Abstract. In this paper we present the QAVE (Quake Automatic Virtual Environment) system that we developed based on the computer game Quake III Arena. The QAVE system is an immersive CAVE-like virtual reality system. The system uses the open source Quake 3 game engine that was adapted to run and display the virtual world, and the open source computer vision library OpenCV to perform optical tracking of the QAVE users, which makes the system very cost-effective. The use of a computer game as the basis for the QAVE system means that it is able to take advantage of the content and tools created for the game and provide rich and dynamic worlds for the system’s users. The QAVE system demonstrates that computer games are a viable platform for scientific research, especially in the development of virtual reality applications.

Keywords. QAVE, virtual reality, CAVE, Quake, computer game

1. Introduction

In recent years games have been increasingly used for scientific purposes because a great deal of effort had gone in their development, and by nature they are flexible so that with some small changes they can be easily put to other uses. One of the most obvious applications is the field of virtual reality since games are by their nature virtual environments.

Virtual Reality (VR) is a field that has long interested scientists, engineers, artists and the general public. Virtual reality has many applications; it is used in engineering to help visualize data, simulate complex mechanisms, and so on. It is also used in entertainment, medicine (imaging, surgery), training (pilots, astronauts), psychological therapy (phobias, post traumatic stress), etc.

The main goal of virtual reality is to immerse a human into a virtual world. Some of the most immersive systems developed to date are the CAVE-like systems, named after the first such system developed at the University of Illinois at Chicago [1]. The CAVE (CAVE Automatic Virtual Environment) is basically a small room where the walls, the floor and the ceiling are computer displays. Each of these displays shows the view of the virtual environment that can be seen from the position of the user (inside the virtual environment) in the direction of the wall, if the user’s position changes the displays have to change their view so it corresponds to the new perspective. If the displays are properly aligned, and if the perspective of the image shown is properly adjusted as the user moves around, the immersion will be complete, some CAVE-like systems are so convincing that people actually bump into the walls.

An interesting example of a CAVE-like system is CaveUT, developed by Jeffrey Jacobson and his collaborators. CaveUT is based on Epic Megagames' computer game Unreal Tournament and uses Unreal Tournament’s built-in support for game modifications to create a CAVE-like system. In its latest version, CaveUT 2004 v2.0 has reached a high level of immersion with support for active stereoscopic graphics and motion tracking, confirming the great potential computer games have in the development of VR systems [2], [3].

This paper presents the QAVE system, a game-based CAVE-like system similar in concept to the CaveUT system. QAVE offers an immersive virtual environment based on id Software’s Quake III Arena computer game, a direct competitor to Unreal Tournament. At the time of its release in 1999 it represented the very apex of real time computer graphics on PCs, in August of 2005 its source code was released under the GPL license and so became available to everyone and enabled many applications including our QAVE system. There is another system that combines CAVEs and Quake, Paul Rajlich’s CaveQuake [4]. However the two systems are diametrically opposed: CaveQuake enables a regular CAVE system to play the Quake game; on the other hand QAVE uses the Quake game to create a CAVE system.

2. Games in research

Games are of interest in research because they can provide inexpensive, fast, well rendered,
interactive virtual worlds and modern games even have realistic physics models. They also often have free development tools that can be used to create game content (player models, game maps, etc.) and modify the game. Popular games have large fan communities that create large amounts of content and various game modifications (“mods”) sometimes completely changing the game. Given that the gaming industry is a multi-billion dollar business games will only get even more advanced, therefore even better platforms for research. [5], [6]

The games most often used in research today are either part of the Quake series or Unreal series of games. These games belong to the First Person Shooter (FPS) genre. A FPS is a type of action game with a 3D world viewed from the player's perspective, the player looks at the world through the eyes of the character s/he's controlling. Although the first FPS games only had the first person view (hence the name), modern FPS games have no such restriction.

Modern FPS games are based on the client-server architecture. Although it’s actually one program, the game is logically split into two parts. The server runs the simulation, while the client provides a user interface. One of the main reasons for this separation is support for multiple players.

