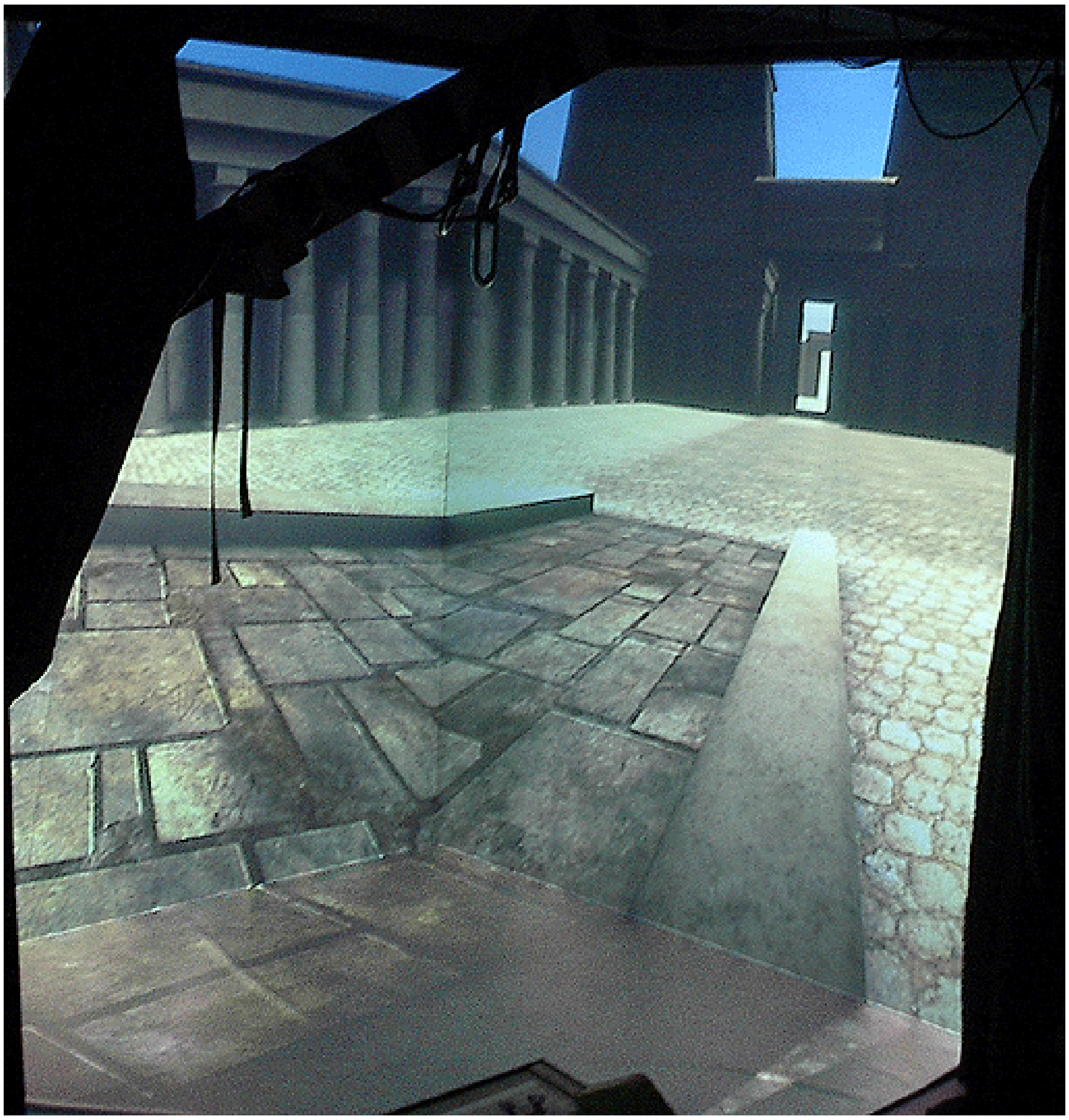
CaveUT 2003

Freeware for Low-Cost Integrated Multi-Screen Displays Using *Unreal Tournament 2003*



CaveUT 2003 is a set of modifications allowing the ***Unreal Tournament 2003*** (UT2003) game engine to display in **CAVE**-like displays and panoramic digital theaters. It supports off-axis projection for correct perspective and multiple views from the observer's viewpoint.

This page provides an introduction to CaveUT2003, including download information, installation instructions, guidelines for the program's use, explanations of how it works, and descriptions of improvements planned for the future. **CaveUT version 1.2** uses the older Unreal Tournament in much the same way, and is still available. Some sections of this documentation use images of CaveUT 1.2 in the general discussion.

All versions of CaveUT is open source and freely available to the public. It works under all versions of windows and we plan on porting it to Linux.

Page Index

If you have comments about CaveUT or this website, please contact **Jeffrey Jacobson**.

If None Of This Sounds Familiar

You may have arrived at this web site via a link, page redirection, because you were searching for a term that occurs on these pages, or by accident. In such a case, the information on the **top page** describing this web site may be confusing or unhelpful.

CaveUT is software that allows someone to set up a CAVE-like virtual reality environment.

"CAVE" is a recursive acronym for "CAVE Automatic Virtual Environment", coined in or near 1991 by Carolina Cruz-Niera, then a graduate student of Computer Science at the University of Chicago Illinois. She and her collaborators were the first to make a small room from (the equivalent of) giant computer monitor screens, using software to present an immersive view of a virtual environment. In other words, someone standing in front of the screens of a CAVE-like display will be able to look up, down, and sideways at the details of the world the screens show, making the experience far more realistic than staring at a computer or TV screen.

CAVE-like systems are used to allow people to experience large environments in a manner similar to walking through and looking at them. Such systems can be used recreationally (allowing people to play in computer games such as "first-person shooters") and educationally (allowing people to experience recreated historical sites, alien worlds, and the like).

The CAVE acronym is a registered trademark of the Trustees of the University of Chicago. This has led to all sorts silly permutations for similar devices. Examples include RAVE, NAVE, BNAVE, Grotto, Alcove, and CaveUT.

Next: **Introduction**

Previous: **Back to Page Index**

Start: **Back to Page Index**

Introduction

CaveUT differs from most CAVE-like setups in a variety of ways, but the difference that is most significant to many people and institutions is cost. Even a very elaborate CaveUT setup costs a fraction of what standard CAVE setups do. This makes CaveUT affordable for individuals, small companies, and budget-strapped academic departments.

CaveUT costs so little to implement for these reasons:

- **Hardware:** CaveUT installations require off-the-shelf personal computer technology.
- **Software:** CaveUT requires purchase of licenses to a low-cost "first-person shooter" game engine and installation of freely-distributable, zero-cost code provided at this site.

There is no predictable fixed cost for installing a CaveUT setup, but a ballpark figure for a typical two-screen setup (using all-new components) is around $12,000 for a portable system utilizing laptops or around $10,000 for a permanent installation utilizing desktop computers. The system can easily be expanded, adding one screen to a portable arrangement for around $6,000 or adding one screen to a permanent arrangement for around $4,500. These prices assume that the builder is buying everything new--it is entirely possible to use borrowed or existing equipment!

Currently, CaveUT offers a (compelling!) monoscopic image and no head tracking, although stereo and tracking will be added. Compared to most software authoring environments used in CAVEs, it offers superior graphics quality and greater ease of content production and networked communication. Using a true game engine, as CaveUT does, is best for virtual worlds populated with pre-defined shapes, be they rigid or flexible. The only situation where it is not appropriate is for applications (e.g. scientific visualization) where shapes have to be generated on-the-fly from a data stream or some algorithm.

A detailed account of the costs and setup requirements for an implementation of CaveUT appears on the **Physical Setup** page.

How CaveUT Works

Typically, each player of a networked *Unreal Tournament 2003* (UT2003) game has a standard desktop computer running Windows, Linux or MAC's OS-X. Each has a complete installation of the game, which we'll refer to as a client installation. Each client communicates over a LAN or through the Internent with the server installation, a process usually running on a desktops along with a client.

The server maintains the authoritative copy of the virtual world which all the players share. This allows the players to interact with each other -- they move and function within the exact same environment, though each player experiences it from his own perspective.

Each client maintains a complete copy of that world and all the things in it--including the figures representing each player. (These figures, referred to as "avatars," are usually, but not always, human.)

Every time a player takes an action, the environment's creatures and avatars, in the client copy of the virtual world, respond appropriately. As often a possible, each client tells the server what has changed as consequences of its player's actions. The server reconciles the actions of all the players and issues an authoritative update to all clients, so they all change their copies of the world to reflect the new state of things. The process is remarkably seamless if the network will allow these updates to happen frequently enough.

This arrangement has a lot of virtues. It allows for graceful and robust recovery from unexpected network delays. It also uses the minimum bandwidth possible, because the server and clients need only exchange updates, not models or imagery.

How Spectator Mode and Open Source Make CaveUT Possible

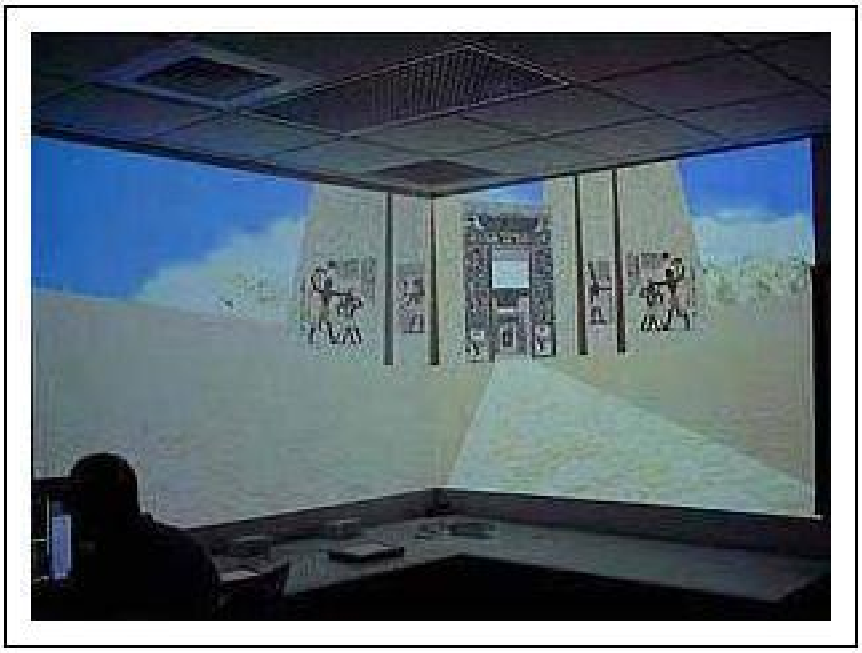
While the rendering engine for UT2003 is proprietary, everything else in the game's software is open source. This allows a large community of fans, programmers and even some scientists to make changes and distribute them freely, as with CaveUT. UT2003 and its modifications are available free to the public for most non-commercial purposes, while Epic Games retains the sole right to make money from their use.

Rather than showing up with an active avatar in the game, a player can act as a spectator instead. Using the UT2003 software's "behind view" option, the spectator player can see through the eyes of any other player in the game. In each of the installations depicted, here, there is only **one** regular player at a time.

Note: The images on this page show the old CaveUT 1.2, adequate for demonstration purposes.

For simplicity's sake, this documentation will use the following terms with the following definitions: The **player** is the person who controls the character or avatar moving within the virtual world. The **operator** is the person who controls the CaveUT computer setup (and may or may not be the same person as the player). The **console** is the computer from which the operator controls CaveUT.

Each desktop computers showing a spectator view is connected to a digital projector, which projects the view onto one wall.



In the two-screen example showed at CHI 2002 (shown to the right), and with the five-screen **Earth Theater**, all the views are front-projected. In the **four-screen BNAVE**, the three vertical walls are rear-projected and the one floor screen is front-projected from overhead.

CaveUT is a set of modifications to UT's open-source game code which changes the view rotation and perspective correction of a spectator's view.

In the simplest possible implementation, a CaveUT setup is two-walled, using the corner of a room as its projection surface.

In the UT2003 software one spectator's view is hacked to look 45 degrees to the right, rather than straight ahead, rather than the straight-ahead view the console shows. That image is projected on the screen on to the right of the optimal viewing point. The other spectator's view is turned 45 degrees to the left and shown on the wall to the viewer's left. The perspective correction is changed on each screen to be account for the established viewing point.

This provides a single, contiguous image to the viewer. It is immersive because the perspective correction is approximately the same as if the virtual world were real the viewer were seeing it from that location.

As the operator navigates through the virtual world, the view on the two screens change in lockstep. Using a typical LAN, the client-server updates can be so quick that there is no latency between the screens. The game provides nearly thirty frames per second, which is identical to the refresh rate utilized by commercial television. There is no need to explicitly synchronize the screen updates when they update so quickly.

Getting Creative

As you might expect, a CaveUT setup can have many screens. Currently, UT2003 has a limit of 32 players allowed in any one game, so it is possible to implement a 31-screen display.

The screens can be oriented in almost any arbitrary orientation and distance to the player. The screens don't even have to show the same view! For example, you could build an airplane or even mini-submarine cockpit simulator with a separate CaveUT view for each viewport or window. For a driving simulator, you could simulate a rear-view mirror with a very small CaveUT screen.

It's even possible to have multiple CaveUT instances running in the same world environment in a fashion similar to several players participating in a game of *Unreal Tournament*. The total number of servers and clients, regardless of how they are divided among CaveUT instances, count against *Unreal Tournament*'s 32-player limit.

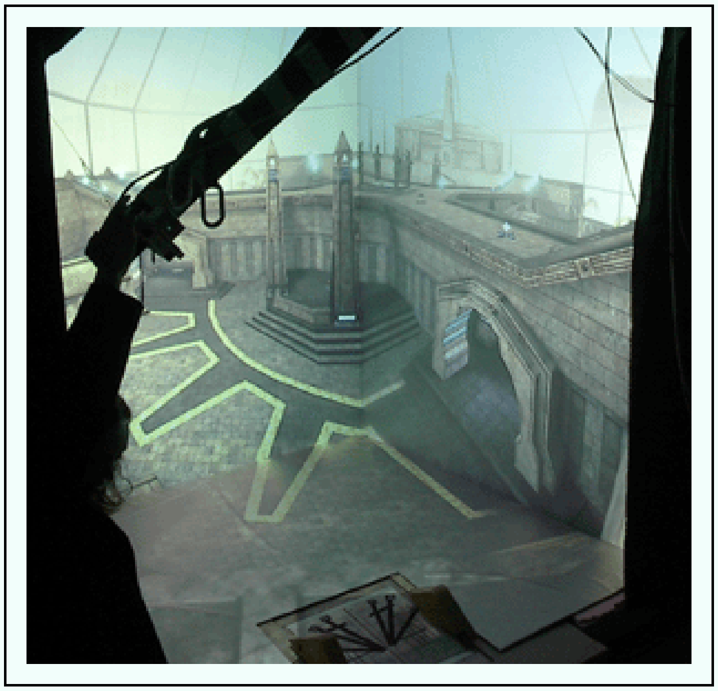
To do this, you would one computer on your network hosting a game; it would connect more than one CaveUT setup. If a specific CaveUT setup had four spectator/clients focused on one operator console, this would use up five players against *Unreal Tournament*'s limit of 32. Six such setups would account for 30 players of UT's 32-player limit, and so six different players could interact in the same game environment while experiencing it through an immersive CaveUT interface.

Next:	**Examples of CaveUT Implementations**
Previous:	**If None Of This Sounds Familiar**
Start:	**Back to Page Index**

Examples of CaveUT Implementations

BNAVE

CaveUT was originally developed on the **BNAVE**, a PC-based CAVE-like display the Medical Virtual Reality Center, Department of Otolaryngology, University of Pittsburgh.

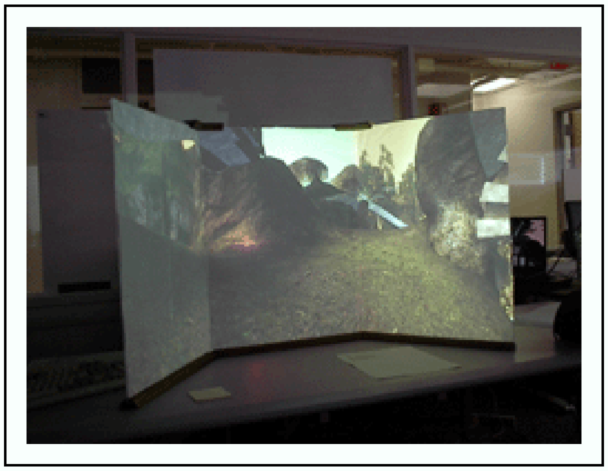


To the right is a picture of the BNAVE showing DM-Suntemple, a virtual world that comes prepackaged with UT. The screens are at an 80.5-degree angle to each other, and we are looking straight into the corner. (The image looks bent because the camera taking this picture could not be placed in the ideal viewing location; the image does not appear bent to someone standing in the correct position before the screens.)

The BNAVE consists of three walls arraigned in a "U" around the viewer. A fourth projection surface has been added to the floor.

Each wall is a rear-projection screen. Onto each of the three screens is rear-projected projected the SVGA output of a dedicated PC.

Mini Cave at VISIC Lab


DM-Antauls

DM-Serpentine

Physical Setup

These three images show CaveUT in the "Mini Cave" at the **VISIC Lab** at the University of Pittsburgh. The mini-cave is a three-screen affair small enough that all the computers required to run will it fit on a desktop. Its basic design is much like the BNAVE or any other generic PC-based cave. The first image shows a virtual world called DM-Antauls. The second shows a virtual world called DM-Serpentine. The last shows the physical setup of the computers running the mini-cave.

