Gesture-Based, Touch-Free Multi-User Gaming on Wall-Sized, High-Resolution Tiled Displays

Daniel Stødle, Tor-Magne Stien Hagen, John Markus Bjørndalen and Otto J. Anshus
Dept. of Computer Science
University of Tromsø
9037 Tromsø, Norway
{daniels, tormsh, jmb, otto}@cs.uit.no

ABSTRACT

We investigate the responsiveness of a touch-free multi-user human-computer interface we have developed for a wall-sized, high-resolution 28-tiled display. We adapted Quake 3 Arena (Q3A) and Homeworld to use the interface, and parallelized them to run with high framerates on the display wall. The interface comprises 16 cameras and 9 computers, using triangulation to detect object positions. The games use the positions to identify hand and arm movements, mapping them to keyboard and mouse events. The display wall can be used by one or several simultaneous players.

To parallelize Q3A, we exploit its existing client-server architecture and spectator-functionality. Each tile runs a spectator, which displays its corresponding part of a given player's view. To parallelize Homeworld, we run a copy on each tile, and synchronize the rendering of each new frame. Each copy receives the same input and shares a global clock.

The intrinsic latency of the touch-free interface averages 140 ms, with about 100 ms from the cameras. We expect improvements in camera technology to reduce this. The time to detect the simple gestures is insignificant. The framerate for parallel Q3A was 398, vs. 21 when using Chromium to distribute Q3A's graphics output. Homeworld did not run using Chromium, but the parallel version achieved a framerate that was always above 80. The framerates achieved by the parallel games were well above the refresh rate of the projectors used. Informal use of the touch-free interface indicates that it works better for controlling Q3A than Homeworld.

INTRODUCTION

We have developed a touch-free multi-user human-computer interface for wall-sized, high-resolution tiled displays. Users standing in front of such a display wall can interact with applications using hand- and arm-gestures. Games are a class of applications that require low-latency input. To investigate the responsiveness of the touch-free interface, we applied it to two commercial, but open-source games[1]. The two games were Quake 3 Arena (Q3A) [9] and Homeworld [6], respectively a first-person shooter (FPS) and a real-time strategy (RTS) game. Figure 1 shows two persons playing Q3A against each other on a display wall. The person in the middle is playing Homeworld.

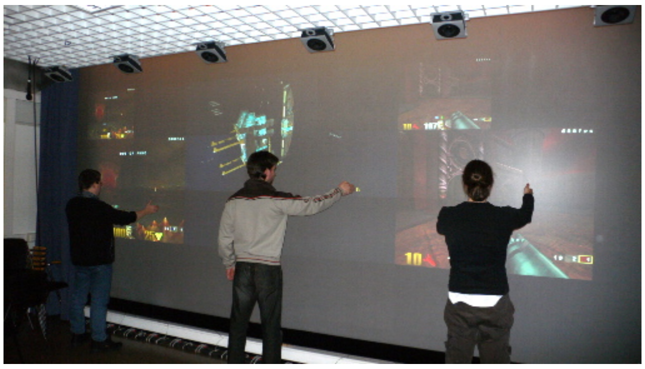


Figure 1. Two persons playing Q3A and one person playing Homeworld simultaneously on a 7x4 tile display wall. Q3A runs on 2x2 tiles to the left and right, and Homeworld on 3x3 tiles in the middle.

For games, high framerates are important [4]. Maintaining high framerates becomes increasingly difficult as the resolution goes up. The resolution of a typical desktop display is about 2-3 megapixels, while the resolution of a display wall ranges from 10 to 100 megapixels [11, 15]. Display walls are usually built by arranging tiles with a resolution of 1-2 megapixels in a grid. One way to run games on display walls is to use Chromium [8] to distribute the game's graphics output to the tiles. We measured the framerate when using Q3A with Chromium, and found that when scaling up from 2x2 tiles to 7x4 tiles, the average framerate decreased from 73 to 21. We were not able to run Homeworld using Chromium.

To solve this, we parallelized both Q3A and Homeworld. Q3A required parallelization to maintain high framerates beyond 2x2 tiles, and Homeworld required parallelization to run on more than one tile. The result was that the games

[1]Only the game engines are open source. The data files still require a license.

could run at high framerates on a tiled display wall comprised of 28 projectors, each driven by one computer and arranged in a 7x4 grid, for a total resolution of 7168x3072 pixels.

The touch-free interface uses 16 cameras and 9 computers to detect objects in front of the display wall, and is able to detect multiple objects simultaneously at a rate of 30 Hz. When three or more cameras see the same object, triangulation can be used to determine the object's position. We refer to it as touch-free, as users can interact with the display wall without actually touching its canvas. This is an important advantage over existing solutions that require touch to work [7], as our canvas is flexible and thus prone to perturbation when users touch it.

The games were modified to accept position data from the touch-free interface, and convert them into simple hand- and arm-gestures. When a gesture is recognized, a corresponding keyboard or mouse event is injected into the game's input event stream and handled normally by the game.

We used two different approaches to parallelize the games. For Q3A, we run a client on each tile of the display wall. We then exploit Q3A's client-server based architecture and the existing concept of a spectator. The server keeps the clients in sync, and the spectator-concept enables different clients to be configured so as to constantly follow a given player as that player moves around. We also modify each spectator's view frustum in accordance with the tile it runs on to create a coherent, multi-tile view for a given player.

Homeworld was parallelized by running a copy on each tile, and ensuring that each copy gets the same input. All the copies share a global clock and random number generator seed. The goal is to make each copy compute the same game state for each frame.

