3-D warping equation can be rewritten as rational expressions, which is computationally efficient and allows for incremental computations along individual scanlines. Note that the mapping of the 3-D warping equation is not homeomorphic, thus, resolving visibility is required. We have implemented the technique proposed by McMillan and Bishop [12], which essentially keeps a specific warping order, back-to-front, to resolve this problem. As an extension, we plan to implement a shear-warp factorization of the 3-D warping equation[13] to further increase performance of the warper by exploiting texture mapping hardware.

In contrast to systems like QSplat [18] or Surfel rendering [16], we have a sparser sampling of points since PAL video images have a resolution of 768×576 pixels, which leads to a total of 442,368 pixels. Our example objects cover only a fraction of the reference images (see Figure 4). This leads to reduced quality in the rendered images compared to the above systems, but the quality is still adequate for a prototype (see Figure 4b).

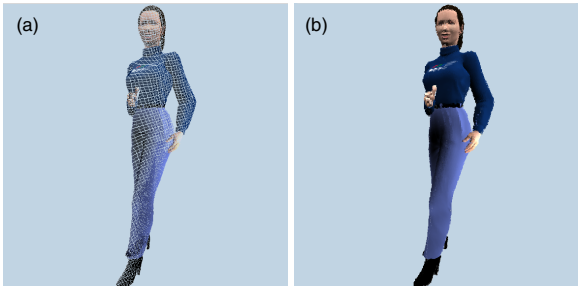

Figure 4*: *a: Warped point cloud with 45,000 points. b: Splatted surface.*

7.2. Multiple reference images

In order to render an object from arbitrary view points, a single reference image does not provide sufficient information. We represent an object with ten reference images, eight on a circle around the object and one on the bottom and on top of the object. Currently, we choose the two reference images from the nearest camera positions with respect to the position of the desired view. The number of reference images that are employed simultaneously could be increased to find an optimal balance for the total number of points in each warped image. Note that the number of points being warped is constant. The contribution of each reference image is inverse proportional to the distance from the position of the reference views to the desired view.

7.3. Surface reconstruction

Individual point samples are rendered using simple splat primitives. In an initial implementation, we employed a push-pull filter [7] which leads to high quality images at reduced performance. We decided to implement a splatting in the fashion of QSplat, where the footprint of the splats is fixed and the splat size can be variable. In our example prototype, we used fixed splat sizes for better performance.

7.4. Integration and compositing

Our method represents objects from multiple images with depth, which is similar to the concept of layered depth cubes [11]. The goal of the blue-c is the real-time reconstruction of humans and other real-world objects and their integration into virtual environments. The described rendering technique is the first step towards that goal.

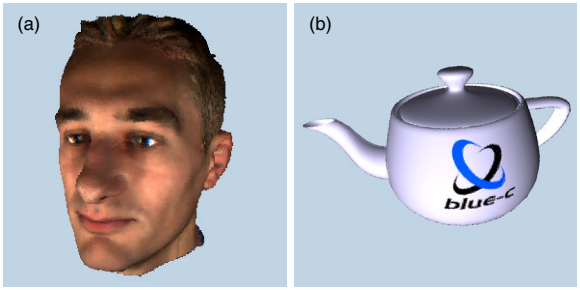

Figure 5*: *Examples for point-based objects. a: Head. b: Teapot.*

As described in Section 5.4, the warped object is inserted into the scene graph as a point-based object. The point samples are rendered very efficiently using `glVertex` primitives with splat size defined by `glPointSize`. Note that the OpenGL state change induced by `glPointSize` is expensive and reduces the rendering performance by 50 per cent. Thus, frequent changes of the splat size should be avoided. Table 2 lists performance measurements for the different models shown in Figure 4b and Figure 5. The frame rates were measured on a SGI Onyx 3200 using one 400 MHz MIPS R12000 processor and a single InfiniteReality3 pipe.

data set	warped points	fixed splatting	variable splatting	push-pull filter
woman	45,000	18.0 fps	8.0 fps	3.5 fps
head	70,000	14.5 fps	4.5 fps	3.5 fps
teapot	120,000	12.0 fps	3.5 fps	2.5 fps

Table 2: *Rendering performance for frame buffer resolution 768 x 576.*

Note that the performance is proportional to the number of points. When we render only a subset of 9,000 points of the woman model, we achieve a rate of 38 frames per second. The resulting visual quality is sufficient for a moving object.

8. Application Scenarios

8.1. Entertainment

Entertainment applications such as multi-user games are common scenarios for collaborative virtual environments. We have implemented a multi-user version of the popular *Moorhuhn* game (http://www.moorhuhn.de/).

Each user gets a view on the scene with chickens flying in front of a background composed of several layers of textured polygons. The chickens are simple textured polygons. Polygon vertices and texture coordinates are updated for every frame for a smooth and dynamic flight.