The VISC Lab also built a complete V-Cave (see below) for more immersive applications.

The Portable V-Cave

Illustrated here is the portable V-Cave, which was demonstrated at **CHI 2002** and will be shown at the **Ultra Unreal** and **HFES 46th meeting** events.

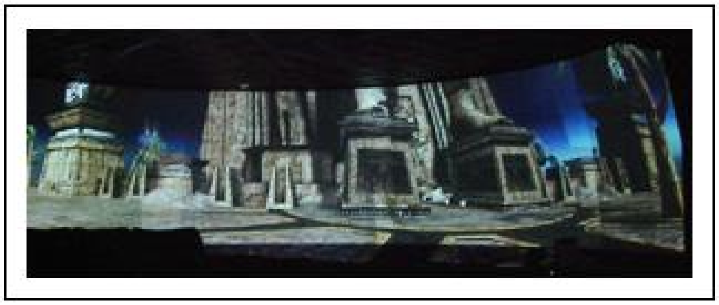
This setup uses tripod-mounted projectors to project onto screens stretched from a collapsible frame made from PVC pipe. The basic configuration is much as same as in Figure Two.

More screens can be added to this arrangement; the only limits are the *Unreal Tournament 2003* limitation of thirty-one screens, and the limits on the fabrication skills and patience of the people putting together the CaveUT setup.

The screens

can be front or rear projected, or can include some of each. The only requirement is that a user in the ideal viewing location should not be in the way of any projectors.

Earth Theater



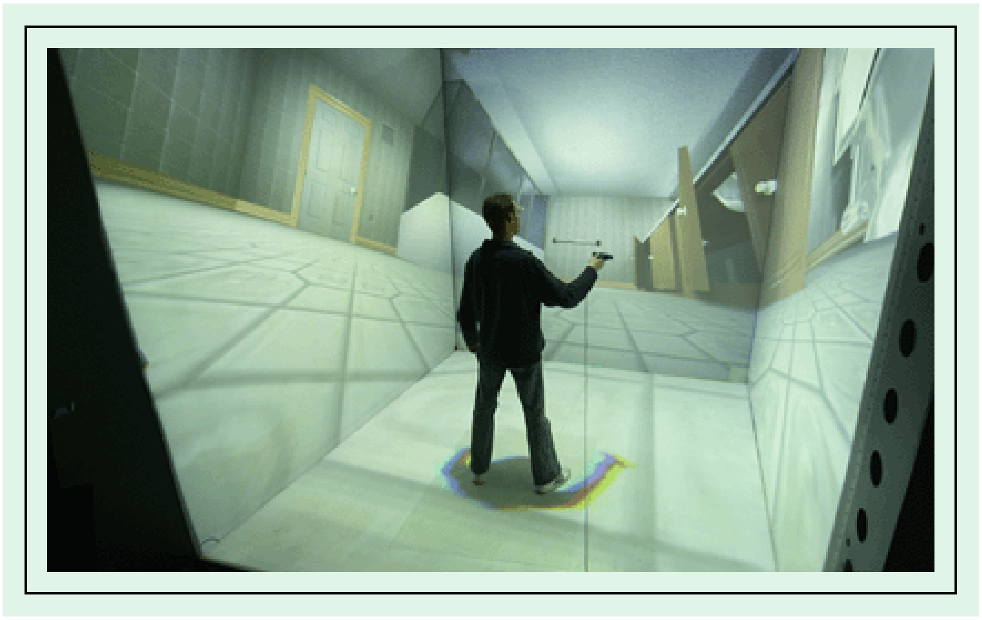
The **Earth Theater** (left) at the Carnegie Museum of Natural History has a CaveUT installation. The theater has a fully digital display composed of five curved front-projected screens spanning 210 degrees horizontal and 30 degrees vertical. Five standard video projectors, each driven by a PC running windows, produce the Unreal Tournament display. As you would expect, each of the five projector PCs is running a spectator, which provides the appropriate view from a single player on a sixth PC.

Interestingly, the careful design of the theater makes off-axis projection unnecessary. However, the curved screens require a spherical correction of the image, otherwise we get those wedge-shaped overlaps you can see in the image if you look closely. Willem de Jonge are just now working on a spherical correction to the OpenGL code.

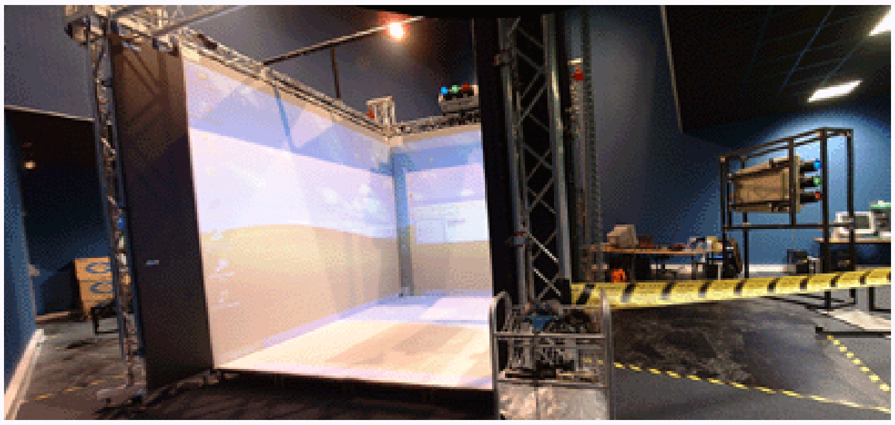
On the right is a Schematic of the Earth Theater. The screen is a section of a sphere, 210 degrees horizontal by 30 deg vertical. In the figure, the screen is depicted in a transparent white in front of the seats.

CLARTE

CLARTE (**www.clarte.asso.fr**) is a European research center specializing in immersive displays for virtual reality and advanced perhipherals for VR. They are using CaveUT in their SAScube (**www.sascube.com**) a fully equipped four-walled CAVE-style display.



Their programmer and main contact for their CaveUT project is **Marc LeRenard**. He has already added stereographic (3D) imaging to the CaveUT and will soon add real-time head and hand tracking.



Next: **Hardware and Software Requirements**

Previous: **Introduction**

Start: **Back to Page Index**

Hardware and Software Requirements

Unreal Tournament 2003

For *each* PC in your mini-network, you will have to buy and install a copy of *Unreal Tournament 2003*. Copies are available from many software retailers, including several linked to the *Unreal Tournament* **web site** (click the **Buy Now!** button in the upper right to reach a page of online retailers). **Be sure to buy Unreal Tournament 2003, *not* 2004.** UT2004 will not actually be released until mid-March! You can buy it, but you can't *have* it....

In addition, for each copy of UT2003, you will need to install the latest **code patch**.

Operating System

Unreal Tournament 2003 operates under most modern flavors of Microsoft Windows (98, ME, 2000, XP) and (soon) Linux. CaveUT's modified OpenGL code currently works only under Windows, limiting implementations of CaveUT (for the time being) to that platform. However, the OpenGL code will soon be ported to Linux, allowing for CaveUT installations under that OS.

CaveUT's design team will not be porting the code to Macintosh OS or the Xbox, but would enthusiastically support anyone who wants to undertake such a project.

All the instructions currently on this web site assume the user is using some version of MS Windows.

Hardware

CaveUT requires use of multiple PCs that can run *Unreal Tournament 2003*, which generally means run-of-the-mill PCs with good graphics cards.

For details of UT2003's hardware needs, see **Epic Games' FAQ** page.

A full CaveUT setup also requires a variety of projector and networking hardware, depending on how ambitious you are. An illustrative, simple, example is described in the **Physical Setup** page.

Next: **Physical Setup**

Previous: **Introduction**

Start: **Back to Page Index**

Example Physical Setup: The V-Cave

This page has full instructions for setting up a simple CaveUT display using two screens. (Hence the name V-cave, because the two screens form a letter "vee".) The methods shown here generalize easily to more complex multiscreen arraingements. You can click on most of the illustrations shown below to bring up larger versions.

Note that some of these instructions will be unclear until you have also read the pages on software installation.

Using the Corner of a Room

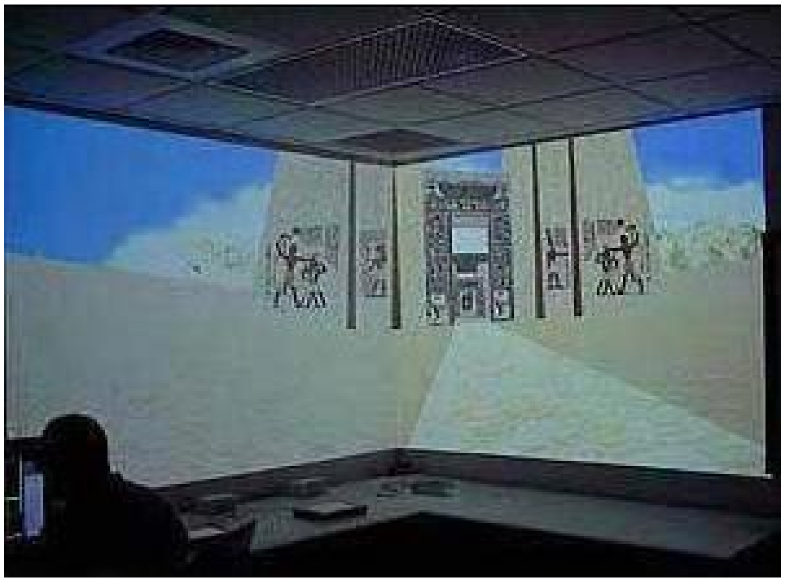


Figure One: The Two-Walled CaveUT Display

The simplest physical setup for a CaveUT display doesn't require that you have screens -- you simply need the clean, white corner of some room. (In academic and packrat environments, such a corner will be difficult to find.) If the walls are not white you can cover them with everything from paper to screen material. White shower curtains work rather well, as do very large pieces of white paper. The brighter the projectors are, the less pristine the surfaces need to be; in the illustration shown to the right, the walls are actually 50% gray!

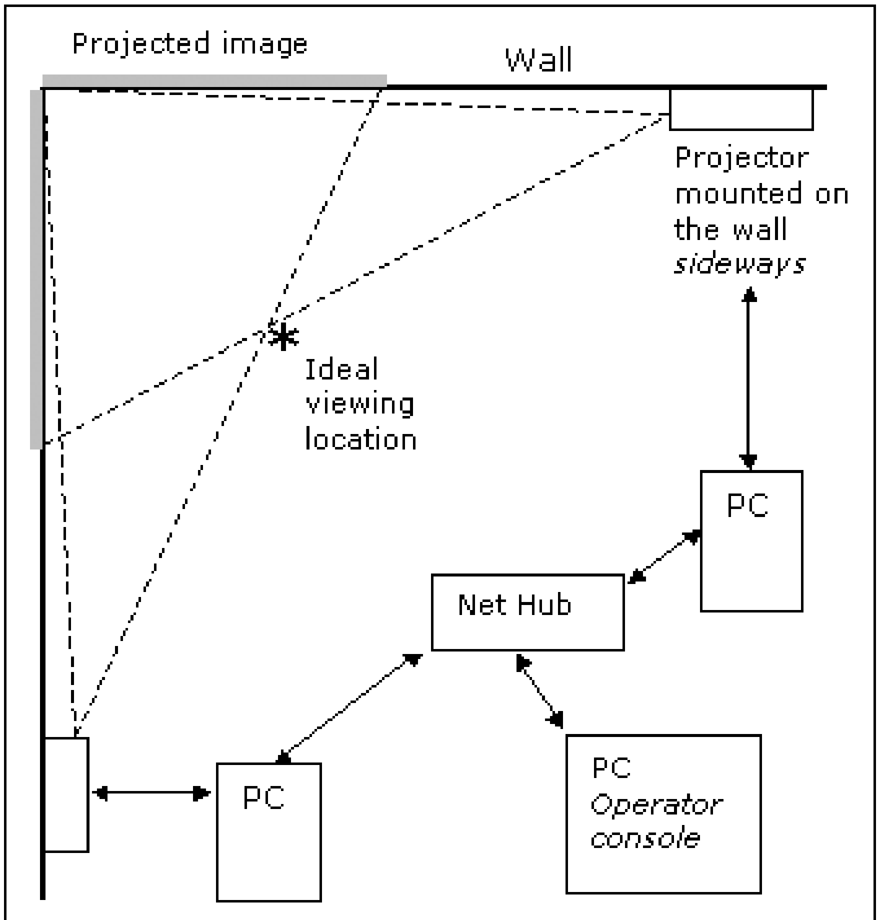


Figure Two: Schematic of the Minimal UT-Cave

In this setup, two digital projectors are each mounted on a wall and pointed into the corner of a room, as shown in Figure One. Each projector displays on the wall perpendicular to its own. One edge of each projection lies along the corner itself, so the two images abut.

If the projectors are bright enough and the wall is both clean and white, no screen material is
needed. Otherwise, you can use some covering material like very large sheets of paper, white foam core, white plastic, or Tyvek®.

Figure Two shows the layout, from overhead, of the components used for the corner CaveUT setup shown in Figure One. The setup is controlled from the server computer, labeled "PC Operator Console" on the diagram. From the Console, data is sent to the Net Hub, and from the hub to each secondary (client) PC. Each PC runs a client version of *Unreal Tournament 2003* in client mode and sends its data, in turn, to the Projector associated with it. Each Projector displays its portion of the UT2003 player's view to the corresponding screen (or, in this case, portion of wall).

The setup shown in Figure One has its ideal viewing point, or "sweet spot," at the location shown in Figure Two. This gives a viewer there approximately 100 degrees of horizontal FOV.

Note that the setup shown in Figure One differs from the diagram slightly: Its the projectors are not turned sideways. One is on top of a bookshelf and the other hangs upside-down from hooks in the ceiling, as shown to right, demonstrations of the ways one can (and often must) improvise when accommodating a CaveUT setup to its environment.

Necessary Components

Following is an inventory of the parts used for the two-walled **V-Cave** (which was described on the **Examples of CaveUT Implementations** page, excluding the projection surface. In many cases, setting up an implementation of CaveUT can require even fewer materials, and should not require more.

Standard 100mbps Unmanaged Ethernet Hub ($50-$100)

It's fine to purchase an inexpensive hub (which is also sometimes called a "Network Switch"), but be sure to get one that is physically rugged, particularly if the CaveUT setup is to be regularly transported.

Four 25' 10-based-T LAN cables and One Coupler ($100)

These cables are also often labeled as "RJ45 CAT-5e Patch Cables" or something similar. These lengths are sufficient for a CaveUT setup like the one shown in Figures One and Two; a more ambitious setup, or one where the console and client PCs are set up well away from the screens, will call for longer cables.

Three Standard 25' Power Cords And Two Surge-Protected Power

Strips ($60)

Be sure to plug everything into the surge protectors for the safety of the equipment. It's also important to be sure its the surge protectors are of a good, reliable brand. If the CaveUT setup is to be regularly transported, it's also helpful to have protectors that are physically light and small.

You could even choose to use an Uninterruptible Power Supply (a UPS) if you are uncertain about the quality or stability of your power supply.

Three Desktop Computers (Approximately $1500 to $2500) *or*, Three Laptop Computers (Approximately $4000 to $7000)

Desktop computers are best-suited to a CaveUT implementation that is intended to stay in one specific location; laptops are better suited to implementations that will be moved on a regular basis (such as for computer and trade shows, for demonstrations at various schools and college departments, etc.).

Regardless of whether the machines are desktops or laptops, they should be fairly robust, in terms of graphics cards and processor speeds, to produce the best graphics performance.

The V-Cave shown in Figure 1 utilized two Pentium 4 1.1GHz laptops, with substantial amounts of RAM and geForce2 cards, for the client machines; they were operated at 1024x768 resolution. The console laptop was much more modest, a Pentium 3 800MHz machine with 128Meg RAM, running Windows NT.

Typically, the console machine only has to be strong enough to provide smooth interaction.

The machines mentioned above are more than enough for showing off CAD models. If a CaveUT setup shows off lots of human figures, or very complex models, a falloff in performance may result. (Should this happen, and should no more robust hardware be immediately available, a useful trick is to reduce the *Unreal Tournament 2003* resolution and rendering quality values.)

The client machines must be running Windows, at least until the Linux port is done. However, the server machine can run MAC-OS, Windows, or Linux, and it only has to be fast enough to provide smooth interaction between the machines in the network.

A Spare Keyboard and Two Keyboard Extension Cords ($35)

This extra keyboard is the input device used by the player to navigate and interact with the virtual world. A CaveUT setup can use any kind of control device -- keyboard, joystick, trackball, or other favorite game peripheral.

The extra extension cords are recommended for a setup in which the player is also the operator; it gives the player/operator much more freedom of movement, which is particularly useful during demonstrations and presentations.

Two Tripods ($200+)

The tripods are used to hold up the projectors. It's important to purchase sturdy, good-quality tripods; cheaper ones will be flimsy. Though they may be capable of holding up the projectors' weight, they may bend or be unbalanced, denying the CaveUT setup their full range of motion.

Typically, a good tripod (about $100 and up) has a simple platform (the head) with a slot in it and a single bolt. Attach the projector onto the bolt, and the tripod both holds the projector and allows it to be angled it in any direction. The stronger the tripod and the lighter the projector, the easier it will be to make fine-tuned adjustments.