We conducted experiments to measure the latency of the touch-free interface, as well as the framerate of the two games. The experiments show that the time before an object's position is available to the games averages 140 ms, with the majority of this latency incurred by the FireWire-based cameras. Game-side gesture-processing did not incur significant latencies, due to the simple gestures involved. For Q3A, we measure the framerate to be as much as an order of magnitude better than using Chromium [8]. Homeworld's framerate remains high, and outperforms the single-display configuration when running on both 2x2 and 3x3 tiles.

Our main contributions with this paper are (i) a distributed, touch-free multi-user interface, (ii) a prototype system for gesture-based input to games in the FPS and RTS genres, (iii) an evaluation of the interface's responsiveness when used to interact with two games, and (iv) evaluation of three different approaches for making existing games run on display walls.

RELATED WORK

The Quake-series of games have been popular targets for modification and extension, both in terms of input devices and display surfaces. Some examples include playing Quake using Nintendo's Wiimote, using eye-tracking to play Quake, or controlling Quake from a PDA[2]. CaveQuake is a limited re-implementation of Quake II and Q3A for use in a CAVE[3], but does not support all the features of the full games, and for the Q3A case does not even support playing. We are not aware of any work to integrate new input devices or new display surfaces for Homeworld.

In [2], a gaming interface based on a commercially available stool, "The Swopper," is presented. The stool and a light gun is used to produce joystick input events to control an FPS game. By shifting the body weight and rotating on the stool in combination with aiming and firing the gun, the user can navigate and interact with the world. Our approach does not use any external devices, but instead uses simple hand and arm gestures to interact with the games. The large display wall also allows our system to support multiple players playing at once. The stool-and-light-gun approach is more expressive compared to our simple gestures, however.

Gesture VR [14] is a video-based, hand-gesture recognition system. The system recognizes three gestures which are used to provide applications with different input events, as demonstrated by controlling Doom, an FPS game developed by id Software. Their solution is centralized, using two synchronized cameras connected to a single computer. We use 16 cameras connected to 8 computers, enabling us to cover a larger area at the cost of a more complicated implementation. Our system can only recognize simple gestures (2D position and radius of detected objects), while their system allows for detection of 3D position and three different gestures.

In [16], the authors argue that a digital table is a conductive form factor for general co-located home gaming. By combining speech and hand gestures as input to two commercial games, The Sims and Warcraft III, several persons can interact with the games running on the tabletop. Our solution is based on hand- and arm-gestures alone on wall-sized displays. The physical dimensions of the display wall enables more than a couple of people to play at once, against each other or co-operatively. Further, we have modified the source of the two games, enabling more flexible multi-point interaction. The games they have used are not open source, requiring that they build custom wrappers that translate touch- and speech input to mouse and keyboard events.

The authors of [16] use the Diamondtouch [5] tabletop for multi-touch interaction. Other technologies for multi-touch interaction include [7], where infrared light is projected into a canvas and internally reflected. The internal reflection of the light is frustrated at points where the user touches the canvas. The escaping light can be detected using a camera mounted behind the canvas. Our system is based on detecting the presence of objects directly, and does not require the

[2]http://www.youtube.com/watch?v=n1tsXc2RoeM
http://www.youtube.com/watch?v=3pRWYE2LRhk
http://www.youtube.com/watch?v=tNJXjNBgmLs
[3]http://www.visbox.com/cq3a/

user to actually touch the display wall's canvas. In [13], the author presents a camera-based solution to detecting and positioning objects in front of a whiteboard. The approach is similar to ours, except that we take a distributed approach with 16 commodity FireWire cameras connected to a set of computers, whereas they use custom cameras with on-chip processing to perform object recognition.

Chromium [8] is a system for distributing streams of rendering commands, allowing many existing OpenGL applications to run on tiled display walls without modifications. By making applications use Chromium's OpenGL library, Chromium can intercept rendering commands and forward them to remote rendering nodes. We were not able to run Homeworld using Chromium, and Chromium's rendering performance running Q3A did not scale well beyond 2x2 tiles.

In CaveUT [10], a set of modifications to Unreal Tournament is presented that allows it to display in panoramic theaters. They apply the same principle of using spectators to support multi-tile rendering as we have done when modifying Q3A, but do not present any measurements regarding the performance of their approach. We measure the framerate and document the latency incurred by using spectators in Q3A to generate the tiled view.

DESIGN

Quake 3 Arena [9], developed by id Software, is an open-source first-person shooter designed for multiplayer gaming. It is based on a client-server architecture where the server maintains the state of the game. At a fixed rate, independent of the connected clients, the server updates its game state, before broadcasting state changes to connected clients. Clients use this to update their view of the game. A client in Q3A is either a player or a spectator. A player is a client that participates in the game. A spectator is a client that instead of participating, follows one of the players around and displays that player's view of the game.

Homeworld [6] is a popular 3D real-time strategy game developed by Relic Entertainment. In September 2003, the Homeworld engine was made open source. Although the Linux version still lacks some of the features of the complete game, including software rendering, cut-scene playback and networked multiplayer support, the game itself is fully playable in single-player mode. In contrast to Q3A, Homeworld has a monolithic design, with all code running inside a single process.

Figure 2 shows the overall design of the touch-free interface, and its use with Q3A and Homeworld. Images are captured and then analyzed to locate objects in a plane parallel to the the display wall's canvas. The positions of these objects are then processed by an object detector that yields the object's 2D position and radius, before the resulting information is sent to the two games. The two games process the data individually, using object positions and radii to detect gestures and handle them in game-specific ways.