The multiplayer mode of FPS games is the key to creating a CAVE-like system. The Quake and Unreal games offer a spectator mode in which players in a multiplayer game can just observe the action without participating in it. The spectator mode can be used in free floating mode where the players can fly the camera through the world, or in follow mode in which the spectator follows one of the players in the game and sees the world from his/her perspective. This mode is what is used when creating the multi-screen display of a CAVE-like system.

In a QAVE (or CaveUT) system there is one computer assigned to each screen of the display, there is also a computer assigned to the user. Each computer runs an instance of the game, one computer (usually the user's) starts a multiplayer game and acts as the server and the other computers connect as clients. The game instance controlled by the user represents the user's avatar; the clients all switch to spectator mode and select the avatar as the player to follow. So far all this can be accomplished with no changes to the game and at this point all the screens would be showing the same view. However by making a small modification to the game the views can be rotated so that, for example, in a three wall CAVE, one view is looking forward, one is rotated 90 degrees left, an the last one is rotated 90 degrees to the right. Each client is providing one part of the composite view. Since the rotations are relative to the player's view and the player is followed around, the user can move around the world freely and the display will stay a continuous view of the world from the perspective of the avatar.

To be fully immersive a CAVE-like system must maintain the proper perspective for all of the different parts of the display. Simply rotating the view will only provide proper perspective if the user is in the very middle of the display. To allow for correct perspective from other positions CAVE-like systems have to provide perspective correction. The position of the player has to be known or determined and the view frustum (the portion of the world that is rendered) has to be adjusted accordingly. This basically turns a static screen into an active window into the world. In CaveUT adjusting the frustum is done by using a library (VRGL) that acts as an intermediary between the game and the OpenGL library, in QAVE the game itself has been modified to allow perspective correction.

QAVE and CaveUT share many of the same traits, most notably the low cost; however our system (QAVE) has one major advantage over CaveUT. Unreal Tournament is only partially open source and although a lot of changes can be made by using UT's mod system there is a limit. On the other hand Quake III Arena is completely open source so we have absolute freedom to modify the code and adjust it to our needs; however this advantage is also its main drawback. CaveUT is built as a regular modification of UT and therefore it can be (and has been) upgraded to new versions; although Quake also supports mods, the QAVE system takes advantage of the fact that we now have access to the full code and has changed parts of the game that were not initially modifiable, because of that it can't be easily updated.

This is the biggest difference between CaveUT and QAVE; CaveUT can always stay up-to-date with the latest advances in game technology but has less control, whereas QAVE has full control but doesn’t benefit from the advances in the gaming industry. Ultimately, the QAVE system has to have its own programmers keep up with the latest technologies; fortunately there are a lot of enthusiasts in the world improving on the Quake 3 engine and as long as
the QAVE system stays compatible it should benefit from their work.

3. QAVE overview

QAVE (Quake Automatic Virtual Environment) is a CAVE-like system that uses a modified version of id Software's Quake III Arena computer game as its graphics and physics engine. The QAVE system is comprised of three programs: QAVE Control Center, QAVE Wall Proxy and QAVE Arena.

Figure 2. Construction used during development

As the name implies, the QAVE Control Center (QCC) controls the functioning of the QAVE, more precisely it accepts and interprets gestures from the user's dataglove and also performs motion tracking. Hand gestures in tandem with the relative position of the hand in relation to the head of the user are used to control movement inside the virtual environment while the position of the head is also used for perspective correction.

The QAVE Wall Proxy (QWP) performs the calculations required for perspective correction. There is one QWP running for each screen (usually one per wall) of the display. The QWP is configured with the position and orientation of the screen that it's assigned. It receives messages from the QCC via a network connection. These messages contain the position of the QAVE user. QWP then calculates the perspective correction and passes the results to the QAVE Arena doing the rendering for that screen, the QA then adjusts the rendering of the image for its screen.