Make sure that after the projector

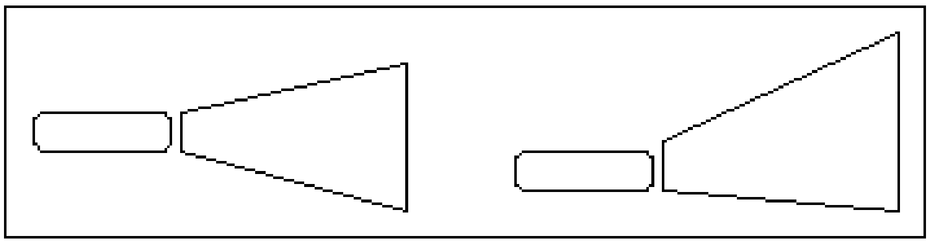
is mounted on the tripod, the tripod can rotate the projector a full 90 degrees to the right *and* a full 90 degrees to the left. This capability is needed for situations in which the projectors will need to be mounted sideways, so that the projection area can be taller than it is wide.

Two Digital Projectors (Approximately $7000)

Important Note: It is critical that small projectors, those weighing nine pounds or less, be purchasd for a CaveUT setup. Small projectors are:

- Easier to mount
- Possible to turn sideways
- Easier to transport
- Less prone to falling over atop their tripods when bumped by a passer-by, costing thousands of dollars to replace
- Much, much better at heat dissipation (they have powerful fans built into them which will keep their internal components cool at any orientation; some larger models will overheat and burn out if they are held sideways and run long enough)

Warning: just because a projector can be turned upside-down to be ceiling mounted does *not* mean it can be turned sideways; some are engineered so that air will flow them correctly when they are upside-down but not when they are sideways.



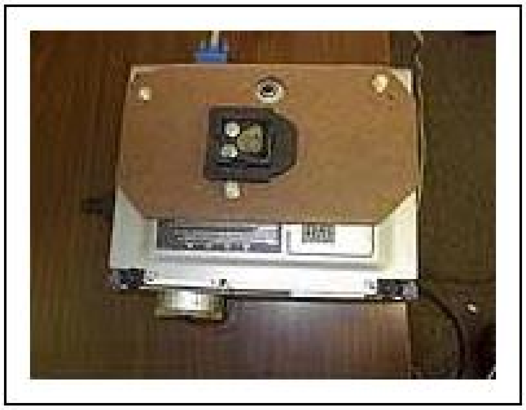
A Tangent on Keystone Correction: Most commercially available projectors have a lens inside them which introduces a keystone correction. This causes the beam to spread upward as shown in the illustration to the right. In the illustration, the left-hand projector has its beam configured normally; the right-hand projector has its keystone correction implemented, giving it a lop-sided projection beam.

This feature of projectors allows a projector to be set up on a conference room table and have its display shown on the wall *without* the projector having to be propped up on books or other elevators.

This feature is a *very good thing*. It allows for the arrangement shown in Figure Two. With the projectors turned sideways and keystone correction implemented, the projection beams allow the viewer to stand about as close as most people ever want to. In general, this it feature provides a CaveUT implementation with two possible locations for a projector to project onto a particular screen.In the boardroom example, an upside-down projector, mounted on the ceiling, could project onto the same screen as it would if it were just sitting on the main table. Said another way, the keystone correction for a sideways mounted projector can be made to go left or right simply by rotating the projector 180-degrees. In fact, the two projectors in the V-Cave are turned sideways in *opposite* directions.

Without this advantage, the **BNAVE** could never have been made as large as it is, given its space constraints.

Materials for Mounting Plates for The Two Projectors (Cost Negligible)



The projectors need to be attached to the tripods. Most projectors are designed with some means to mount them upside-down from a ceiling; most have threaded bolt-holes cut directly into the projector case. This is a flimsy arrangement, particularly since the projector cases are often made of plastic. It's much better to spend a little time making mounting plates that attach directly to the projector case, then attach the tripods to those mounting plates.

The figure to the right shows a projector with its mounting plate attached.

(It's also true that for most projectors, mounting brackets are available for sale. However, they are usually designed for ceiling mounts, they are *always* outlandishly expensive for what they offer, and they are sometimes not engineered to allow the projector to be mounted on a tripod. For all these reasons, it's better to build mounting plates from scratch.

An easy approach is, for each projector, to use one-foot square pieces of 1/4" hardboard, metric screws that correspond to the threading of the holes in the projector cases, and some other very small items. (Detailed instructions for building mounting plates from scratch appear below, under the headline **Building Projector Mounts**.) It's best not to purchase the mounting plate components in advance -- wait until the projectors themselves have been purchased and the setup of the entire CaveUT implementation has been determined, at which point your precise needs will be readily evident.

Materials For Screens (Cost Variable, $0-$650)

Finally, there are the screens themselves -- the surfaces the CaveUT images will be displayed upon.

The cost of screens varies widely. The cheapest screens consist of two walls that are already painted white -- net cost zero. Other approaches to screens include purchasing large commercial projection screens or fabricating screens from scratch, designed to the specifications appropriate to your own CaveUT setup.

Important: With bright projectors, especially, you can get away with all sorts of substandard projection surfaces, but there absolutely cannot be any gaps or ridges in the material. Also, any join between two screens has to be absolutely perfect. Much of design of the V-Cave is centered on keeping the seam between the two screens clean and straight.

Below, under the headline **Building Portable Screens**, are instructions for building an inexpensive set of screens suited to a portable CaveUT setup.

Building Portable Screens for a V-Cave

Components and Costs

Screen Material ($20-$500)

The screens shown above were constructed with two 70"x88" sheets of material used in the making of banners -- the sort that proclaim the opening of new restaurants or the availability of apartments for lease. The precise material used in these screens was "FS 12oz White H/G" with grommets at one-foot intervals on all sides. Such material can be purchased from many banner printing companies. In the V-Cave setup, the sheets cost a total of $160.

For the first V-Cave prototype, white shower curtains, with grommets added to the edges by hand, were used. This worked reasonably well, but the banner material is much better. Composed of three layers of plastic -- one black layer in the middle and two white layers on the outside -- banner material prevents light from getting though, which can be important in an environment with a lot of ambient light. In a front-projection setup, any backlight will degrade the projected image.

It's also possible to order professional screen material from companies that sell projection and presentation supplies and components. (Cost for enough material for two screens would be approximately $500.) These materials fold up nicely and does not crease like banner material does. You can usually ordered with grommets installed and some screens are suitable for back-projection.

It's helpful to remember that the better the projection surface is, the more of the projector's light it will reflect. However, if you have a dark room and good projectors, you can get away with using inferior screen materials.

As a rough guess, banner material reflects about 80% as much light as a professional screen, while shower curtains reflect about 60%. The 1000-lumen bulbs used in many projectors look like 800-lumen bulbs when shining on banner material or 600-lumen bulbs when shining on shower curtains. In ideal circumstances -- that is, in a dark room -- shower curtains can be more than sufficient.

Note also that shower curtains do let enough light through (about 40%) to be used as a rear-projection material, again assuming a sufficiently dark room and sufficiently bright projectors.

Another advantage to very bright projectors is that they wash out imperfections on the screen. Dirt, scratches and creases usually get washed out in the brighter image.

Finally, any projection surface, except a professional screen material, can change the overall color of the scene projected onto it. It's possible to compensate for this by changing the color balance in the projectors.

Plastic Hammer ($20)

The screen frame will be made of PVC pipes. This plastic hammer is for pounding it together and apart.

Avoid the black hard-rubber mallets. They smell bad, leave black marks on the pipes, and tend to become damaged by use.

Pipe Cutter ($25)

Use this to cut the PVC pipes to make the screen frame. You could use any fine-toothed saw, but the pipe cutter is much faster and easier, and gives a smooth, even cut every time.

Screen Frame (90' of 1 1/4" PVC Pipe) ($50)

Obviously, there all sorts of materials are suitable for the construction of a screen frame. PVC pipe is a good choice because it's cheap, not-too-heavy, easy to work with, easily replaced, and rugged.

Cut the pipe into the following pieces:

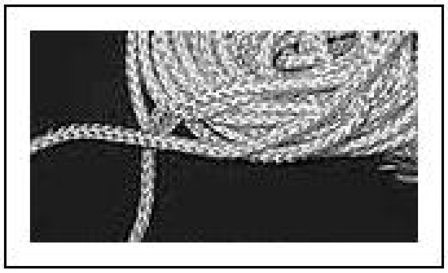
27	31" segments (one is a spare to account for loss or breakage)
2	27.5" segments
4	12" segments
8	2.5" (tiny) segments

You will also need the following specialized PVC pipe pieces, normally available at the same retailer that supplied the PVC pipe:

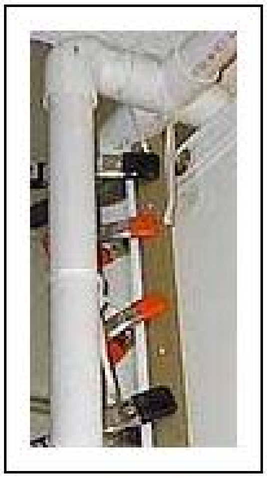
19	straight joiners (one is a spare)
9	90-degree turn joiners (one is a spare)
3	four-way "cross" joiners (one is a spare)
5	45-degree turn joiners (one is a spare)
5	T-joiners (one is a spare)

When these components are first assembled into a frame, it will be terribly unstable, wobbling drunkenly at the merest push. However, once the screen material is on and stretched, the whole thing will become much more stable.

100 feet of 1/4" nylon cord ($3)



It's best to choose one of the more slippery or slick varieties of nylon cord. This makes it much easier to adjust tension of the screen material on the frame -- efforts to tighten the screen material in one spot will spread to nearby grommets on the screen material. This greatly reduces the amount of fine-tuning required by the screen assembly.



16' of Carpet Seam Binder ($10)

A "carpet seam binder" is a strip of metal (as shown to right) which is sometimes used in the United States to cover a seam between two carpets, especially if they are of different types or there is a gap between them. With the V-Cave, carpet seam binders were used to clamp the edges of the two screens together at the corner where they join, as shown to the left.

It is very important that this corner be even and smooth. For rear-projection, the joining mechanism has to be very thin where it touches the screens, otherwise it would get in the way of the projections. The carpet seam binders are perfect for the job.

Exact instructions for use of the carpet seam binders appear below, under the **How to Assemble the Portable Screens** headline. Note that the end pieces each have a hole at the one end, as shown in the illustration to the right; they are used for threading the rope through.

Cut the seam binders with a hacksaw (they are made of aluminum) and smooth the edges with fine-grained sandpaper. Use the sandpaper to round off the corners of the seam binder strips so they don't puncture the screen material or damage other screen components.

Package of Fine-Grit Sandpaper ($3)

Sandpaper is used for grinding off the sharp corners of the carpet seam binders.

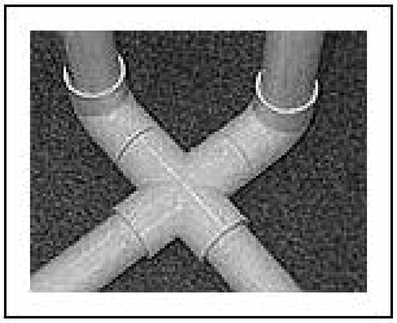
Twenty 2" hand clamps ($40)

See the left-hand illustration beside the **16' of Carpet Seam Binder** headline. The black clamps shown there are 2" clamps. In the example shown in the illustration, ten 2" clamps and ten 1" clamps were used, but it's easier to use 2" clamps exclusively, and these instructions assume that's the technique that will be used. (In other words, do as we say, not as we do.)

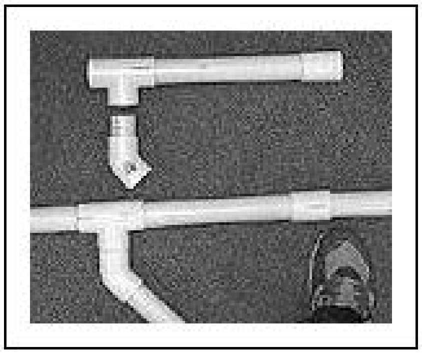
As the illustration shows, these clamps attach the carpet seam binders to the screens, effectively creating a strong seam. The clamps also act as grommets in that nylon cord will be threaded through them as shown in the illustration under **Step 3** beneath the **How to Assemble the Portable Screens** headline below. During assembly, you'll thread the nylon cord through the clamps as shown in that illustration.

How To Assemble the Portable Screens

The general configuration of this portable setup is shown on the illustration of the screens (without projections) under the **Portable V-Cave** headline on the **Examples of CaveUT Implementations** page.



To the left is a close-up of the lower part of the central corner, while the picture to the right illustrates how those two diagonal cross pieces attach to the frame. The cross-pieces are only needed to increase the strength and stability of the frame. The picture to the right also shows how the very short pieces of PVC pipe fit the joiners together into larger configurations.



Follow these steps to assemble the various components into screens.

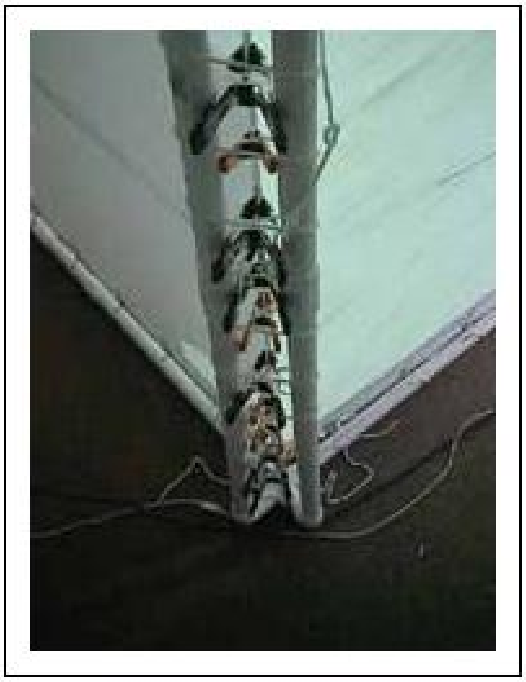
Step 1

On a clean floor or a large clean surface, lay one of the screens face-up.

Step 2

Lay the other screen on top of it, face-down. The grommets on the two screen pieces will not line up, but that's not important.

Step 3



The "stack" of two screens is rectangular. Choose one of the long sides of this stack and lay three of the carpet seam binders on top of the screens, parallel to that edge and about 1" to 1 1/4" from the edge.

Arrange the other three carpet seam binders in exactly the same way, but upside down, underneath the screens, along the same edge. The two rows of seam binders should be facing each other.

One by one, put the metal clamps into place, so the seam looks like the one shown in the illustration to the right. While doing this, make sure that the edge of the two screens is pulled taught. At this point in the process, there will be creases and wrinkles in the screen material, and it's important that no creases or wrinkles remain between the seam binders.

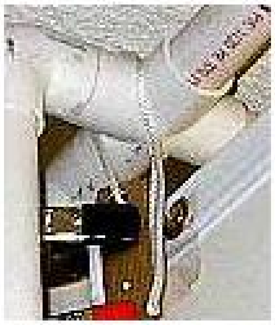
In addition, it's important to make sure that the carpet seam binders are exactly the same distance from the outer edge of the screen at all points. In other words, they need to always be exactly 1" from the edge all along the length of the seam, or all exactly 1 1/4" or even 1 1/2", so long as they are consistent.

Be careful that the bottom pieces perfectly line up with the ones on top.

Assembly of this step is easiest when three people work together, though one person can do it.

Finally, it's best not to stress too much about this step. If it is done incorrectly, it's possible to fix matters later.

Step 4



With the clamps and the carpet seam binders, you now have a seam joining the two screens. Decide which end of the seam is going to be the top, and then put a two-foot piece of cord halfway through the two holes of the carpet seam binders, as shown to the right.

Step 5

Carefully, move the screens to the frame. One person can do it, but it's easier with two people involved.

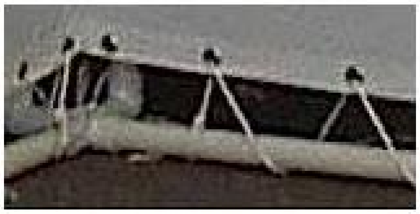
Bring the top end up to the top corner of the frame.

Tie the piece of cord around the 4-way cross piece joiner at the top of the frame. Be sure to wrap the cord through opposite corners of the cross piece, as shown in the illustration that accompanies Step 4. This will allow the seam to hang directly underneath the center of the cross piece.

Step 6

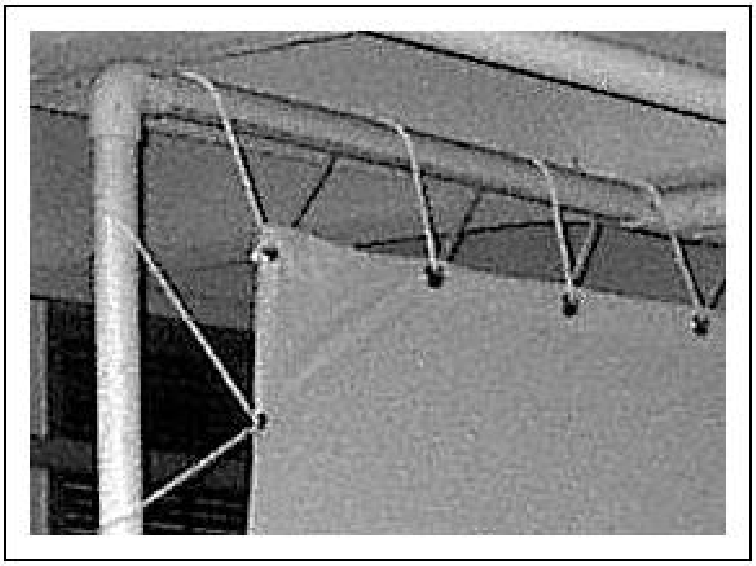
Tie the bottom of the seam to the bottom cross piece in exactly the same way. Make it tight, but not too tight, or the clamps may start to slip off the carpet seam binders.