The design of the parallelized Q3A uses a modified player

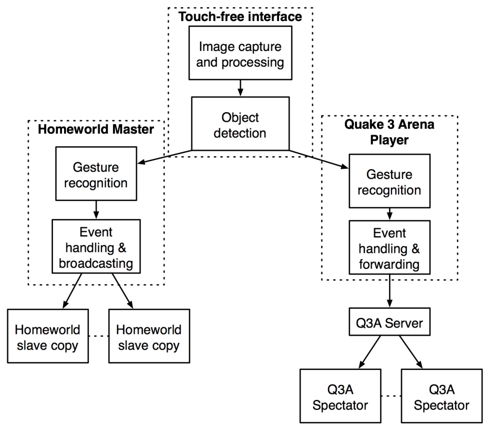
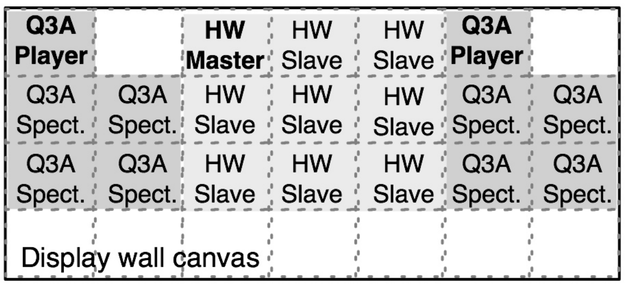


Figure 2. The design of the touch-free interface, and its use with Q3A and Homeworld.

that receives input from the touch-free interface. The player then recognizes gestures and converts them to keyboard and mouse events suitable for the game. The player relays its actions to the Q3A server, which then updates all clients with the new game state. This causes the spectators following a given player to update their view. For Homeworld, a single copy is elected as a master. The master becomes responsible for accepting and interpreting input from the touch-free interface. After recognizing gestures, the input is handled and broadcast to the slave copies. Figure 3 shows one configuration of a 7x4 tiled display wall where two users can play Q3A against each other, while a third user simultaneously plays Homeworld. This configuration is identical to the one pictured in Figure 1. Several other configurations are also possible.



Q3A Player		**HW Master**	HW Slave	HW Slave	**Q3A Player**	
Q3A Spect.	Q3A Spect.	HW Slave	HW Slave	HW Slave	Q3A Spect.	Q3A Spect.
Q3A Spect.	Q3A Spect.	HW Slave	HW Slave	HW Slave	Q3A Spect.	Q3A Spect.
Display wall canvas						

Figure 3. Running Q3A and Homeworld on a 7x4 display wall. To the left and right, two Q3A players control a set of Q3A spectators. In the middle, a single Homeworld master synchronizes the rendering and game simulations of 8 Homeworld slave copies.

Hand- and arm-gestures

When playing an FPS using a mouse and keyboard, the mouse is used to aim and fire, and the keyboard is used for movement. In addition, the mouse's scroll wheel is often used to switch weapons, and the keyboard to control other actions the player can take (ducking, jumping, etc.). We used the following gestures, summarized in Table 1, for controlling Q3A. When only one hand is detected by the input system, its position is used for controlling the player's aim. When the hand is tilted (making it flat), it will additionally fire the player's weapon. When two hands are detected, the right hand controls aim and firing, and the left hand is used to move the player forwards or backwards. Figure 4 illustrates the gestures.

Action	Gesture
Aim	Move right/only hand
Fire weapon	Flat right/only hand
Move forward	Vertical left hand
Move backward	Flat left hand

Table 1. The gestures in Q3A that the game recognizes and maps to actions.

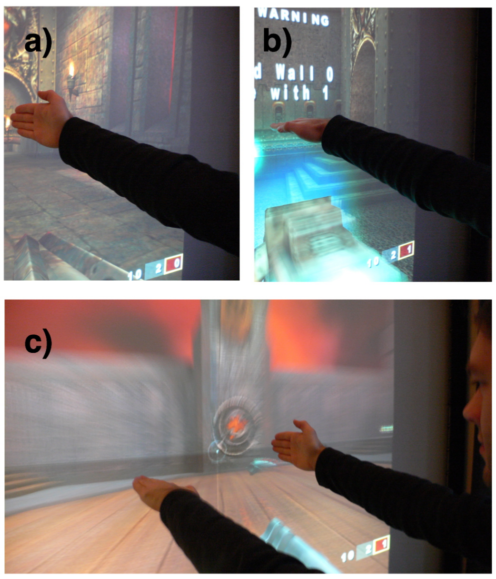


Figure 4. Gestures for controlling Q3A. (a) Using a "vertical" hand to control the player's aim. (b) Controlling aim and firing by making the hand flat. (c) Moving and aiming simultaneously using both hands.

Homeworld uses a different control scheme. When using a keyboard and mouse, the main controls can all be accessed with the mouse, and the keyboard is mostly used for shortcuts for different menu selections and buttons. When no mouse buttons are pressed, the mouse simply controls an on-screen cursor. Holding down different mouse buttons, the user can pan and zoom the camera, as well as select entities and manipulate them from a contextual menu. Table 2 lists the different actions in Homeworld, and our mapping to gestures. The cursor is controlled using a one-to-one mapping from hand location to screen. When the right/only hand is flat (like the fire-gesture in Q3A), the user can select or click items. The user can enter or leave Homeworld's tactical view using a vertical left hand. With a flat left hand, the user can either invoke Homeworld's contextual menu (for moving ships, creating formations, and so on), or panning the camera (by simultaneously moving the right hand). Finally, the user can zoom the camera in and out using a flat left and right hand, varying the distance between them to control the amount of zoom.