The framework gathers mouse and keyboard input from the consoles of the players. Mouse and keyboard handler objects store interaction events for each user. The application main loop handles all input once per frame and updates the global scene and individual user views accordingly. Hit-testing is based on the Performer channel picking feature.

Making the game multi-user capable required very little additional development effort thanks to the shared scene graph and trivial synchronization resulting therefrom. Figure 6 shows two people playing our multi-user version of Moorhuhn.

8.2. Virtual museum

Tele-teaching and virtual guides are promising application areas for collaborative virtual environments. The traditional tele-conferencing approach, which separates the representation of the remote users and the data, leads to an unnatural way of interaction. Recent immersive collaborative systems therefore try to integrate all users directly into the scene.

The virtual museum example depicted in Figure 7, illustrates the concept of an integrated video avatar as a tour guide, who explains the virtual exhibits to a point-based visitor. The position of the video billboard is given by the camera position of the remote user. The users can either navigate with the keyboard, mouse or the 3-D tracker. The users can talk to each other over the audio system and keep eye contact, leading to a natural way of conversation within the virtual scene.

9. Conclusions

In this paper we presented JAPE, a flexible prototyping system for collaborative virtual environments. Instead of employing a complex VR toolkit that might not be suitable for the final CVE, we demonstrated that it is possible to build a prototyping system from basic hard- and software components. This enables us to provide application designers of the blue-c project with a development framework that already includes important components of the final system.

Figure 6: *Picture of two authors playing the JAPE version of Moorhuhn.*

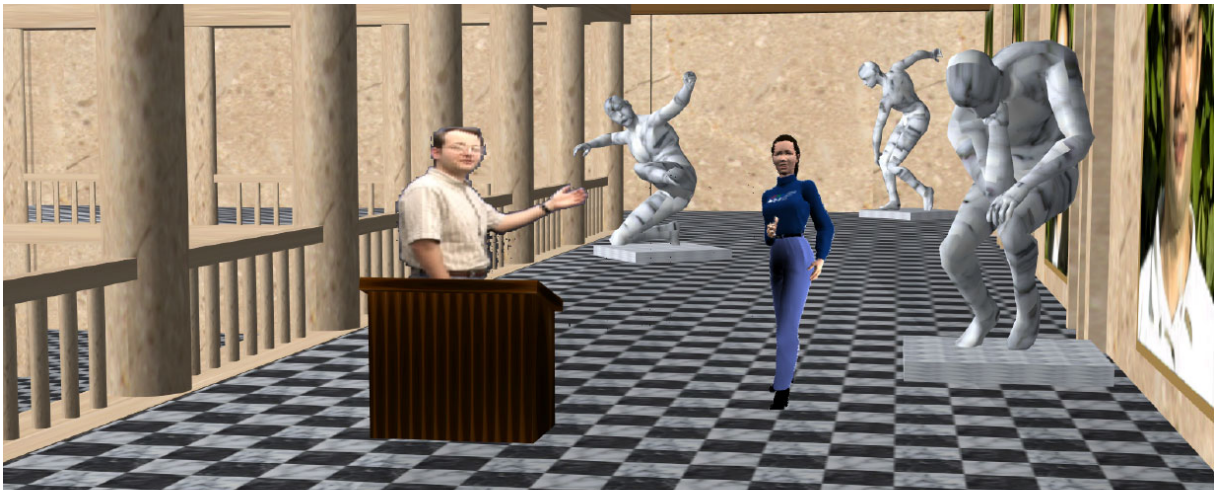

Figure 7*: *Example of the museum scene.*

The system core is based on the OpenGL Perfomer library, which allows us to exploit the multi-pipe rendering capabilities of our target hardware. A set of remote data acquisition components has been implemented to support video and tracking input as well as audio transmission. The core system's flexible interface allows us to incorporate additional algorithmic and data acquisition components.

Note that the application scenarios presented in this paper are not intended for the final blue-c environment. They demonstrate, however, the suitability of JAPE for rapid development of a variety of prototype systems. Applications for the blue-c are currently being designed using JAPE by developers in the fields of architectural design and mechanical engineering. We plan to migrate these applications to the final system once the blue-c API is available.

Acknowledgments

We would like to thank Markus Gross and all members of the blue-c team for many fruitful discussions. We also thank Stefan Hösli for implementing parts of the point-based object rendering method and Rolf Koch for the head data set.

The Moorhuhn data is courtesy of Phenomedia AG, Germany. This work has been funded by ETH Zurich as a "Polyprojekt".