Step 7



Take a 20-foot length of the nylon cord, tie one end to the frame (as shown to the right), and start looping along the top of one of the screens as shown. Work from the first available grommet near the inner corner of the frame towards the outside. Do not pull too hard. Keep the top edge of the screen some consistent distance from the top PVC pipe. At the same time, make sure that the bottom edge is at least an inch or two above the bottom edge of the frame.

Step 8



Keep threading the cord until you reach the top corner of the frame and keep on going down the outer edge, as shown to the right. Roping the outer corners is problematic -- it is difficult to do this without allowing the screen to sag a little, a problem we have not yet solved. (People setting up their own CaveUT implementations who find a solution are encouraged to let us know; we'll add it to the next revision of this documentation.)

Continue threading until you reach the middle of the outer edge, and stop there. Do not pull it tight yet.

Step 9

Thread the other screen, at the top, in exactly the same way.

Step 10

Thread the bottoms of the both screens in exactly the same way you did the top edges. The loose ends of the ropes for top and bottom should meet in about the middle of the outer edges.

Step 11

Thread one long piece of cord through the hand clamps behind the cental corner of the frame, as shown in the illustration beside **Step 3**. Start at the top and work your way to the bottom. Don't pull the cord too tight at this point, but don't let it hang in loops. Attach it at the top and bottom by wrapping it around one spur of each crosspiece a few times, but don't tie it off.

Step 12

Gently, but firmly, tension the top or bottom edge of one screen, whichever one has the most room between the edge of the screen and the frame. Start at the first loop of the rope, which goes though the first grommet nearest to the frame's central corner. Pull the slack out of that loop and get rid of the slack by pulling on the next loop. Then pull the following loop and so on. Keep going past the outer corner of the screen and go down (or up) along the outer edge of screen. Eventually, you will reach the end of cord. All the slack will now become more rope hanging off the end.

Secure it by tying a slip-knot or just wrapping it around the PVC pipe a few times.

Step 13

Tension the other side of the screen, the top or bottom, and work out to the outer edge, just as described in **Step 12**.

Step 14

Tension the other screen in exactly the same way.

Step 15

At this point, take a serious look at the inner corner of the screens. It should bow inward a bit, toward the viewer. This is not a problem; the bowing will be corrected in the next stop.

Take note of whether the screen is clean and consistent, and whether there are any puckers or wrinkles caught in it. If it is flawed in these ways, then you can fix it: Take all the clamps and the carpet seam binders off of the back of the seam, but let the carpet seam binders stay tied to the screen frame. Remove the cord that you had strung through the clamps. Now, rebuild the seam in much the way you did it in **Step 4**, starting from the top and work down. You will find that the process is easier this time, and you should get a good, straight edge.

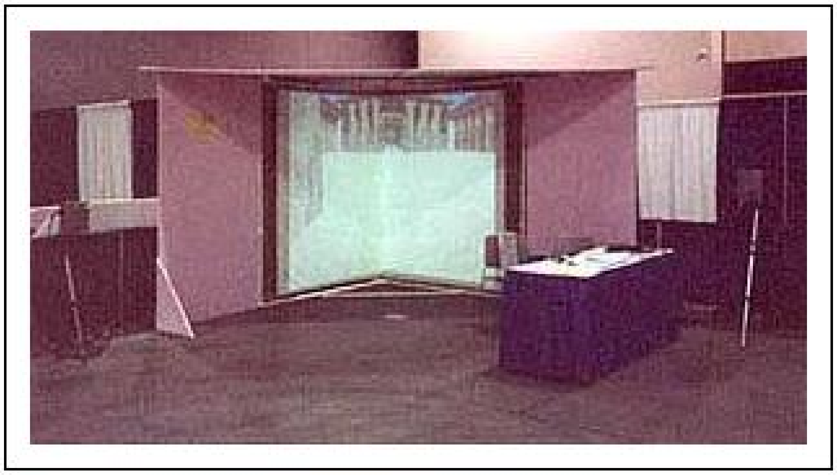
Step 16

Move along the edges of the screens, tightening the loops of rope as much as necessary to make the screen hang flat and to make the main corner clean and straight. You will find that you have to pull a bit more near the center of each side of each screen. This phenomenon will be familiar to anyone who stretched a canvas over the traditional wooden frame to make a painting. The PVC pipes in the frame will bow in a bit, which is normal and not problematic.

Done

Now the screens are finished.

Disposable Overstructures to Shade Portable Screens



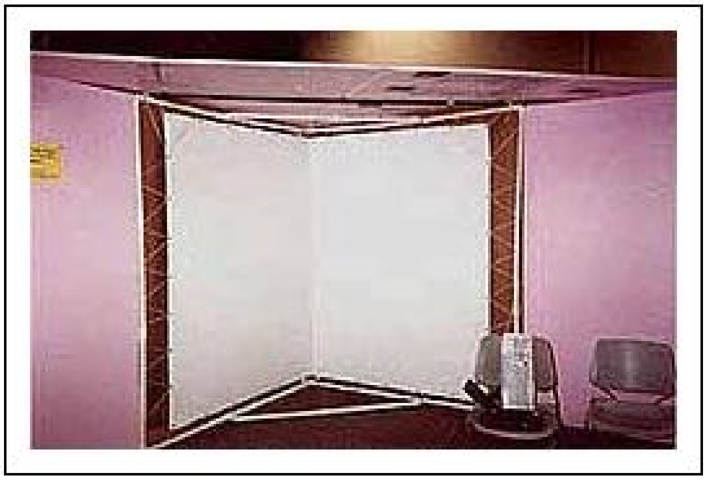
Obviously, portable screens are useful for CaveUT implementations meant to be moved from site to site, chiefly for purposes of demonstrations. It's not usually practical to carry oversized sunshades to shield the screens from bright lights, but it's similarly unrealistic to expect all venues to have the best ambient light for a CaveUT's purposes... and bright lights wash out the images projected on the CaveUT screens.

One such instance was CHI 2002, where the VCAVE was demonstrated. There, it was necessary to improvise a temporary overstructure to shade the screens from the bright lights in the convention hall.

About $100 at a large hardware/construction supplies store purchased enough 4'x8'x1" sheets of pink rigid foam insulation (in this case, Foamular® by Owens Corning). This sort of material is ideal for light improvised construction of this sort -- it's light, strong, cheap, stable, can be cut with a steak knife, and can be attached with packing tape and standard wood glue. (On the down side, it's also extremely *flammable*, making it unsuitable for permanent displays unless you treat it further with some sort of fire retardant.)

The roof of the structure constructed to shield the VCAVE from overhead lights was a right triangle (11.31' x 11.31' x 16') made from three sheets of Foamular® -- as shown here, the two corner pieces are both taken from one sheet. The edges were glued together and additionally taped with clear packing tape.

Once the pieces were taped together, they became remarkably stable. It was then possible to maneuver and cut them as if they were a single piece. (A tip to such construction: Packing tape will usually peel off with time, but not until it has given glue time to dry. For fast, improvised construction of this sort, it's convenient to glue the pieces together for their long-term stability, then immediately tape them for short-term stability.



As shown to the right, simple brown construction paper served to block light from coming through the lacings between the frame and the screens. (In this case, it was not necessary to use it behind the screens, as this particular screen material was already opaque.) In a similar situation, if your screen material is at all transparent and you are using front-projection, you must put something behind the screens to prevent light from leaking through the screens and washing out the projection image.

Finally, there is single black bar supporting the frame's roof, which which is barely visible in the illustration to the right. This as just a water-pipe laid on top of the PVC pipe structure and tied onto the frame with two bits of rope.

The whole thing went into the trash once CHI 2002 was done.

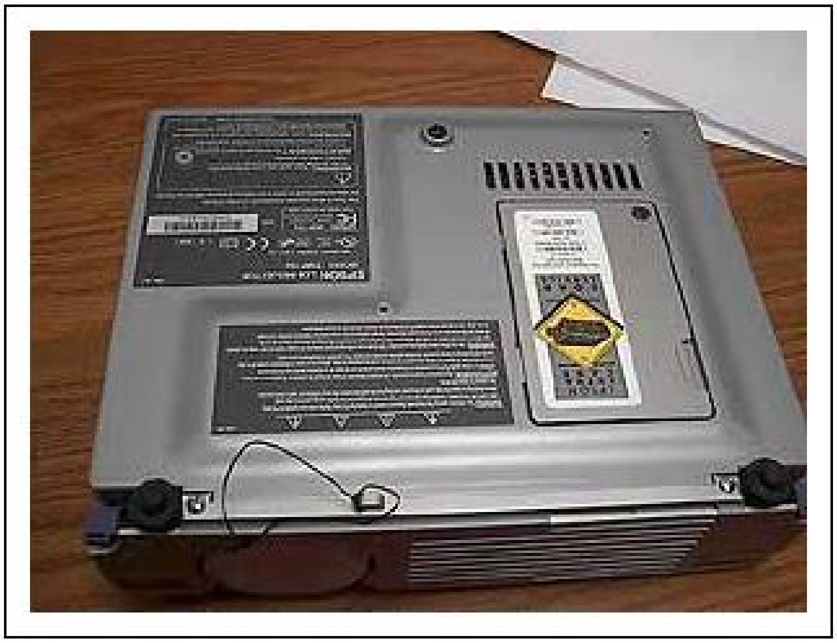
Building Projector Mounts

Earlier, we discussed the fact that most CaveUT implementations will require that mounting plates be fabricated for the projectors. Here are some guidelines for fabricating them.

First, some warnings:

- The plastic case of the projector is much softer than any metal bolt or screw, and the threaded holes in them are not likely to have any kind of metal lining -- they are just cut into the plastic case. This is not a problem if you are gentle, but you *must not* try to force the bolt in. Doing so can easily strip the threads in the bolt-hole.
- The bolt-holes may have a metric threading. If the required bolt size is not in the projector's specifications, it will be necessary to go to a good hardware store with the projector and experiment with metric screws until you find one that fits. Forcing a bolt with the nearest English system approximation of the threading into the hole will inevitably strip the bolt-hole's threads.
- Most importantly, however you mount or place a projector, *do not* cover up any of its air vents. This will definitely cause it to overheat, ruining the projector and costing thousands of dollars.

Step A



Make a plate of some flat material and cut it to match the dimensions of the side of the projector with the bolt holes. Hardboard and Plexiglas® are ideal, but any construction material that is sturdy, lightwait, and suitable for cutting will do.

If the plate covers up a vent on the projector, then cut an equal-sized hole in the plate to let the air through.

Step B

Drill a hole in the plate for each bolt-hole. Be sure to carefully measure the bolt-holes' relative locations and mark where you want to drill into the mounting plate. Guessing, rather than accurately measuring, is counterproductive.

Step C

Add hardware to the mounting plate which will allow it to attach to the tripod with a bolt that fits through the appropriate slot in the tripod's head. Any number of nuts or nut-and-washer combinations available at the hardware store should serve admirably.

Probably the most durable and effective approach, assuming ready availability of the appropriate tools, is to make the mounting plate out metal and weld a steel nut to the middle of it. Then it's a trivial matter to attach the plate to the tripod's face-plate with a steel bolt. Steel is best, as the entire weight of the projector will be resting on just the threads in the nut and the tip of the bolt.

It's also not a bad idea to put some space between the mounting plate and the projector by using longer screws and placing additional washers or nuts between the projector and the mounting plate. This allows permits use of a larger, sturdier nut for the point where the tripod bolt attaches to the mounting plate, and provides even more room for air-flow and cooling of the projector.

Step D

Attach the mounting plate to the projector. Be aware that the bolt-holes are not usually deep. If you cannot find bolts that are the right length, purchase bolts that are a little too long and use washers and/ or nuts as spacers.

Next: **Software and Downloads**

Previous: **Hardware and Software Requirements**

Start: **Back to Page Index**

Software and Downloads

Required Files

The following files are the minimum necessary software acquisitions and downloads needed to set up and run CaveUT.

(From Computer Game Retailers:)

- ***Unreal Tournament 2003***. An installation of CaveUT requires the purchase and installation of one commercial copy of *Unreal Tournament 2003* per client PC/screen in the CaveUT setup and an additional one for the server-PC.

(From Planet Jeff Site:)

- **ut2003-winpatch2225.exe**. This package contains the latest *Unreal Tournament 2003* code patch. It is a self-extracting archive that will install itself on your existing installation of *Unreal Tournament 2003*, so do not run it unless you have UT2003 installed.
- **CaveUT2003.zip**. This package contains the CaveUT Mutator and the executable for VRGL. Download it to any directory and de-archive it with WinZip or any other utility that handles .ZIP archives.

Currently, download of CaveUT is offered via http only. If you have any problems obtaining it, contact **Jeffrey Jacobson**.

Optional Files

- **VRGL Source Code**. From this page, download VRGL.zip. It has been compiled with MSVC++ 6.0, and other compilers should work.
- **CaveUT 1.2**. All the instructions and downloads you need for CaveUT on the old Unreal Tournament

Next: **VRGL**

Previous: **Hardware and Software Requirements**

Start: **Back to Page Index**

VRGL

Off-Axis Projections for OpenGL Applications

by Willem de Jonge

VRGL is a modified OpenGL libarary which makes it possible to use off-axis projections with *Unreal Tournament 2003*. In principle, VRGL is generic and can also be used with other OpenGL applications to get an off-axis frustum.

It is a custom OpenGL library that calls the original system OpenGL library for most of its work. It changes the projection of the application to a user-defined (by means of parameters in an .ini file) off-axis frustum.

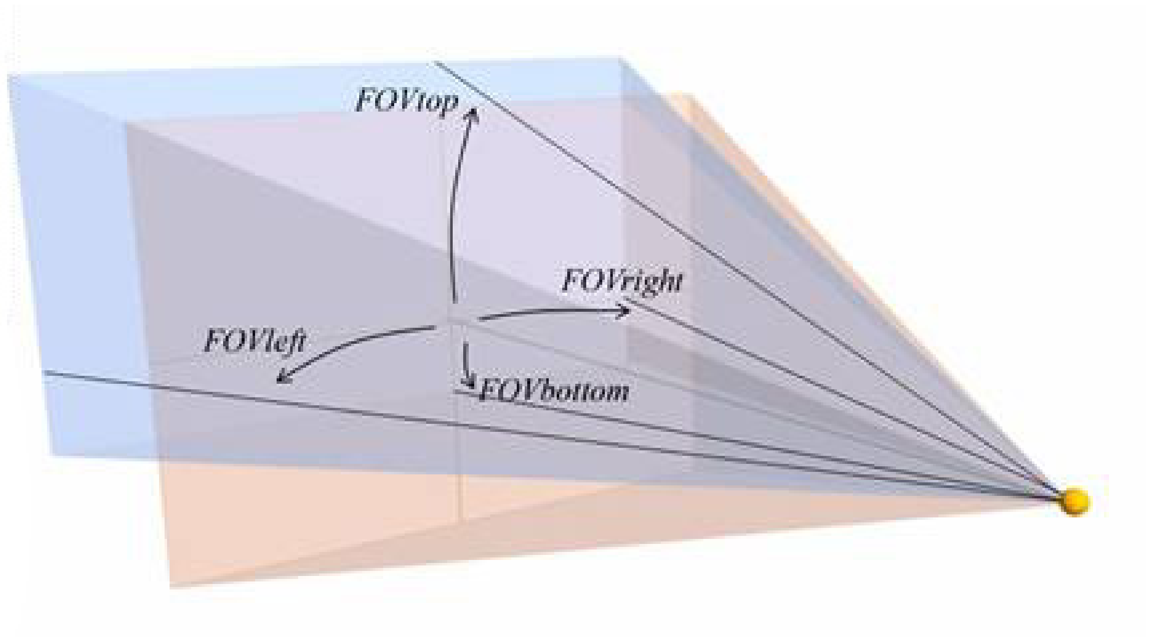
Note that while VRGL is a generic tool for off-axis projections it is developed for, and tested with UT2003. VRGL is used by **CaveUT2003** to allow UT2003 to work on multiscreen immersive displays. It is this package, part of the game itself, which synchronizes the rotation and position for all display clients. It calls VRGL to handle the perspective effects.

Configuration and usage

For an application to use the custom OpenGL library it should load this one instead of the system one, this is accomplished by placing the library in the same directory as the executable, this directory is searched before the system directory for any dll’s to load by Windows. For UT2003 this directory is `<install-dir>\System.`

The parameters for the off-axis projection are read from the file

`CaveUT.ini` in this same directory. Four parameters, *FOVleft*, *FOVright*, *FOVtop* and *FOVbottom* , are used to define the projection. These parameters are the angles for all sides of field-of-view in degrees relative to the view axis (perpendicular to the viewing plane through the center of projection).



If the part of the new off-axis is outside the original field-of-view it is possible that objects in that part aren't rendered because the application thinks they won't be visible and thus doesn't send them to OpenGL. In UT2003 this is fixed by setting the fov as large as needed to fit the whole off-axis frustum.

VRGL reads its parameters from the CaveUT.ini configuration file. The parameter *OffAxis* can be set to *Yes* or *No* to turn on or off the off-axis effect. VRGL ignores the other parameters, (Roll, Pitch,..., CaveFOV) which are used by the CaveUT2003 mutator.