Action	Gesture
Control cursor position	Move right/only hand
Select/click entities	Flat right/only hand
Pan view/contextual menu	Flat left hand
Toggle tactical view	Vertical left hand
Zoom	Flat left and right hand, distance between hands control zoom factor

Table 2. Actions in Homeworld and their corresponding gestures.

IMPLEMENTATION

Figure 5 shows the architecture of the touch-free multi-user interface. The interface makes use of 16 FireWire cameras, connected in pairs to 8 Mac minis. The cameras are mounted along the floor, enabling the detection of objects in a plane parallel to the display wall's canvas. The cameras capture images at 30 FPS. Each image is processed by subtracting the background, removing noise and thresholding the result to identify objects (which are typically hands or arms). This yields zero or more pairs of 1D position and radius. Each Mac mini sends its identified positions and radii via an event server to a MacBook Pro that determines the position of each object in 2D space using triangulation (Figure 6). The resulting 2D positions and radii are sent via the event server to either Homeworld or Q3A. The event server's role is to distribute events of different kinds to software used with the display wall.

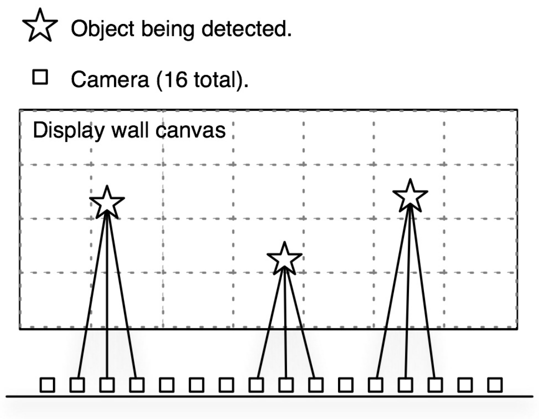


Figure 6. 16 cameras positioned below the display wall's canvas is used to triangulate the position of different objects. Each camera has a 42 degree field-of-view, capturing images at 30 FPS with a resolution of 640x480 pixels in 8-bit grayscale.

We modified Q3A and Homeworld to receive object position events from the touch-free interface, and then interpret them according to the gestures outlined in the previous section. When a gesture is recognized, events corresponding to the action associated with the gesture is injected into the game's input event stream. Depending on the relative amount of movement detected, mouse events can be generated, and the

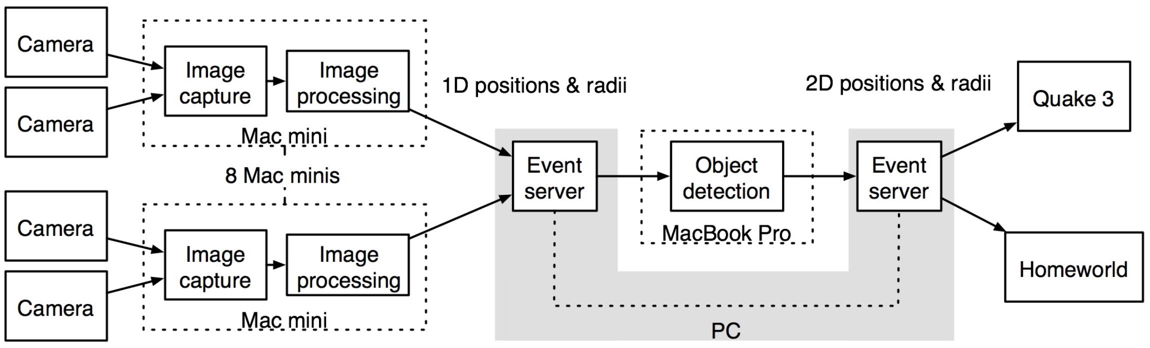


Figure 5. The architecture of the touch-free interface.

object's radius is used to determine whether it is interpreted as a flat hand or a vertical hand.

The software for capturing images, detecting and positioning objects was implemented for Mac OS X in Objective-C and C, using libdc1394[4] to communicate with the FireWire cameras.

Quake 3 Arena and Homeworld on Tiled Display Walls

Running Q3A and Homeworld on a tiled display wall requires that each tile displays a part of the total view for each game. To achieve this, the view frustum used by OpenGL for both Q3A and Homeworld must be modified in relation to the tile on which the game runs.

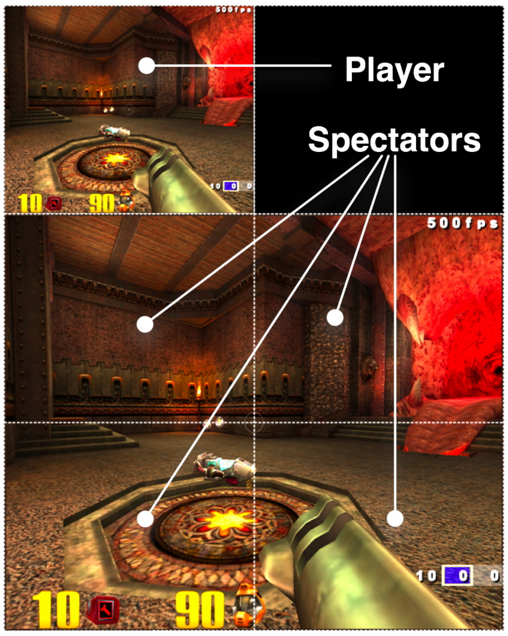


Figure 7. Example Q3A configuration on a display wall. The upper left corner shows the player, while the remaining four clients are spectators following that player, with modified view frustums to match the tiles on which they run.