QAVE Arena (QA), a modified version of Quake III Arena, is the heart of the QAVE system. One instance of QAVE Arena is the user’s and deals with the actual actions within the virtual environment (VE); other instances belong to the QAVE screens and are in charge of the rendering. QA is controlled by messages received from the QAVE Control Center and the QAVE Wall Proxy.

Figure 2 shows the construction we used for testing purposes during QAVE’s development; it is build out of PVC pipes. The physical and logical structure of the QAVE system is shown in Figure 3.

3.1. QAVE Control Center

QCC is the only part of the QAVE system that is directly controlled by the operator and the user. The QAVE operator is in charge of making sure that QCC is properly tracking the user. To insure this, the operator has to first configure QCC with the position of the camera used by the tracking system and also the user’s height. Later when the system is running, the operator selects the user’s head and hand in the tracking windows and adjusts the tracking parameters to insure the system keeps track of the user.

The user interacts with QCC (and therefore the QAVE system) by means of a dataglove and the tracking system. Currently, QCC only supports 5DT datagloves but it can easily be reprogrammed to use other datagloves or other types of input devices. The QCC’s tracking system uses the OpenCV computer vision library [7], specifically it uses the CamShift algorithm implementation provided by OpenCV. The algorithm performs tracking based on color and uses the HSV color model. To track objects QCC uses a simple C++ class created for the QAVE system that takes care of all tracking and provides an easy to use interface to the operator.

3.2. QAVE Wall Proxy

The QWP acts as an intermediary between QCC and QA and is the program that performs the calculations necessary for perspective correction. It was created to minimize the changes done to the code of the game itself. There is an instance of QWP running for every screen and comes paired with that screen’s QA, both programs run on the same computer. Since each instance of QWP is assigned its own screen it has to be configured with the screen’s position, QWP uses this information with the user’s head position it got from QCC to calculate the necessary corrections. The corrections are field-of-view angles for four directions (up, down, left, and right) and are sent to the screen’s QA.
3.3. QAVE Arena

QAVE Arena is the heart of the QAVE system; it is what makes everything possible. It runs the simulation, renders the virtual environment and even allows connections between multiple QAVEs. In the QAVE system there are a number of instances of QA that run on different computers. One of these instances is connected directly to the QCC and serves as an avatar for the user of the QAVE; all actions by the user are performed by the player in this instance of QA. Other instances of QA are there to provide the various different views that are projected onto the screens.

An instance of QA acts as the server that actually creates the virtual environment; the other instances connect as clients. Usually the user’s QA will be the server and the display QAs will be the clients but this is not necessary. Any instance of QA can be a server even those that are not part of the QAVE system itself, this means that somebody inside a QAVE system can connect to any QA anywhere in the world. Furthermore this allows several different QAVE systems to be connected to each other so that
different people from across the world can be in the same virtual environment.

Quake III Arena, like most FPS games, has what are known as console variables. The console is used by players to change the game's settings and status while it is running; these changes are done in part by issuing commands and in part by setting variables. QAVE Arena has defined several console variables in addition to those it inherited from Quake III Arena. Some of these variables are used to configure QA with the orientation of the screen (the rotation from the forward direction); others are used to adjust the perspective correction, to rotate the whole display when necessary and so on. The configuration variables are set by the operator manually but the perspective correction and display rotation variables are set by QA automatically when it gets a message from QWP specifying the new values.

4. QAVE Messages

The QAVE system is made up of multiple components which need to work together, this is accomplished by passing messages. The different QAVE Arena instances communicate with each other to maintain the virtual world like any FPS and use internal systems inherited from Quake III Arena; they also accept messages sent by QCC and QWP that are unique to the QAVE system. The QAVE messages are either windows messages that are used to send information from QCC or QWP to the QA they control (and therefore running on the same computer), or they are network broadcast messages (UDP datagrams) that are sent from QCC to QWP. Although QCC and an instance of QWP can be running on the same computer they still communicate via UDP. The flow of messages is one way, QCC only sends messages, QWP sends and receives but only in one direction: from QCC and to QA.