Example CaveUT.ini file:

```
[CaveUT.Spectator]
CaveRoll=0.000000
CavePitch=0.000000
```

```
CaveYaw=0.000000
CaveOffsetX=0.000000
CaveOffsetY=0.000000
CaveOffsetZ=0.000000
CaveFOV=100.000000
OffAxis=No
FOVleft=-45
FOVright=45
FOVtop=36.87
FOVbottom=-36.87

[CaveUT.CaveUTInteraction]
RotateIncrement=0.250000
OffsetIncrement=0.500000
FOVIncrement=0.500000
```

Download

The source and executable are in: **VRGL.zip**.

The source code was originally compiled with MSVC++ 6.0, though it should be compatible for other compilers.

Inner workings

This section gives an overview how VRGL works; exact details can be found in the source code.

Basic operation of VRGL consists of the following:

- All calls that have an effect on the projection matrix are intercepted to maintain an exact copy of the intended projection matrix stack
- Whenever the projection matrix changes a user defined one is supplied to OpenGL, using the 4 FOV parameters
- Pass remaining calls 1:1 unchanged to the real OpenGL driver

A copy of the matrix stack is kept to ensure that the behavior custom OpenGL library appears the same as the real one to the application, ie stapling and/or querying the projection matrix works as expected. For non-perspective projection matrices the intended matrix is used directly and not the user defined one, the reason for this is a practical one: Even if you have a really odd user projection you can still 2D elements such as the game menus and console in UT2003.

To build the user defined projection matrix we need a near and a far plane besides the 4 FOV parameters. To ensure that behavior is as much as was intended by the application we extract the values for the near and far planes from the intended projection matrix.

To pass calls to OpenGL the custom library implements a jumptable to the real OpenGL functions is maintained. To fill this table the original OpenGL library is dynamically loaded and the table filled on initialization. We can’t link directly/statically to OpenGL as we are building the same library (as far as interfaces are concerned).

As this jumptable and the code to initialize is quite dump (for all the calls that are passed unchanged) we automatically generate is. This is done with the `genstub.pl` perl script, it takes for input OpenGL header files, gl.h (the mesa header file is used for this one) and wgl.h (specifically composed for this purpose), the latter containing the function declaration for the Windows specific calls. In addition to the header files two files containing function names are read:

- Autostubs.txt – functions for which automatic stub code should be generated
- Replaced.txt – functions that are implemented manually, ie calls that affect the projection matrix

From this the following is generated:

- Gljumptable.h – the class declaration of the jumptable, having a function pointer for every OpenGL call
- Gljumptable.cpp – code to initialize the jumptable
- Glstubs.cpp – implementation of the OpenGL functions that are

passed unchanged to OpenGL, ie these functions just call the matching function in the jumptable.
- Exports.def – this is only to keep VC from exporting mangled names for the exported OpenGL API

CopyLeft Notice

To make a long story short, VRGL is "BragWare": You can use it for anything you want, as long as you give us credit. If you do use it, please let *us* know so we can tell people about your excellent work.

Officially, VRGL is distributed under the **Lesser GNU Public Licence**. In summary, this means that VRGL is public domain and stays that way, whether you change it or not, and you may not in any way claim it as your own. If you make some software which calls VRGL or otherwise depends on it, you need to keep VRGL a distinct entity. The LGPL will *only* apply to the VRGL portion. You can even sell your software as a commercial product and just provide a free copy of VRGL as part of the distribution. For the legal details, read **LicenseVRGL.html**.

Next: **How to Install and Configure CaveUT**
Previous: **Software and Downloads**
Start: **Back to Page Index**

How to Install and Configure CaveUT

Intalling and then configuring CaveUT is a procedure not suited to the faint of heart, but methodically following the steps described below should see you through it with setup and sanity intact.

The numbered sections below show the major phases of the installation and configuration operations. They are:

Next: **I. Install Software on Each Client Computer**

Previous: **VRGL**

Start: **Back to Page Index**

I: Install Software on Each Client Computer

These instructions describe what must be done for each PC that produces the digital image for one screen of your multi-screen display. They presume that you have already downloaded all necessary files described on the **Downloads** page.

Step 1: Install *Unreal Tournament 2003*

This merely requires that you follow the instructions that came with the *Unreal Tournament 2003* download or disc.

Step 2: Install the Latest Code Patch

From whichever directory you saved it to, execute this file. It will find your UT2003 installation and modify it.

Step 3: Backup Specific Files

Installing CaveUT will change the following files in your C:/UT2003/system/ directory:

- *opengl32.dll*

It's a good idea to make copies of them before you get started. In the unlikely event that installation of CaveUT corrupts your UT installation, you can always repair it by bringing these files back.

Step 4: Extract CaveUT Files

Extract these files from CaveUT2003.zip and copy them into your C:/UT2003/system directory.

- *CaveUT.ini*
  This is the configuration file for CaveUT. It contains only default values, mostly zeros, which specify "no change" for the view screen. Leave this file unmodified for now. You will adjust these parameters after you get everything else working right.
- *CaveUT.u*
  This is both the code and the executable for the CaveUT Mutator. (A "mutator" for *Unreal Tournament*is special way to package changes to the game code for all versions of *Unreal Tournament*. For a general idea of how this works, read this somewhat out-of-date **tutorial**for the old *Unreal Tournament*.
- *CaveUT.int*
  This small config file tells UT2003 to load CaveUT.u. Do not modify it.
- *opengl32.dll*
  The is the executable for OffAxis, the modified OpenGL library which CaveUT depends upon for its advanced functions.
- *README.TXT*
  This document refers to this web page and includes change history notes and other notices.

Step 5: Edit UT2003/System/ut2003.ini

Make the following modifications to the ut2003.ini file:

Replace the line: "`ServerPackages=UTClassic`"
With this line: "`ServerPackages=CaveUT`"

Replace the line: "`AllowDownloads=True`"
With this line: "`AllowDownloads=False`"

You will notice three lines which look like this near the beginning of the file:

```
RenderDevice=D3DDrv.D3DRenderDevice
;RenderDevice=Engine.NullRenderDevice
;RenderDevice=OpenGLDrv.OpenGLRenderDevice
```

Put a semicolon in front of the first line and get rid of the semicolon in front of the third line, so they look like this:

```
;RenderDevice=D3DDrv.D3DRenderDevice
;RenderDevice=Engine.NullRenderDevice
RenderDevice=OpenGLDrv.OpenGLRenderDevice
```

When you save the file from your editor, be sure it writes the file in text-only mode. The "Notepad" editor in Windows is fairly safe and easy this way.

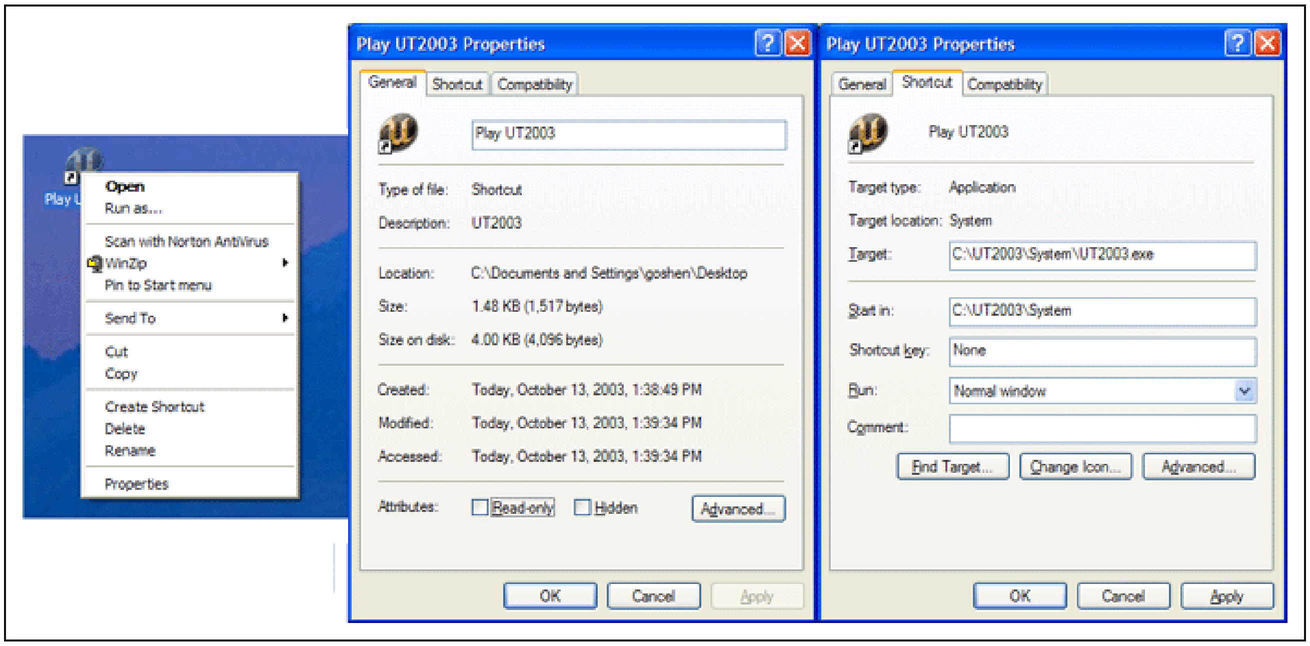
Step 6: Set Up a Connection Shortcut

Create a shortcut to UT2003.exe and put it on the client's desktop, in the Start menu directory, or wherever it best serves your needs.

Right-Click on the shortcut to get a pop-up menu.

Go to the bottom of that menu and select "properties". You will get a dialogue box with three tabs.

Select the "Shortcut" tab.



Play UT2003 Properties

Play UT2003

Type of file: Shortcut
Description: UT2003
Location: C:\Documents and Settings\goshen\Desktop
Size: 1.48 KB (1,517 bytes)
Size on disk: 4.00 KB (4,096 bytes)
Created: Today, October 13, 2003, 1:38:49 PM
Modified: Today, October 13, 2003, 1:39:34 PM
Accessed: Today, October 13, 2003, 1:39:34 PM
Attributes:

Play UT2003 Properties

Play UT2003

Target type: Application
Target location: System
Target: C:\UT2003\System\UT2003.exe
Start in: C:\UT2003\System
Shortcut key: None
Run: Normal window
Comment:

In the textbox "Target", you will see the full pathname to the UT2003 executable. Copy and paste (or type) a single space onto the end of the pathname and then the following text:

```
127.0.0.1?spectatoronly=true?quickstart=true
```

So the whole line would look like this, except that it doesn't all show in the textbox:

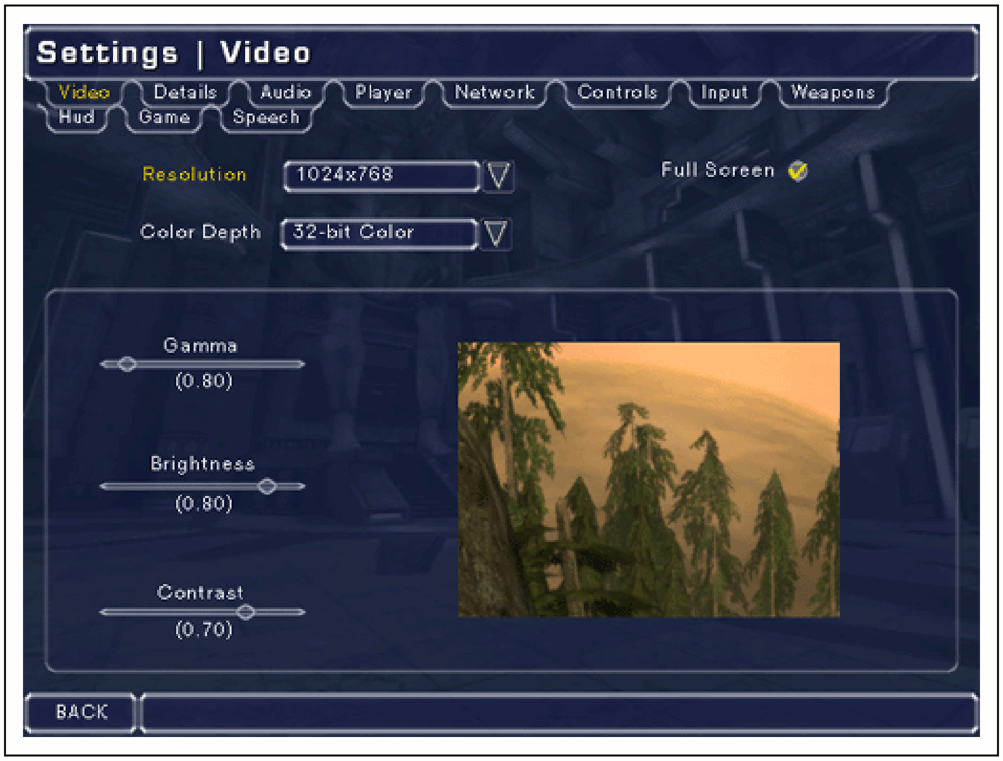
```
C:\UT2003\System\UT2003.exe 127.0.0.1?spectatoronly=true?quickstart=true
```

Replace the local IP address, 127.0.0.1, in the "Target" dialogue box with the IP address for your server machine. If you do not know what this means or how to use it, contact your network administrator.

Step 7: Set the Video Options

Launch UT2003 via its shortcut in the Start Menu. From the initial main menu, click on "Settings" to get the settings menu shown below. The tabs along the top let you change the screen to access different groups of settings. It usually defaults to the video settings, which is convenient.

Set the color depth to 32, or OpenGL will give you lots of nasty flashing polygons effects.



Resolution 1024x768
Full Screen ✓
Color Depth 32-bit Color
Gamma (0.80)
Brightness (0.80)
Contrast (0.70)

Set the screen resolution to whatever your projector (or whatever) can handle.

There are no hard and fast rules for further adjustment of the video options; you should experiment with the other settings until the video projection suits your tastes, preferences, and purposes.

Step 8: Make Sure the Sound Is Turned Off

In a typical CaveUT installation, you only want the server machine to be producing the sound effect. Otherwise you get a cacophony of sound effects, most of which are the same sound, but with each machine slightly out of phase with the server.

The simplest way to disable unwanted sound sources is simply not to plug in any speakers. You can also turn each client computer's sound off.

Otherwise, you can click on the "Audio" tab in the settings panel and set all the volume sliders to zero.

Next: **II. Install and Configure the Server**
Previous: **How to Install and Configure CaveUT**
Start: **Back to Page Index**

II: Install and Configure the Server

Step 0: Make sure your server PC has a static IP address.
If you have the server and all the clients pluged into an unmanaged hub or network switch, then you have to set their IP addresses manually, anyway. If you do not know how to handle PC networking, go find someone who does. It's simple, but not intuitive.

Steps 1-5: Follow Steps One Through Five of the Installation Instructions for a Client Machine (Previous Page)
But do not repeat Steps Six or Seven from those instructions; do not modify the file CaveUT.ini.

Step 6: (Optional) Edit User.ini
In the file User.ini, change the line reading `"Bob=0.????"` to `"Bob=0"`. This will turn off the bouncing effect players experience while navigating UT2003's "walk" mode. (The bouncing effect is not in time with any reasonable walking speed, nor is it the way we visually experience walking. For most uses of CaveUT, it serves best if disabled.)

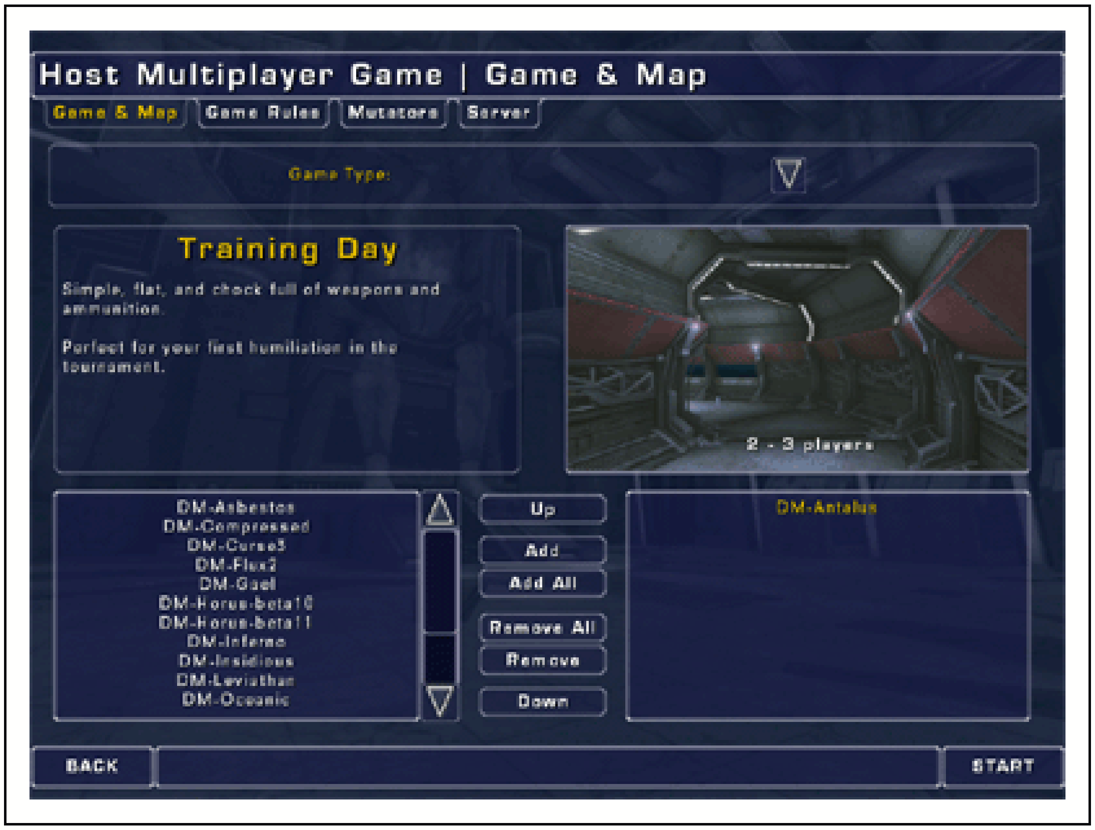
Step 7: Start UT2003
Double-click on the program's desktop icon (if it has one) or launch it from the "Start" menu.