[4]http://libdc1394.sourceforge.net/

For Q3A, we control the parallel version by configuring a set of environment variables, and then reading them from within the game. The variables control how the view frustum is configured, as well as whether or not a client is designated as a player or a spectator, and which player a given spectator follows. Due to the client-server architecture of Q3A, this is sufficient to create a parallel version that will run on the display wall. Figure 7 shows a player in the upper-left corner, with four spectators following that player, as it would appear on a tiled display wall.

To parallelize Homeworld, we could not use the same approach as for Q3A, as Homeworld does not support the concept of a central server to coordinate a set of clients. Instead, we run several tightly coupled copies and manually ensure state consistency between them. Each copy runs on one tile, and the Message Passing Interface (MPI) [3] is used to exchange state information and keep the copies synchronized. One copy is elected as master, and the remaining copies become slaves. For each frame, the master accepts input from the touch-free interface and broadcasts it to the slaves. Before starting a new frame, all the copies synchronize at a barrier. This ensures that each slave receives the same input during the same simulation step in the game, and makes the visual display synchronized. To ensure that each copy's game simulation runs identically on all nodes, the same value is used to seed each copy's pseudo-random number generator. Finally, all copies share a global clock controlled by the master. The drawback to a state-synchronizing solution like this is that it requires great familiarity with the source code in order to guarantee that all the game simulations end up running identically.

EXPERIMENTS

We have conducted three experiments to evaluate our work. The first experiment's goal was to determine the latency involved in using the touch-free interface, to evaluate if it is low enough to be usable for controlling games. For the next two experiments, the goal was to measure the rendering performance of Q3A and Homeworld, when using either Chromium or our parallel versions.

The hardware used was (i) a display cluster with 28 nodes (Intel Pentium 4 EM64T, 3.2 GHz, 2 GB RAM, HyperThreading enabled, NVIDIA Quadro FX 3400 with 256 MB Video

RAM, running the Rocks cluster distribution 4.0) connected to 28 projectors (1024x768, arranged in a 7x4 matrix), (ii) switched, gigabit Ethernet, (iii) 8 Mac minis (1.66 GHz Intel Core Duo, 512 MB RAM, Mac OS X 10.4.8), (iv) 16 Unibrain Fire-i FireWire cameras, (v) a MacBook Pro (2.33 GHz Intel Core 2 Duo, 3 GB RAM, Mac OS X 10.4.8). Each Mac mini was connected to two cameras. The MacBook Pro was used to run the object detection software.

Latency Measurements: Methodology and results

Referring to Figure 5, there are five areas where significant latency may be introduced: (1) The time taken from the camera captures an image, until the image is available to a Mac mini for processing, (2) the time taken by the Mac mini to process the image, (3) the time taken to transfer processed data over the network to the MacBook Pro, (4) time taken by the MacBook Pro to detect objects using information gathered from all the Mac minis, and (5) the time taken to distribute the resulting object positions to the two games.

For Q3A, there is one additional, latency-inducing step. This step is the time from a gesture is recognized, until the action caused by the gesture is shown by the spectators. This latency stems from the required round-trip from a Q3A player via Q3A's server to the spectators.

The camera-induced latency is measured by pointing a camera at the screen attached to a computer capturing images from the camera. The computer's screen is initially black, before it is turned white. At this point, a timer starts. The timer stops when the images captured by the camera show a white screen, with the resulting latency being the elapsed time since the timer was started.

The processing-sensitive latencies (2 and 4) are measured by measuring typical execution times for the code that respectively performs image processing and object detection. The network latencies are measured by measuring the time taken to send a message from one computer via an event server to the target, and receiving a reply (similar to ping).

To avoid modifying Q3A's server, we determine the added latency in Q3A as follows. When the player fires his weapon, the Q3A engine will cause a weapon-fire sound to be played. We hook into the sound-playing code and start a timer when that sound is played. Each spectator reports back to the player when it plays a weapon-fire sound, yielding a rough estimate of the latency from when something happens at the controlling player, until it is visible to the spectators.

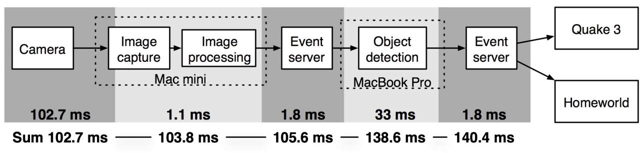


Figure 8. The latency from when the cameras grab images, until positions of objects are available for processing by either Q3A or Homeworld. Each measurement represents an average measure of the latency.

The results from our latency measurements are summarized in Figure 8. The additional latency introduced through Q3A's client-server architecture is shown in Figure 9. The average latency before an object's position is available to either game is 140 ms. The camera-induced latency is the greatest contributor, at about 100 ms. Object detection requires about 33 ms. For Q3A, the added latency averaged 87 ms with a standard deviation of 59 ms, with 1287 samples gathered from 9 spectators.

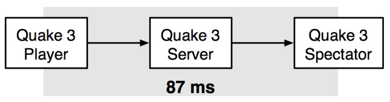


Figure 9. The additional latency as input events are delivered to a Q3A player, sent to the server and finally made visible by the spectators.

Rendering Performance: Methodology and results

The metric used to measure the performance of Q3A and Homeworld is frames per second. For both Q3A and Homeworld, input events are recorded over a period of about 30 seconds. For each experiment, the game is started in a known state, and the input events are played back[5]. During playback, the framerate is logged continuously.