QCC sends QA messages regarding the user's interaction with the environment, specifically it sends movement commands. There are three types of messages sent by QCC to the QWPs: the first specifies the user's position inside the QAVE display, the second specifies view rotation (which is passed on to QA) and the last shuts down the QWPs. QWP sends QA messages specifying the perspective correction and view rotation.

5. Tracking system

The QAVE's tracking system uses the open source OpenCV library originally developed by Intel [7]. The library implements several computer vision algorithms including many tracking algorithms; the one used by the QAVE is the CamShift (Continuously Adaptive Mean Shift) algorithm [8]. CamShift is a modified version of the Mean Shift algorithm which is a technique of finding the peak of a probability distribution. CamShift works by using the hue value of a frame as the probability distribution. In the QAVE system the hue to track is determined by the histogram of the object being tracked. The histogram is specified by the QAVE operator selecting the object on the image captured by the camera. The object location provided by the CamShift algorithm tends to jitter, to provide a stable image for the QAVE these jitters have to be smoothed out.

After the head and hand have been found on the camera frame image it is necessary to calculate their real world positions. QCC is configured with the position and orientation of the camera (assumed to be facing straight down), its field of view and the user's height. The system knows the location of the head and hand in the image coordinates. To determine the real world coordinates a simple geometric calculation is performed.

The system will only give accurate positions for objects in the plane defined by the user's height. If the user were to duck the system would no longer be able to track his/her position accurately. This is a known restriction of the system and can be removed in the future by using two non-parallel cameras instead of just one.

Another related problem is that this system can only accurately track the user's hand if it is in the same plane as his/her head which would be awkward. This is actually not that big of an issue since the exact position of the hand is not currently used by the QAVE system. The system does not need to know the position of the hand in relation to the rest of the world, it only needs to know the angle in the horizontal plane (the yaw) formed by a line from the head to the hand. Since the camera is facing down on the user this angle can be determined without knowing the vertical position of the hand, there will be a small error if the user is not directly below the camera but this is not a critical problem.
6. Using the QAVE

The QAVE has been designed with a very simple user interface; there are only two basic actions the user needs to perform to interact with the system: pointing and dragging. When moving around inside the QAVE the perspective will be automatically corrected but the user will not move inside the virtual environment. If the user wants to move somewhere s/he just has to point in the desired direction and gesture whether s/he wants to go forward or reverse (by using the little finger), fast or slow (using the thumb). Figure 4 shows the gesture used to move in reverse. With displays that don’t have a full 360 degree view it is sometimes necessary to rotate the view, to do this the user simply “grabs” the view, drags it around in the direction it needs to be rotated, and then lets it go.

![Figure 4. Glove gesture for reverse movement](image)

7. Conclusion and future work

In this paper we presented the QAVE system, a CAVE-like system based on the Quake 3 computer game engine. We have shown how a game can be adapted to support the creation of an immersive multi-screen display for a CAVE-like system. The Quake 3 engine has also been altered to support active perspective correction and enable control with an external program. QAVE is comprised of three programs running on several computers and in multiple instances, the central program coordinates the functioning of the system by using a simple unidirectional message passing protocol. The system utilizes head tracking to facilitate perspective correction, and head and hand tracking combined with dataglove gesture recognition to create an easy-to-use user interface. QAVE offers a complex interactive virtual environment, and although the system is not perfect in this early stage of development, it does provide a viable platform for the development of immersive virtual reality applications. The QAVE system is an open and scalable system, and with additional development it can be improved and provide a high level VR experience.

There are three aspects that should be improved upon. The first is better tracking; the current system cannot track changes in elevation (height). This could be easily accomplished by adding another camera to the optical tracking system. The second is full stereoscopic support which would require some more game modification but also a way of tracking the user’s head orientation. This can be accomplished by using an inertial orientation tracker or by further improving the optical tracking system. The former is relatively simple; the latter would require a considerable amount of work. Finally, the third is to move the avatar in the virtual environment as the user moves inside the QAVE, although this would not be very useful as a means of controlling movement, it would provide parallax depth cues and so improve the user’s immersion.

8. References