Step 8: Prepare to Change the Video and Control Settings
From the main menu, select "Settings."

Step 9: Set the Video Options
Do this in exactly the same way as you did for the clients' setup, but this time it is only for the CaveUT operator's viewing pleasure -- the server machine will essetentially be the control panel.
A helpful trick: You can often get away with the server PC being slower than the client machines. Just set the screen resolution and color depth to mininimum values so the game runs quickly.

Step 10: Set the Control Options
Click on the "Controls" tab to reveal the settings for keyboard, mouse,

and/or joystick controls. At the moment you should probably leave these alone, but you will need to come back to this screen later to customize the controls to your liking.
To get out of the settings menus, click the "back" button on the lower left corner of the screen.



Training Day

Simple, flat, and chock full of weapons and ammunition.

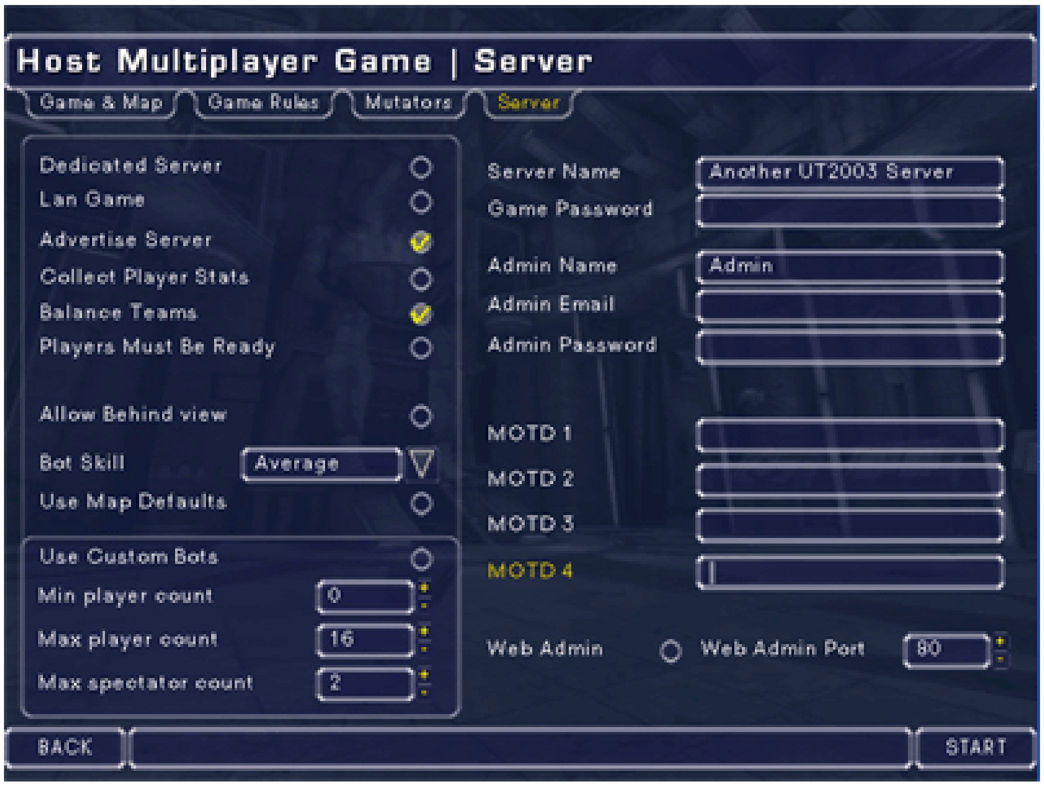
Perfect for your first humiliation in the tournament.

2 - 3 players

DM-Asbestos
DM-Compressed
DM-Curse3
DM-Flux2
DM-Gael
DM-Horus-beta10
DM-Horus-beta11
DM-Inferno
DM-Insidious
DM-Leviathan
DM-Oceanic

DM-Antalus

Step 11: Setup for a Multi-Player Game
You should see a screen like the one to the right.
The bottom portion of this screen is used to select the virtual world you will be using. The default is DM-Antalus, which is a visually appealing setting whose naturalistic shapes tend to cover up flaws in the display. When you start calibrating your CaveUT-based display, you should switch to a more demanding virtual world, such as DM-TrainingDay or DM-Serpentine (calibrating more difficult worlds ensures that your CaveUT implementation is up to the demands of any virtual setting).



Host Multiplayer Game | Server

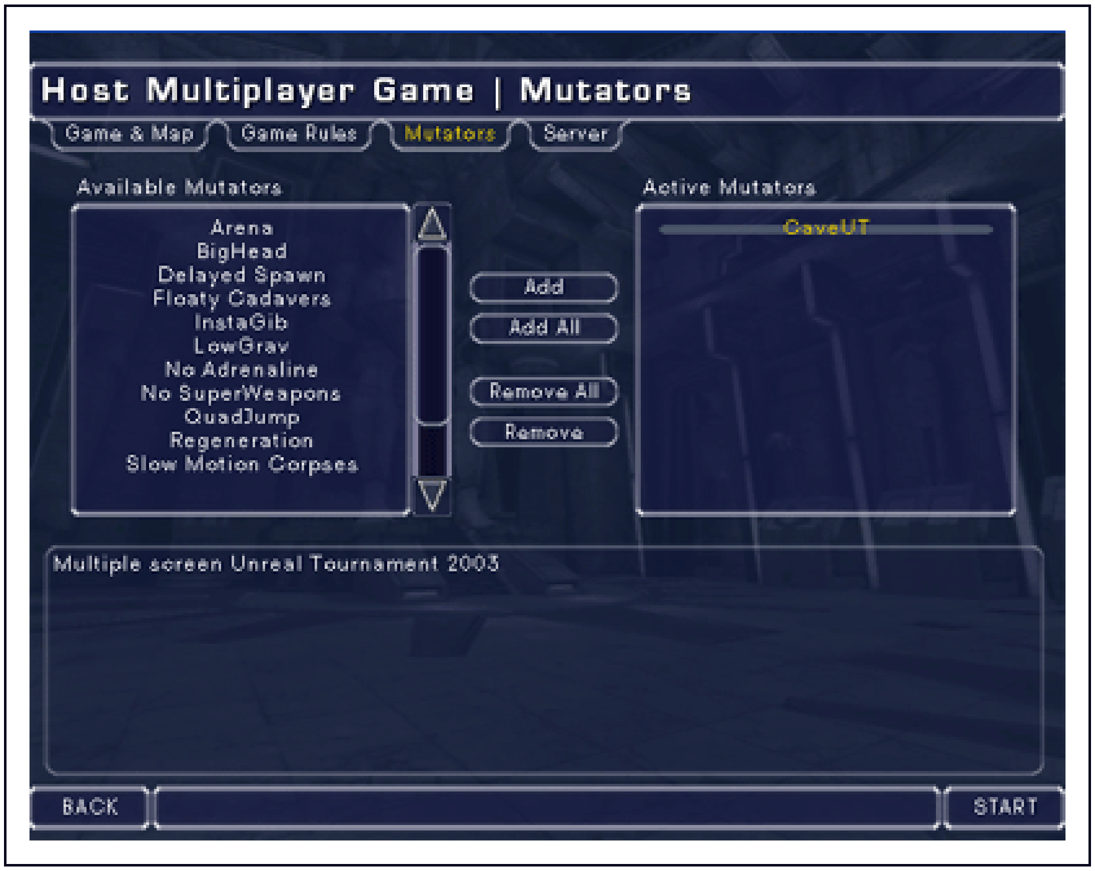
Dedicated Server	○
Lan Game	○
Advertise Server	✓
Collect Player Stats	○
Balance Teams	✓
Players Must Be Ready	○
Allow Behind view	○
Bot Skill	Average
Use Map Defaults	○
Use Custom Bots	○
Min player count	0
Max player count	16
Max spectator count	2

Server Name	Another UT2003 Server
Game Password	
Admin Name	Admin
Admin Email	
Admin Password	
MOTD 1	
MOTD 2	
MOTD 3	
MOTD 4	
Web Admin	○
Web Admin Port	80

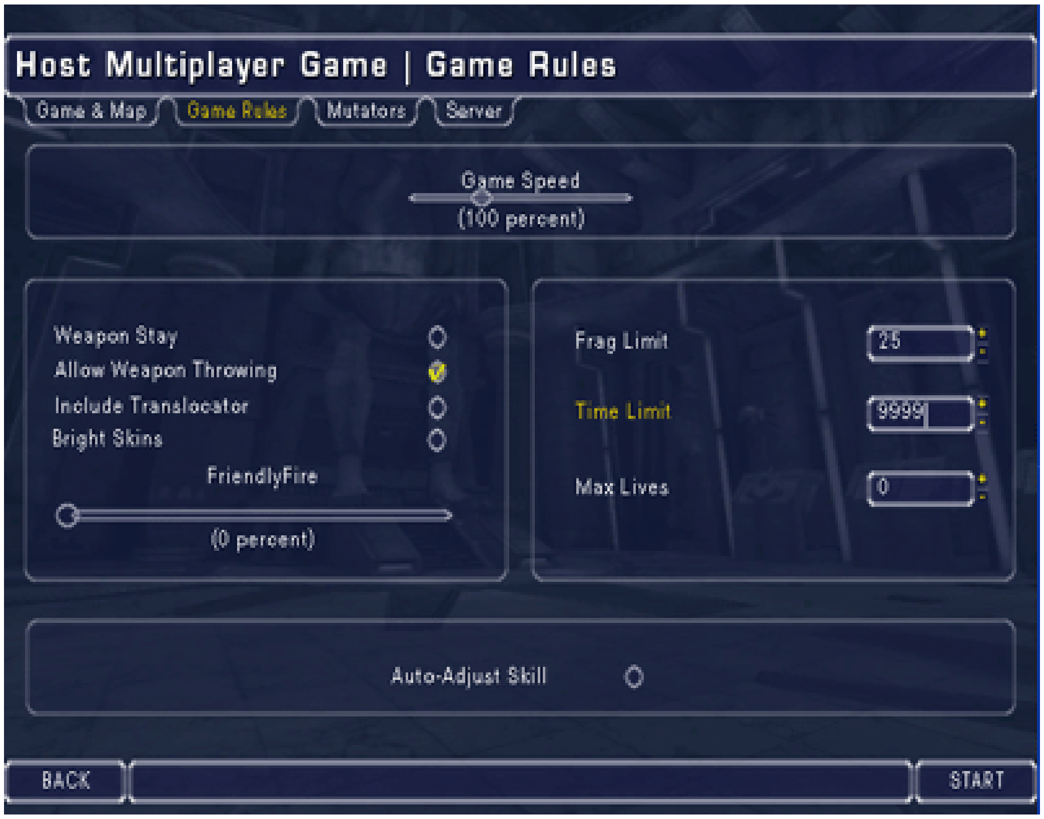
Step 12: Click On the "Server" Tab
You should now see the screen to the right.
Go to the lower-left area of the screen and increase the "Max Spectator Count" to some high number, like 20. You only need as many spectators as you have screens on your display, but having a higher max affects nothing and might save you trouble some day.

Example
If a the UT2003 game server is already supporting the maximum number of spectators and another tries to connect, the would-be-spectator's display will offer up the cryptic error message "Connection Refused by Server" and do nothing else.
This could happen, for instance, if someone were to set up a four-walled display with CaveUT but first test it on only two wall. He leaves the maximum number of spectators at 2 for the test, because that's all he needs. When the test is successfully completed, he sets up the other two walls but forgets to set max-spectators to 4. Once the first and second screens' clients are connected to the server, the server will refuse the third and forth until the max-spectators value is set higher and the server is restarted.

(For some strange reason, changing the "MaxSpectators" variable in the "UT2003.ini" does not work.)
In addition, you should click on the "Lan Game" checkbox, to improve your network performance.



Step 13: Click On the "Mutators" Tab
Select the CaveUT mutator by clicking on "CaveUT" on the list in the left side box, then clicking on the "Add" button in the middle. When you are done, the screen should look as it does, below. At this stage, don't use any other mutators. Most should work with CaveUT, but some may conflict. Get your CaveUT installation working and stable before you try other mutators.



Game Speed
(100 percent)

Weapon Stay
Allow Weapon Throwing ✓
Include Translocator
Bright Skins
FriendlyFire
(0 percent)

Frag Limit 25
Time Limit 9999
Max Lives 0

Auto-Adjust Skill

Step 14: Set the Time Limit for the Game
Click on the "Game Rules" tab, and set the "Time Limit" to the highest number you can, which is 9999. That is the number of minutes the server will allow the game to run before it kills the one server player and restarts the game.

The installation is now complete. At this point, you can start the server by pressing the "start" button in the lower-right hand corner, or start it up later, per the instructions on tne next page.

Next: **III. Start CaveUT for the First Time**
Previous: **I. Install Software on Each Client Computer**
Start: **Back to Page Index**

III: Start CaveUT For the First Time

Step 1: Start a Multiplayer Game On the Server Machine

Choose a level (virtual world) to begin with. Be sure the enter the game as a player, not a spectator.

When the virtual world becomes visible, press the button designated in the Controls menu/screen for "fire," meaning "fire your current weapon." You must do this for your player to enter the game, and the CaveUT Mutator on each client needs that player to function properly. There is an approximately thirty-second wait before the player appears. When s/he does, a swirling light effect will briefly appear on the screen and there may be a sound effect.

Step 2: Connect Each Client to the Server

On each client PC, double-click the specialized shortcut to UT2003 you built earlier. The game should flash a brief message indicating that it is connecting to the server, then spend up to a minute loading its copy of the level now in use on the server. For this to work, the level in question has to be installed on both the server and client machines. While you are testing your installation, it's a good idea to use the basic levels that come with the game.

Step 3: Press the Fire Button on Each Client

This will change the client's view to be co-located (in the virtual world) with the view of the server player. At that moment, the rotation and offset effects from CaveUT will begin. The perspective effects should already be visible. If the server player is not in the game yet, this won't work. Just wait a few seconds until s/he is.

Clicking the fire button actually shifts the client's view from one player or bot to another. If there is only one player in the game (on the server), the fire button effectively toggles between the CaveUT view from the player and a non-useful flying view.

If there will be more players and/or bots in the game, its helpful if the player on the server machine enters the game first. That way its view is the first on the list, so the first fire button click will give you the server player's view. Otherwise, you have to keep pressing the fire button until the client shows the correct view. This will also happen if you accidentally skip past the server view.

Step 4: Test Movement and Synchronization

On the server machine, as the game's only player, run around a bit in the level. All of the client views should change in tandem providing a rougly integrated image across all your screens. If movement displayed on the server's monitor is jerkey, try reducing the screen resolution, color depth, etc., to improve performance. If the server is quick and the clients displays are jerkey, you probably need to get them better video cards. If you can play UT2003 on a client, as a stand-alone machine at the same video resolution settings, and you get good performance, then it's a networking problem. If you are using a wireless network hub, get rid of it. Strining the network cables may be a chore, but they are much more reliable.

Most likely, everything will work fine, except that the projected images from the client machines don't quite line up. The following sections will show you how to compute the correct settings for each client and fine tune everything.

Next: **IV. Read This: Perspective**

Previous: **II. Install and Configure the Server**

Start: **Back to Page Index**

IV: Read This: Perspective

In western culture, ever since the Renaissance, paintings have used the technique of **perspective** to create an illusion of depth.

In the most simple implementation of artistic perspective, the primary lines in the composition all converge to a single point on the canvas; that location is called "the vanishing point."

The same effect is widely used in photography, film, and computer imagery. This page will relate artistic perspective to player view in CaveUT in a simplified fashion; you are encouraged to read more on perspective in textbooks for art, computer graphics, human perception, and mathematics.

Coordinating Off-Axis Projections in a Multiscreen Display

Think of UT2003 displaying on a single screen in a fashion consistent with most first-person computer or video games. The vanishing point of the image is co-located with the physical center of the screen. A first-person view of a virtual world presents the screen as a moveable window onto the world. Ideally, the perspective of the presented image is the same as if you are looking through a window onto a real scene.

The general situation is illustrated here: In Figure One, the viewing **frustum** is simply a way to describe what the viewer can see through a window. With a painting or photograph or video game, the frustum extends from the eye, through physical space, and into virtual space. The illusion of depth is correct, but (usually) only so long as a perpendicular drawn from the plane of the image to the observer's eye is over the vanishing point in the display.