We ran both Homeworld and Q3A in four different configurations, with 1, 4, 9 and 28 rendering nodes. For Q3A, we limited the framerate to 500, and measured the performance both when using Chromium to distribute the rendering, and when running the parallel version. The Q3A server used ran locally on the same network. For Homeworld, which we were not able to run using Chromium, we only measured framerate for the parallel version, and compared its performance to running Homeworld on a single display.

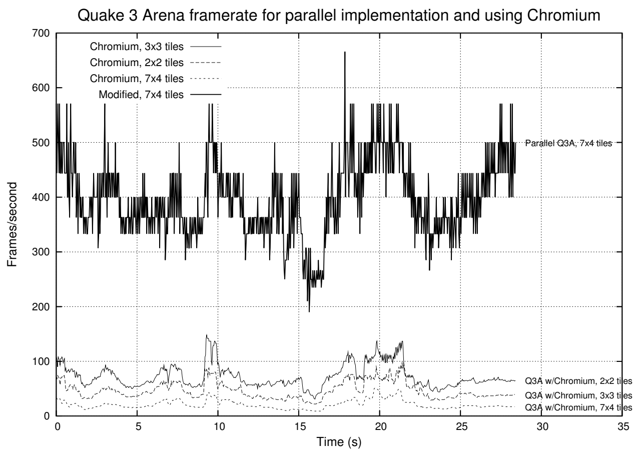


Figure 10. The framerate when running Q3A on 2x2, 3x3 and 7x4 tiles using Chromium, compared to the parallel version's framerate running on 7x4 nodes.

[5]This is similar to measuring Quake performance by running a timedemo. The timedemo mechanism already in Quake does not work for our parallel version, as it is designed to run on a single computer only.

Figure 10 shows the results from Q3A experiments. The peak performance with Chromium on 4 rendering nodes (2x2 tiles) is 148 FPS, and the average at 73. For 9 (3x3 tiles) nodes, the peak FPS is 97 and the average is 47, and for 28 nodes (7x4 tiles) the peak is 51 and the average 21 FPS. The figure only lists the results from the parallel version running on all 28 nodes - the reason is that there are no major differences in performance when varying the number of rendering nodes for the parallel version. The maximum framerate for the parallel version was 666, and the average framerate was 398.

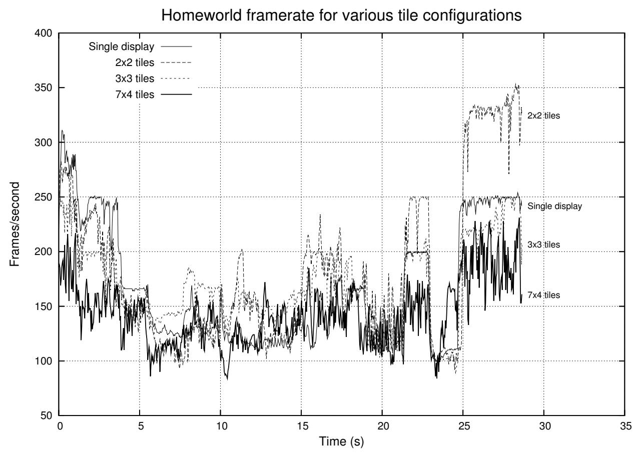


Figure 11. The framerate when running Homeworld on a single display, compared to running it on 2x2, 3x3 and 7x4 tiles.

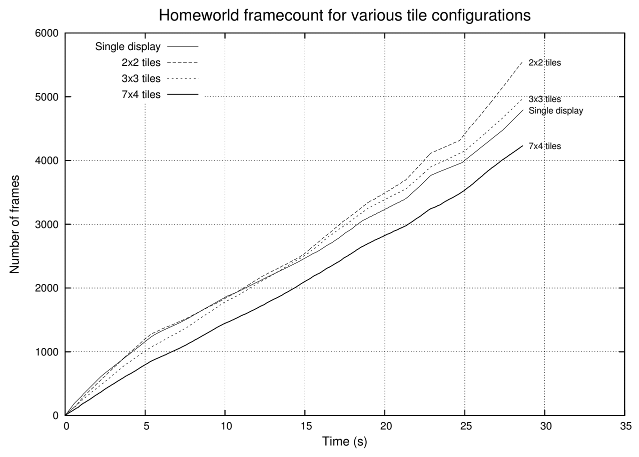


Figure 12. The total number of frames drawn when running Homeworld on a single display, compared to 2x2, 3x3 and 7x4 tiles.

Figure 11 shows the results from measuring Homeworld's framerate, while Figure 12 shows the cumulative number of frames shown by the game during the experiment. The framerate varies much more compared to the Q3A measurements. The maximum framerate for Homeworld running on a single tile, 2x2, 3x3 and 7x4 tiles were respectively 311, 353, 250 and 231. The respective average framerates were 168, 183, 169 and 143. Figure 12 shows that running Homeworld on both 2x2 and 3x3 tiles performs better than running it on a single display. The framerate was never lower than 80 for any of the configurations.