References

1. A. Bierbaum, C. Just, P. Hartling, and C. Cruz-Neira, "Flexible application design using VR Juggler". *Technical sketch presented at SIGGRAPH 2000,* New Orleans, July 2000.
2. C. J. Breiteneder, S. J. Gibbs, and C. Arapis. "TELEPORT – An augumented reality teleconferencing environment." Proceedings of EUROGRAPHICS Virtual Environments and Scientific Visualization '96, pp. 41–49, 1996.
3. M. Capps, D. McGregor, D. Brutzman, and M. Zyda. "NPSNET-V: A new beginning for virtual environments." *IEEE Computer Graphics & Applications*, pp. 12–15, Sep./Oct. 2000.
4. C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti. "Surround-screen projection-based virtual reality: The design and implementation of the CAVE." *Proceedings of SIGGRAPH 93*, pp. 135–142, Aug. 1993.
5. E. Frécon and M. Stenius. "DIVE: A scalable network architecture for distributed virtual environments." *Distributed Systems Engineering Journal*, 5, pp. 91–100, 1998.
6. C. Greenhalgh, J. Purbrick, and D. Snowdown. "Inside MASSIVE-3: Flexible support for data consistency and world structuring." In *Proceedings of Collaborative Virtual Environments 2000*, pp. 119–127, San Francisco, September 2000.
7. J. P. Grossman and W. J. Dally. "Point sample rendering." In *Proceedings 9th Eurographics Workshop on Rendering*, Rendering Techniques, pp. 181–192, 1998.
8. R.Hubbold, J.Cook, M.Keates, S.Gibson, T.Howard, A.Murta, A.West and S.Pettifer. "GNU/MAVERIK: A micro-kernel for large-scale virtual environments." In *Proceedings of ACM Symposium on Virtual Reality Software and Technology 1999 (VRST'99)*, London, pp. 66–-73, Dec. 1999.
9. E. Lantz. "The future of virtual reality: head mounted displays versus spatially immersive displays (panel)." In *Proceedings of SIGGRAPH 96*, pp. 485–486, Aug. 1996.
10. J. Leigh, A. E. Johnson, and T. A. DeFanti. "CAVERN: A distributed architecture for supporting scalable persistence and interoperability in collaborative virtual environments." *Journal of Virtual Reality Research, Development and Applications*, 2(2), pp. 217–237, Dec. 1997.
11. D. Lischinski and A. Rappoport. "Image-based rendering for non-diffuse synthetic scenes." In *Proceedings of the 9th Eurographics Workshop on Rendering 98*, Rendering Techniques, pp. 301–314, 1998.
12. L. McMillan and G. Bishop. "Plenoptic modeling: An image-based rendering system." In *SIGGRAPH 95 Conference Proceedings*, ACM SIGGRAPH Annual Conference Series, pp. 39–46, 1995.
13. M. M. Oliveira, G. Bishop, and D. McAllister. "Relief texture mapping." In *SIGGRAPH 2000 Conference Proceedings*, ACM SIGGRAPH Annual Conference Series, pp. 359–368, 2000.
14. K. Park., Y. Cho, N. Krishnaprasad, C. Scharver, M. Lewis, J. Leigh, and A. Johnson. "CAVERNsoft G2: A toolkit for high performance tele-immersive collaboration." *Proceedings of the ACM Symposium on Virtual Reality Software and Technology 2000,* Seoul, Korea, pp. 8–15, 2000.
15. S. Pettifer, J. Cook, J. Marsh and A. West. "DEVA3: Architecture for a large scale virtual reality system." In *Proceedings of ACM Symposium on Virtual Reality Software and Technology 2000 (VRST'00)*, Seoul, October 2000.
16. H. Pfister, M. Zwicker, J. van Baar, and M. Gross. "Surfels: Surface elements as rendering primitives." In *SIGGRAPH 2000 Conference Proceedings*, ACM Siggraph Annual Conference Series, pp. 335–342, 2000.
17. R. Raskar, G. Welch, M. Cutts, A. Lake, L. Stesin, and H. Fuchs. "The office of the future: A unified approach to image-based modeling and spatially immersive displays." *Proceedings of SIGGRAPH '98*, pp. 179–188, July 1998.
18. S. Rusinkiewicz and M. Levoy. "QSplat: A Multiresolution point rendering system for large meshes." In *SIGGRAPH 2000 Conference Proceedings*, ACM Siggraph Annual Conference Series, pp. 343–352, 2000.
19. H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. "RTP: A transport protocol for real-time applications." *RFC 1889*, January 1996.
20. C. Shaw, M. Green, J. Liang, and Y. Sun. "Decoupled simulation in virtual reality with the MR Toolkit",. *ACM Transactions on Information Systems,* 11(3), pp. 287–317, July 1993.
21. S. P. Smith and D. J. Duke. "Binding virtual environments to toolkit capabilities." *Proceedings of EUROGRAPHICS 2000,* pp. C-81–C-89, 2000.
22. O. G. Staadt, A. Kunz, M. Meier, M. H. Gross. "The blue-c: Integrating real humans into a networked immersive environment." *Proceedings of ACM Collaborative Virtual Environments 2000,* pp. 202–202, 2000.
23. "Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and rules." *IEEE Standard 1516,* Sep. 2000.
24. H. Tramberend. "Avocado: A distributed virtual reality framework." *Proceedings of the IEEE Virtual Reality 1999,* pp. 14–21, 1999.