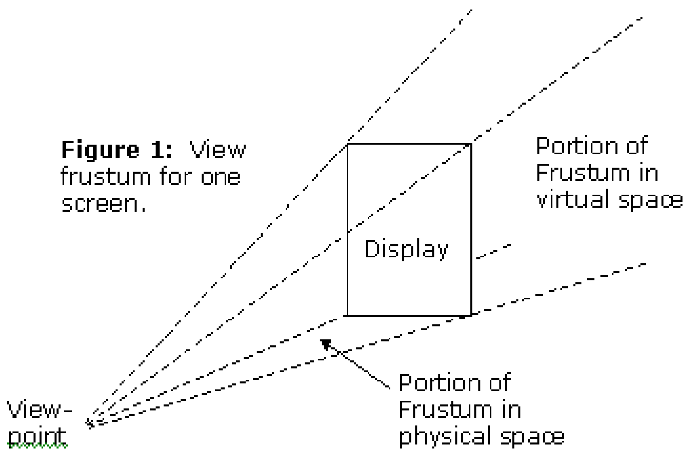
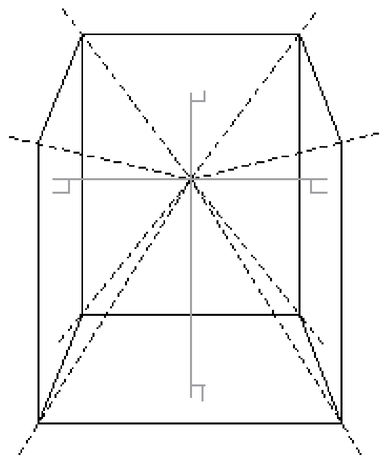


Figure 1: View frustum for one screen.
Portion of Frustum in virtual space
Display
View-point
Portion of Frustum in physical space

For UT2003, this means that you have to be looking straight at the center of the screen for the depth illusion to be mathematically correct. Most first-person shooter games are structured this way.



The human brain is very agile, and will allow a viewer to comfortably view a movie/photo/painting/video-game screen from a considerable angle, far off the ideal viewing location and direction. However, a normal person's visual centers can only do so with an image that reside entirely on one screen.

The idea behind CaveUT is to array several screens around the viewer to create an immersive display, as shown in the figure on the left.

In this schematic, the viewing frustum associated with each screen is shown in dashed lines. (The floor is also a screen) Each frustum begins with the viewer's eye, occupies an area in physical space shaped roughly like a pyramid up to the projection screen, then continues through virtual space to an imaginary infinity. (The perpendicular line from the viewer's eye to each screen is shown in light gray.) Where the perpendicular intersects the plane of the screen is the vanishing point of the image shown on the screen.

It is important to note that all four viewing pieces fit together like a simple 3D puzzle. This allows the viewer to look in any direction and see all objects in the virtual scene in correct perspective. In fact, for any direction that the viewer looks, s/he will enjoy a proper perspective view. Instead of a cave, you could use four paintings or photographs to achieve the same effect, but for one static scene only.

As you can see in the schematic above, all four of the view frustums have a common apex, which is where the viewer's "eye" is supposed to be. (In practice, the player does not have to be so precise as to place his eye there -- just putting one's head in the vicinity is close enough.) From now on we will refer to that location as the **sweet spot**, which is a term used by audiophiles. It describes the best location to listen to sound produced by particular configuration of a sound system in some space. Each screen has a sweet spot for the eye to view it.

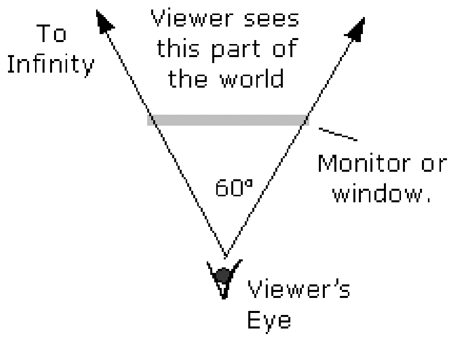
Think of CaveUT as a tool that can move the sweet spot for a particular screen by changing the size and shape of its view frustum. Whatever the arrangement of the view screens for your display, the sweet spots for screens all of them must be in the same place. Also, the view frustums must fit together exactly -- allowing no overlaps or space between them.

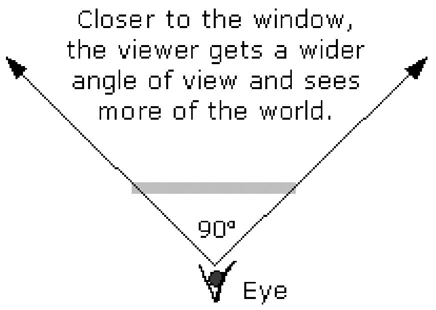
Theoretically, someone could use CaveUT to orient lots of view screens at all sorts of weird angles to the viewer, and each one would show part of a consistent view of the virtual world.

Perspective Correction

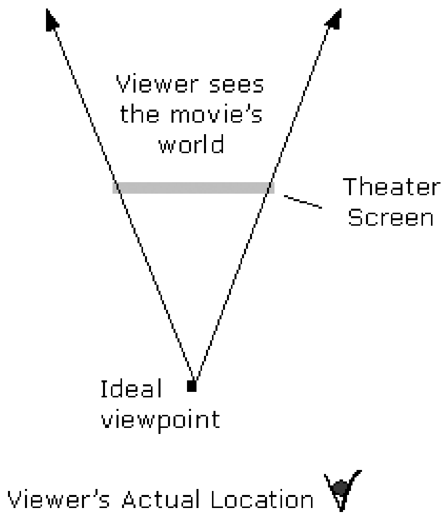
Think of each view screen in a Cave arrangement as a window into the virtual world. At its simplest, it's like looking though the center of a real window and keeping your eye fixed a certain distance from the screen.

As you can see in the drawing to the right, the distance of your eye from a real window governs how big a slice of the world you can see. That's why people move closer to a window when they want to look outside.



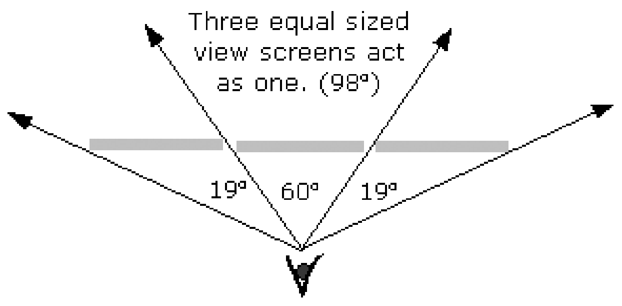

Closer to the window, the viewer gets a wider angle of view and sees more of the world.
90°
Eye

Unlike a real window, the view on a typical monitor screen does not change when the viewer moves. (Actually, there is specialized hardware and software which make this possible, but that's not relevant to this explanation.) Still, the viewer knows what s/he is seeing is just an image and the brain accepts it as conditionally three-dimensional. That is why someone can be far off of the ideal viewpoint in a movie theater or in front of a television and yet easily make sense of the show.

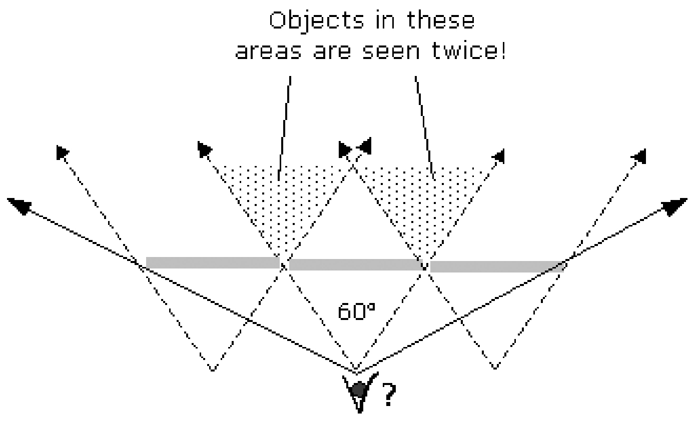


All of the low-cost multi-screen display solutions (like the Matroxx Parhila) have one thing in common: The resulting display is FLAT. In effect, they simulate one big window onto the virtual world.

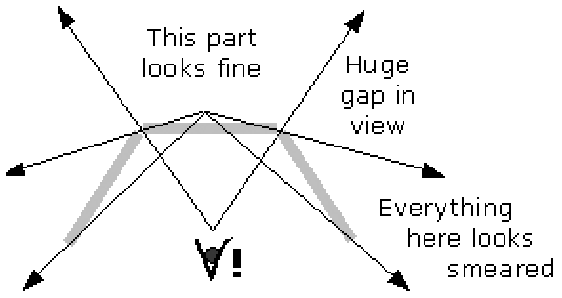
Note how lopsided the views (the "view frustums") are for the right and left screen.

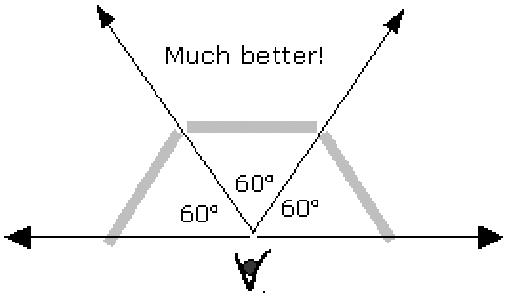


Note that if all the screens had the standard perspective correction, their view frustums would overlap disastrously, like this:



Of course, simple monitors could be used to physically display side views, but this requires more than just a physical rearrangement of the screens. For example, in a three-screen flat display (i.e., the Matroxx Parhela), each of the screens shows what is in front of the viewer. Just turning the monitors inward won't change that.

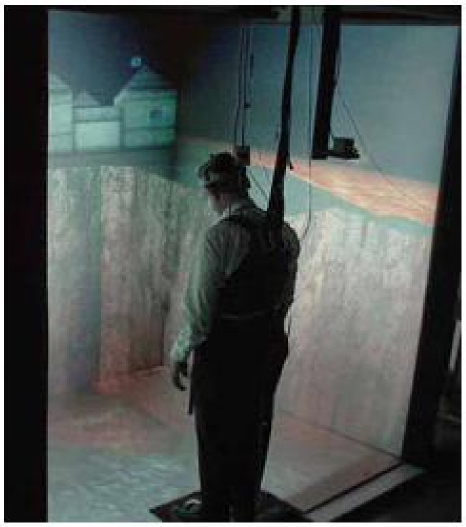


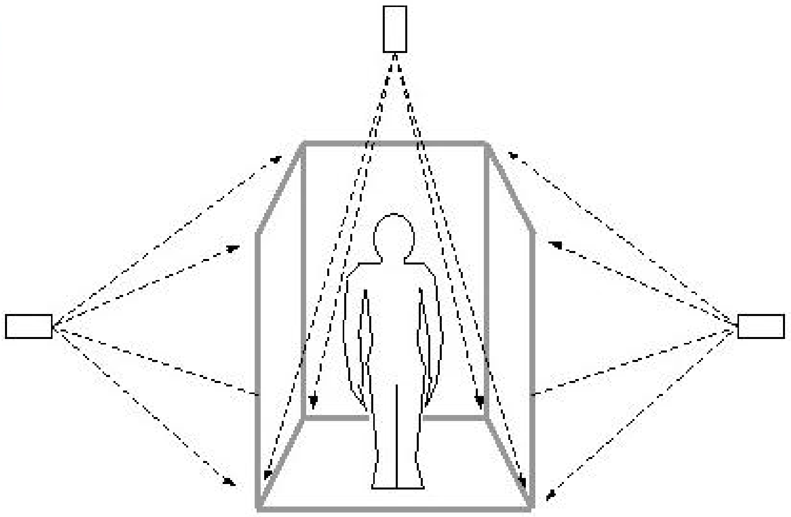


The drawing on the left shows a three-monitor flat display with the monitors turned inward.

The drawing on the right shows what happens when the software has been changed to show whatever is to the right of the viewer in the right screen and whatever is to the left on the left screen. With the proper perspective correction, the view frustums will fit together like puzzle pieces and the user will enjoys seamless, integrated view.

While large, flat-panel displays are ideal for television viewing, they lack a certain immersion which a surround view provides. For example, seeing a landscape in a simulator with a wraparound display, like the BNAVE, shown below, would allow the user to enjoy a much wider view than any flat screen.





Next: **V. Configure View Rotations and Offsets**
Previous: **III. Start CaveUT for the First Time**
Start: **Back to Page Index**

V: Configure View Rotations and Offsets

Below are the contents of the default CaveUT.ini file:

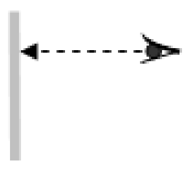
```
[CaveUT.Spectator]
CaveRoll=0.000000
CavePitch=0.000000
CaveYaw=0.000000
CaveOffsetX=0.000000
CaveOffsetY=0.000000
CaveOffsetZ=0.000000
CaveFOV=100.000000
OffAxis=No
FOVleft=-45
FOVright=45
FOVtop=36.87
FOVbottom=-36.87

[CaveUT.CaveUTInteraction]
RotateIncrement=0.250000
OffsetIncrement=0.500000
FOVIncrement=0.500000
```

For the moment, we are only concerned with four parameters: CaveRoll, CavePitch, CaveYaw and CaveFOV. (They are shown here in bold face -- they are not bolded in the actual cave.ini file.) Follow these steps:

Step 1: Place the Screens

Decide where you want to place the screens of your immersive display. You can have up to 31 screens, with each one arraigned in any orientation to the viewer.

Before physically setting up the screens, first **make a diagram!** Draw it out carefully, with all the important angles and sight-lines specified. You will have to use a fair bit of trigonometry as you go. If you are new to it, or need a review, this **tutorial** is among the best available online.



The term **"screen"**, as it is used in this document, refers to the portion of the physical display that is actually showing pixels generated by the PC's video card. Almost always, there is some extra projection screen or monitor material around the edges of computer driven display.

For an immersive display, it is usually best to have as much horizontal display area as possible and to balance the left and right halves of it. That way, the viewer sees as much (horzontal view) in the left eye as in the right eye.

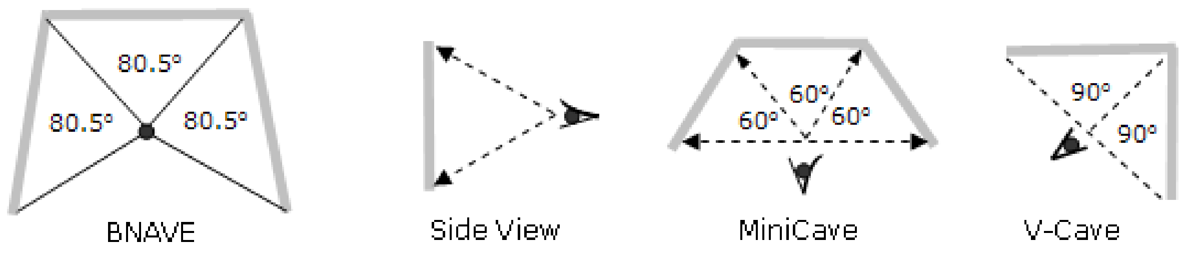
However, humans have more vertical range *down* than they do *up*. So when you arrange your screens, it's usually best to sacrifice some display area at the top for more display on the bottom. The image on the left illustrates this for one screen.

Extending this idea, it's generally better to have a floor screen than a ceiling screen.

Step 2: Find the Default Sweet Spot

While multiscreen immersive displays can be irregularly shaped, the vast majority have symmetrical designs, because they are easier to build and easier to work with. CaveUT naturally supports symmetric and asymmetric (irregular) displays equally well.

Symmetrical displays have a useful property: There is a single symmetric view frustum such that when all the screens use it, all the screens' frustums will meet at a common point -- the default sweet spot.



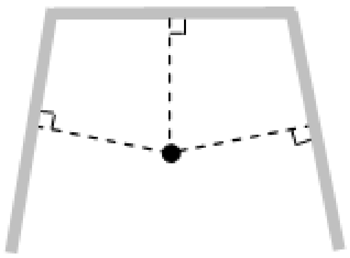
Three examples are shown on the right: the BNAVE (ignoring its floor), MiniCave, and V-Cave. The side view illustrates that the viewer's eye is level with the midpoint of each screen for each display.

The default sweet spot is the easiest one to configure and is adequate for many immersive displays, such as the MiniCave. You can calculate it while working with just the view rotations and the `CaveFOV` parameters in cave.ini.

If the sweet spot needs to be elsewhere, it's still best to get everything working first for the default sweet spot.

The procedure for finding the sweet spot is quite simple:

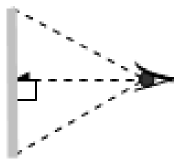
1. Graph or sketch where the screens are or will be.
2. Draw a perpendicular line from the center of each screen.
3. If the display is symmetrical, all of the perpendiculars will intersect at a single point, which is the default sweet spot.

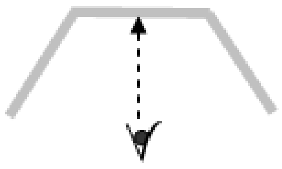


The diagram on the left shows where the sweet spot can be found for the BNAVE in an overhead view. Since all the screens are the same height, the vertical location of the sweet spot must be half the length of a screen up from the surface the BNAVE is resting on.

For more complex displays, it is best to use a 3D modeling program to "draw" your display in three dimensions before using this procedure.

Step 3: Decide Where to Put the "Control Point"





Even a single screen display has a sweet spot, and in a normal UT2003 game it is always located directly in front of the center of the screen, as shown on the left.

As we'll explain in Step V, below, the distance from the screen is determined by the field-of-view (FOV) parameter. Notice that the viewer's the line of sight goes directly from the sweet spot to the exact center of the screen. It determines the direction of pure forward motion and the direction the player's weapon weapon fires when the player shoots it.

From now on, we will call the point on the screen where the line of sight intersects the screen the **Control Point**.

By extension, a multiscreen CaveUT display also has a control point, usually located on the center screen, as with the BNAVE and the MiniCave, depicted in the image on the right above.

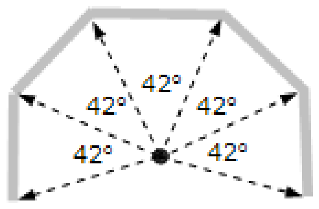
For your display, you need to decide where you want the Control Point to be. At this stage, when you are using the default sweet spot, the view frustums have to be symmetrical, therefore, the control point has to be in the center of one of the screens. So this exercise is really about choosing one of the screens, presumably the center screen.