DISCUSSION

In [12], the authors investigate the effect of lag (i.e, latency) on human performance in interactive systems. As latency goes up, accuracy deteriorates and time to perform tasks increases. For this reason, it is important for the touch-free interface to provide input with as low latency as possible. In [1], the authors show that Q3A players prefer using Q3A servers where their average ping[6] is no more than 150-180 ms. The touch-free interface has a latency of 140 ms, and the average latency from the parallelized Q3A implementation is 87 ms. This gives a total latency of 227 ms, 47 ms more than the maximum preferred latency. The latency for Q3A fluctuated with a standard deviation of 59 ms, which may be an artifact of the latency measuring experiments, or a result of the Q3A server experiencing varying loads. The biggest contributor to latency in the touch-free interface are the cameras. Improvements in the cameras, I/O bus and OS may reduce this latency. The next-biggest contributor is the object detector. The detector waits for all the cameras to provide data before triangulating object positions. This synchronizes the cameras, using only fresh data from each camera for the triangulation. This results in improved accuracy. The cameras all run at 30 FPS, which corresponds well with the 33 ms average latency from the object detector. With cameras running at higher framerates, the latency incurred by object detection will be reduced, as less waiting will be required in order to ensure fresh data from each camera.

One problem with the touch-free interface is that its accuracy for positioning objects decreases as the objects move faster. This is caused by the use of many different cameras to capture images. Although each camera operates at the same framerate, they capture images at slightly different points in time. For a moving object, this results in the object appearing at different positions for different cameras. When these positions are used to triangulate an object's 2D position, the result is inaccurate. These inaccuracies appear as jitter in the object's vertical position. The horizontal position is also affected, although not as much as the vertical position. This problem can be alleviated by using cameras with higher image capture rates, or cameras where the image capture can be synchronized.

By utilizing Q3A's existing architecture when parallelizing it, we were able to rapidly port the game to the display wall's cluster. When running Q3A on the entire display wall, the framerate for the parallel version was an order of magnitude higher than the framerate achievable using Chromium. The performance penalty was an 87 ms increase in the latency from a player takes an action until it is visible on the display wall. This latency is independent of the input system used (keyboard/mouse or touch-free interface). Even better results may be achieved by parallelizing the game from

[6]The latency from a player takes an action until it becomes observable by other players.

scratch, but at the cost of a much greater effort in parallelizing it.

Homeworld's architecture made it possible to parallelize it by running synchronized copies on the tiles. However, to determine where to synchronize, we had to analyze the game engine, identifying all places where data is used that could impact the game simulation. At these places the copies must synchronize in order to use identical data. Finding all these synchronization points is difficult, and we have not verified that we have been able to identify all of them. We have only played the game's first level. To better check that all synchronization points have been identified, the entire game should be played from start to finish. Even then, minor bugs and timing issues can also potentially skew the copies out of sync. For these reasons, parallelizing Homeworld required more effort than parallelizing Q3A.

When running Homeworld on 2x2 and 3x3 tiles, the parallel version gave higher framerates compared to running Homeworld on a single tile. We did not parallelize the simulation itself, and all copies run the same simulation on the same data. Additionally, there is an increased overhead from synchronizing the copies. The fact that we still achieve a higher framerate for these tile configurations, is because the tiles share the rendering workload. For the 7x4 configuration, the framerate is lower than for a single display. We hypothesize that this is due to increased synchronization overhead, mainly from the MPI barriers used. We have not verified this.

Our expectations prior to implementing touch-free, multi-user support in Q3A and Homeworld were that using gestures to control Q3A would be awkward and difficult, while gestures for controlling Homeworld would be more natural as the pace of the game is slower and the gestures similar to emulating a mouse. Although we haven't conducted any formal user studies, our initial, subjective experiences indicate that the touch-free interface was more natural when controlling Q3A than controlling Homeworld. There are several potential explanations, including the characteristics of the touch-free interface and the intrinsics of the games.

CONCLUSION

We have presented a touch-free, multi-user interface for controlling applications on wall-sized, high-resolution tiled displays, and measured its responsiveness for controlling games. Games are useful in this context, as they generally require low-latency input to be playable.

Display walls provide high resolution by tiling a set of independent displays in a grid. The tiles are usually driven by a cluster of computers. The cluster-based architecture makes running existing games difficult, especially if good performance is to be maintained. This is due to the fact that practically no games are written to run on a cluster of computers or use more than a few displays.

We modified Quake 3 Arena (Q3A) and Homeworld, a first-person shooter (FPS) and real-time strategy (RTS) game, to run on a display wall, converting input from the touch-free interface to hand- and arm gestures. Players control the games by using one or both hands. The gestures for the two games are similar, but interpreted in different ways by the two games. We did not conduct any formal user studies to determine how well the gestures worked, but our subjective experiences indicate that using gestures to play Q3A works quite well, and better than controlling Homeworld.

The touch-free interface is built using 16 cameras and 9 computers. To determine the interface's responsiveness, we measured the latency between its different components. The total latency incurred by the interface was 140 ms. Of this, the biggest contributor was capturing images from the cameras, which incurred a latency of 102 ms. The touch-free interface's architecture is currently bound latency-wise by existing camera-technology. As camera technology improves, the intrinsic latency of cameras can be reduced, which will directly affect the latency of the touch-free interface. Cameras with higher image capture rates will also result in lower latencies for the object detector. This is because the detector can wait for shorter amounts of time before it has the most recent image data from all cameras. Another benefit from the architecture is that all image processing is done locally by each computer capturing image data. This reduces the amount of data required to be processed by the object detector by several orders of magnitude.

To run the games on the display wall, the games were parallelized. For Q3A, this was necessary to maintain high framerates when running the game on more than 2x2 tiles. Homeworld did not run on more than one tile of the display wall at all before being parallelized. Q3A was parallelized by exploiting its client-server architecture, and configuring a set of spectators to follow a given player. Each spectator's view was modified according to the tile on which it ran, resulting in several spectators contributing to a multi-tile view. Homeworld was parallelized by running a copy on each tile, and by keeping the copies' input and state synchronized.

We found the spectator-concept in Q3A to be useful in creating a parallel version of the game. We consider the spectator-concept as a single data, multiple view model. Homeworld did not support a similar model, requiring a much more labor-intensive effort to parallelize it. For this reason, we expect games with support for a single data, multiple view model to be more easily parallelizable for tiled display wall environments than other games. The drawback to using spectators for the purpose of parallelizing Q3A, is an increase in latency. No similar additional latency was measured in Homeworld.

For Q3A, we measured the performance of the sequential version using Chromium to display on one to 28 tiles. We were not able to run Homeworld at all using Chromium. For the parallel versions of the games, we measured their framerates running on one to 28 tiles. For Q3A, the parallel version runs with an average framerate of 398, an order of magnitude better than the sequential version's framerate of 21. The framerate of the parallel Homeworld was highest using four tiles and lowest using 28 tiles. The framerate was consistently above 80 regardless of the number of tiles used.

The high framerates, even for the 7x4 tile configuration with 7168x3072 pixel resolution (average 143 for Homeworld and 398 for Q3A), indicates that the parallelized games will scale to more tiles and higher resolutions. The framerates are well beyond what is displayable by a typical 60 Hz LCD panel, or in our case projectors with a 60 Hz refresh rate.

For Q3A, the overhead directly introduced by the parallel version results in an increase in latency of 87 ms. This is due to the player-server-spectator setup, and comes on top of the latency introduced by the touch-free interface. However, the touch-free interface's latency is bound by cameras, thus a reduction in its latency can be expected as camera technology improves. This will partially offset the effects of the latency induced by the player-server-spectator setup.

For Homeworld, the overhead of parallelization is mainly due to synchronization, as all the copies run in lock-step. In Q3A, the game simulation runs on a centralized server with each client showing a view of that simulation. In Homeworld, each copy runs its own simulation but with centralized synchronization of data.

ACKNOWLEDGMENTS

The authors wish to thank Espen S. Johnsen and Tore Larsen for their discussions, as well as the technical staff at the CS department at the University of Tromsø. This work has been supported by the Norwegian Research Council, projects No. 159936/V30, SHARE - A Distributed Shared Virtual Desktop for Simple, Scalable and Robust Resource Sharing across Computer, Storage and Display Devices, and No. 155550/420 - Display Wall with Compute Cluster.

REFERENCES

1. Grenville Annitage. An experimental estimation of latency sensitivity in multiplayer quake 3. In *ICON 2003: Proceedings of the 11th IEEE International Conference on Networks*, pages 137–141, 2003.

2. Steffi Beckhaus, Kristopher J. Blom, and Matthias Haringer. A new gaming device and interaction method for a first-person-shooter. In *Proceedings of the Computer Science and Magic 2005*, 2005. GC Developer Science Track.

3. Greg Burns, Raja Daoud, and James Vaigl. LAM: An Open Cluster Environment for MPI. In *Proceedings of Supercomputing Symposium*, pages 379–386, 1994.

4. Mark Claypool, Kajal Claypool, and Feissal Damaa. The effects of frame rate and resolution on users playing first person shooter games. In *Proceedings of ACM/SPIE Multimedia Computing and Networking (MMCN)*, January 2006.

5. Paul Dietz and Darren Leigh. DiamondTouch: a multi-user touch technology. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 219–226, New York, NY, USA, 2001. ACM Press.

6. Relic Entertainment. Homeworld. http://www.relic.com/rdn/, http://www.homeworldsdl.org/ and http://www.thereisnospork.com/projects/homeworld/.

7. Jefferson Y. Han. Low-cost multi-touch sensing through frustrated total internal reflection. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 115–118, New York, NY, USA, 2005. ACM Press.

8. Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 693–702, New York, NY, USA, 2002. ACM Press.

9. id Software. Quake 3 arena. http://www.idsoftware.com/ and http://ioquake3.org/.

10. Jeffrey Jacobson and Zimmy Hwang. Unreal Tournament for Immersive Interactive Theater. *Commun. ACM*, 45(1):39–42, 2002.

11. Kai Li, Han Chen, Yuqun Chen, Douglas W. Clark, Perry Cook, Stefanos Damianakis, Georg Essl, Adam Finkelstein, Thomas Funkhouser, Timothy Housel, Allison Klein, Zhiyan Liu, Emil Praun, Rudrajit Samanta, Ben Shedd, Jaswinder Pal Singh, George Tzanetakis, and Jiannan Zheng. Building and Using A Scalable Display Wall System. *IEEE Comput. Graph. Appl.*, 20(4):29–37, 2000.

12. I. Scott MacKenzie and Colin Ware. Lag as a determinant of human performance in interactive systems. In *CHI '93: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 488–493, New York, NY, USA, 1993. ACM Press.

13. Gerald D. Morrison. A camera-based input device for large interactive displays. *IEEE Computer Graphics and Applications*, 25(4):52–57, 2005.

14. Jakub Segen and Senthil Kumar. Gesture VR: Vision-based 3D hand interace for spatial interaction. In *MULTIMEDIA '98: Proceedings of the sixth ACM international conference on Multimedia*, pages 455–464, New York, NY, USA, 1998. ACM Press.

15. Bram Stolk and Paul Wielinga. Building a 100 Mpixel graphics device for the OptIPuter. *Future Gener. Comput. Syst.*, 22(8):972–975, 2006.

16. Edward Tse, Saul Greenberg, Chia Shen, and Clifton Forlines. Multimodal multiplayer tabletop gaming. In *PerGames '06: Proceedings of the 3rd International Workshop on Pervasive Gaming Applications*, 2006.