While all symmetrical displays have a default sweet spot, some asymmetrical displays do also. The BNAVE, with its floor screen active, is an example. As long as you can draw a perpendicular from the exact center of each screen a common point of intersection, then the multi-screen display has a default sweet spot.

Step 4: Make Note of the "Line of Sight"

The "line of sight" starts at the sweet spot, goes through the control point, and continues onward into the virtual space. In that sense it is like the view frustum. Discussions later on this page require calculations based on the line of sight.

Step 5: Determine the Field of View (FOV)



For each screen, you need to figure out what its horizontal FOV angle should be, so the apex of its view frustum will be located at the sweet spot. The best way to do this is go back to the drawings you used to determine the default sweet spot. Then draw a line from the sweet spot to the edge of each screen. For each screen, calculate the angle between the two lines going from the sweet spot to its two edges, with **trigonometry**. All you need is the width of the screen (assuming an overhead view) and the distance from the sweet spot to the screen.

While CaveUT does all sorts of things to the computer's display, the engine for the UT2003 game on each computer has no idea all this is going on. It still thinks that it is generating a view to be shown on an ordinary monitor sitting on someone's desk. For this discussion, let's call the width of a normal unmodified UT2003 display the "long axis" (because the vast majority of computer displays are wider than they are tall), with the height of the screen the "short axis."

Now, one more term: the "aspect ratio" of a display is just the ratio of its height and its width, by convention (width / height). In our new terms, that would be the (long-axis / short-axis). Most commercial displays have a 4/3 aspect ratio (for example, 800x600 or 1024x768).

We will need these new terms because determining the right FOV gets a bit more complicated when the screens are sideways, as they are in the MiniCave, the V-Cave and the BNAVE (except the floor.) The `CaveFOV` parameter in cave.ini allows you to set the game's generic FOV setting. This allows you to set the FOV of the long-axis, at which point the game will calculate the FOV of the short axis to be just right so the display's aspect ratio stays the same.

For example, setting the long-axis FOV to 90 degrees will automatically set the short-axis to 53.13 degrees.

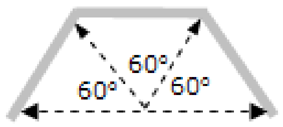
This formula shows the relationship between the long-axis and short-axis FOVs:

```
tan( L ) = A * tan( S )        Where:

L = ( Long Axis FOV ) / 2
S = ( Short Axis FOV ) / 2
A = ( Long Axis Length ) / ( Short Axis Length )
```

From this, you can calculate the short-axis FOV from the long-axis FOV and vice-versa:

```
S = atan( 1/A * tan(L) )         L = atan( A * tan( S ) )
```

For example, the diagram on the right depicts an overhead view of the MiniCave. Because the screens are turned sideways, the 60-degree FOVs in the drawing are short-axis FOVs. So what does the long-axis FOV have to be so we can get a 60 degree short-axis FOV?

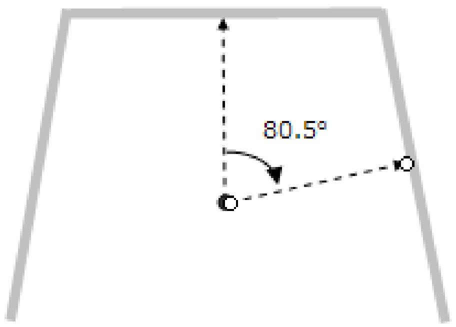


```
S = ( Short Axis FOV ) / 2 = 60/2 = 30 degrees

A = ( Long Axis Length ) / ( Short Axis Length ) = 1024/768
= 4/3

L = atan( A * tan( S ) ) = atan( (4/3) * tan(30) ) = 37.59
degrees

( Long Axis FOV ) = 2 * L = 2 * 37.59 = 75.18 degrees.
```

Set the `CaveFOV` parameter in cave.ini to 75.18 degrees to get a short-axis-FOV of 60 degrees.

Step 6: For Each Screen, Determine the Roll, Pitch and Yaw

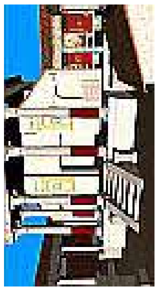
If you have a screen directly in front of the user, as with the BNAVE and the mini-cave, keep its Roll, Pitch and Yaw values zero. The screen is already where it needs to be. Each change of the other screens' views will be created by starting with the (virtual) front view and rotating it until it matches the its intended physical screen.



For example, in the BNAVE, the image on the right screen view would start out being the same as on the front screen. The user would rotate the view frustum *around the default sweet spot* by +80.5 degrees, as shown in the diagram on the right. A negative value, like -80.5, turns the view to the left. You are actually rotating the virtual line of sight until it intersects with the center of the side screen.

Set the Pitch (up-down) in the same way as the Yaw. For example, the floor screen image in the BNAVE is produced by setting the pitch to -90 degrees. Alternatively, setting it to +270 would have the same effect.

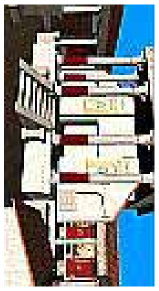
Set the Roll. This will rotate the entire scene, on the flat screen, around the control point. This is especially handy when you want an individual screen to be taller than it is wide. All of the vertical screens in the BNAVE, the mini-cave, and the V-Cave are this way. Positive numbers rotate the scene clockwise, negative numbers rotate it counter-clockwise.



For example,

```
CaveRoll=90.0
```

will turn the scene sideways, as shown on the right. Or, you can rotate the image counter-clockwise with:



```
  CaveRoll=-90.0
```

which will also turn the scene sideways, but in the way shown on the left.

Step 6: For Each Screen, Set the cave.ini Values

Open cave.ini in a text editor and change the values for the `CaveRoll`, `CavePitch`, `CaveYaw`, and `CaveFOV` parameters to the values you determined. All angles are measured in decimal degrees.

When you are doing this for the first time, it's worthwhile to play around with these values, just to get a feel for what the changes look like. You will be amazed at how much difference a single degree can make. Save the file and the changes will be visible the next time you use that client to connect to the server.

Final note: be sure to use a text editor that will write the cave.ini file in text-only form. Notepad is good for this.

Step 7: Fine Tune the View Rotations

At this point, you should have a single contiguous view of the virtual space. If you eye is not at the sweet spot, things on-screen will look bent where the screens meet, but that's normal. The main thing to look for at this point is how well the images line up where the screens meet. They will probably be off a bit because of small, unavoidable irregularities in the physical setup.

You can fine-tune the Roll-Pitch-Yaw values on any individual screen from the keyboard for that machine. To do this, start CaveUT/UT2003 on your server and client machines, and be sure to click the clients' fire buttons. **The help menu will only appear on a client machine when movement on its display is locked to the server's controls.** In other words, you have to have CaveUT fully engaged and working, otherwise the debug menu simply won't appear.

Debug keyboard commands are:

Shift+h	Show debug menu
Ctrl+h	Hide debug menu
Shift+a	Add increment offset X
Ctrl+a	Subtract increment offset X

Shift+s	Add increment offset Y
Ctrl+s	Subtract increment offset Y
Shift+d	Add increment offset Z
Ctrl+d	Subtract increment offset Z
Shift+z	Add increment rotation pitch
Ctrl+z	Subtract increment rotation pitch
Shift+x	Add increment rotation roll
Ctrl+z	Subtract increment rotation roll
Shift+c	Add increment offset yawchanged
Ctrl+c	Subtract increment offset yaw

The first two commands just toggle the debug menu on and off. The third set allow you to add or subtract a numeric value called "increment" to/from Roll Pitch or Yaw with a single keystroke. Press the key more times to get a bigger change. Play around with these to get a feel for what the changes look like.

The second block of commands allows you to add a permanent offset to the client's viewpoint with respect to the player's view on the server.

The X, Y and Z refer to the coordinate axies of the game. The X axis governs left-to-right movement. The Y axis governs up-and-down movement. The Z axis is depth, or forward-back.

Try to avoid using the offset adjustments, because they can be problematic. First, if you have even a small offset, it is possible to put the view though a wall just by pressing the server-player up against that wall. Second, the apparent shift in the display will be greater for objects up close than for far-away objects, which can be terribly confusing and ineffective.
Strictly speaking, you don't need the offsets. It's always possible to get the screens to line up with each other just by using the rotations and the zoom feature on the projectors. Still, there are times when the offsets are too handy to ignore.

The values of "increment" are stored in the last part of caveut.ini:

```
[CaveUT.CaveUTInteraction]
RotateIncrement=0.500000
OffsetIncrement=1.000000changed
```

As you might guess, you can adjust the size of the change simply by editing these values and saving cave.ini. You will be amazed at how much difference a single degree can make, especially with Roll.

The changes made with the keyboard will be saved automatically when you exit UT2003. The numeric value of each keyboard parameter will be added to the corresponding parameter in cave.ini. Alternatively, you can change the cave.ini parameters manually, but you have to stop and restart the client each time, which is an inefficient use of effort.

Note: When CaveUT saves the offset values, it also writes a lot of other irrelevant material to the cave.ini file. This material is harmless and you can delete it if desired. Just be sure not to eliminate any of the CaveUT parameters.

So now you need to fine-tune the rotations and other parameters to make all the screens line up. It is usually best to tweak the center screen's configuration until it looks the way you want it to (which sometimes call for no changes at all), then line up all adjacent physical screens next to it, then fine-tune the rest.

Step 8: (Optional) If You Are Happy With the Default Sweet Spot, Then You Are Done

Otherwise, follow instructions in the next section.

Tip

Every time you edit CaveUT.ini, open it in your text editor, then close it when you are done. Many text editors will not allow another program to change the file while it is open for edit, so CaveUT will not be able to save your fine-tuning adjustments if the file is open in another program. Some text editors allow this (i.e. Notepad) but that can be confusing.

Next:	**VI. Configure Perspective Correction**
Previous:	**IV. Read This: Perspective**
Start:	**Back to Page Index**

VI: Configure Perspective Correction

This section describes how to put the sweet spot anywhere it is desired, as long as it is within view of the multiscreen display. This is accomplished with with certain parameters shown in the CaveUT.ini file. They are shown highlighted here:

```
[CaveUT.Spectator]
CaveRoll=0.000000
CavePitch=0.000000
CaveYaw=0.000000
CaveOffsetX=0.000000
CaveOffsetY=0.000000
CaveOffsetZ=0.000000
CaveFOV=100.000000
OffAxis=No
FOVleft=-45
FOVright=45
FOVtop=36.87
FOVbottom=-36.87

[CaveUT.CaveUTInteraction]
RotateIncrement=0.250000
OffsetIncrement=0.500000
FOVIncrement=0.500000
```

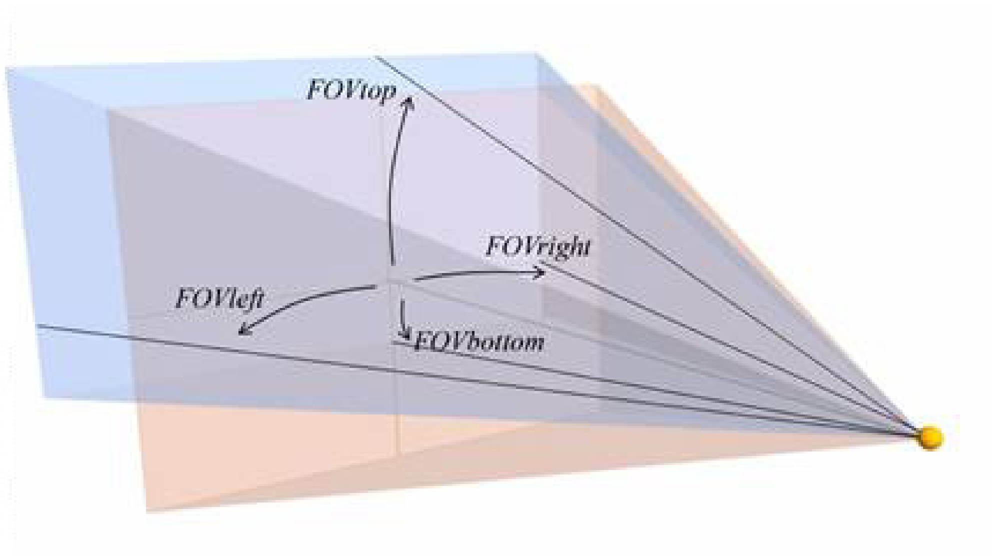
In other literature, this technique of producing a "lop-sided" projection is referred to as "off-axis projection."

As with the other parameters, they only govern the one display for the computer it is installed on. For this capability, CaveUT depends on VRGL, a generic package for perspective and other VR related visual effects.

Step 1: Set OffAxis=Yes

This will activate CaveUT's perspective effects. When OffAxis=No, the values in FOVleft, FOVright, etc., do not matter.

Step 2: Read This: What the Four "FOV..." Parameters Do



The four perspective parameters, **FOVleft, FOVright, FOVtop,** and **FOVbottm,** refer to the left, right, top and bottom sides of that display, as shown to the right. The frustum in pink shows the original, unmodified, view. Both views share a single view axis, which is the shortest perpendicular from the eye to the screen. In the unmodified view it is in the exact center of the screen. In the modified view, it is off-center, because the frustum is deliberately lop-sided, as specified by the perspective parameters.

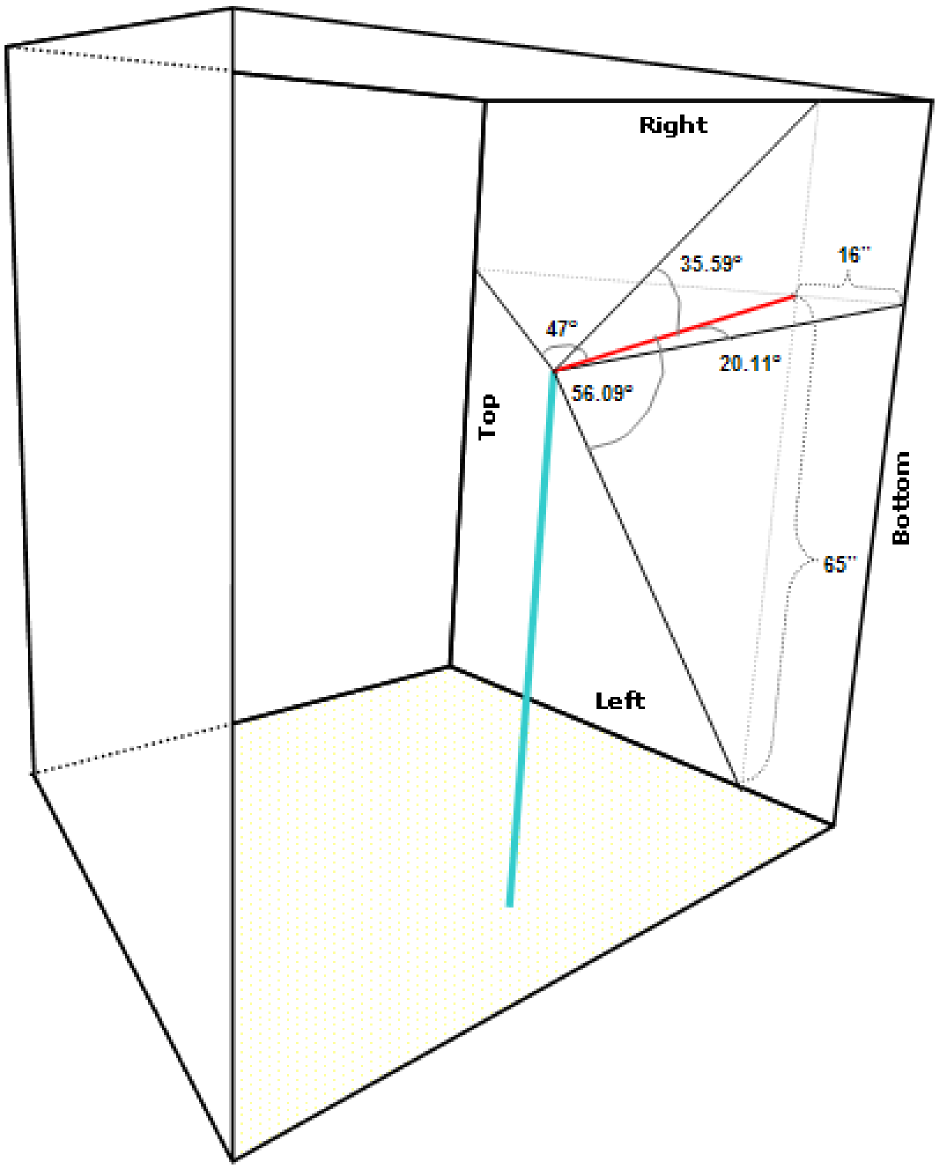
For example, FOVtop is the angle, in degrees, between the view axis and the center line of the top edge of the frustum. For a more technical description of how this works, look at the documentation for **VRGL**.

In the default CaveUT.ini, shown above, note how FOVleft is negative, while FOVright is positive. (By convention, left is the negative direction while right is the positive direction, like clockwise and counterclockwise.) For the view axis to actually intersect the screen, FOVleft must always be negative while FOVright is always positive. Similarly, FOVtop is positive and FOV bottom is negative.

It is sometimes desirable to make both parameters positive or negative to produce an extremely skewed view frustum, a more advanced use which is described at the end of this section. In other literature, this technique of producing a "lop-sided" projection is referred to as "off-axis projection."

Warning: If you enter values that make no sense, such as getting the positive/ negative signs backward, UT2003 will start, but it will produce visual nonsense. To exit the game, press the back quote key, "`", type "quit" and press `return`.

Step 3: Compute and Set the FOV Parameters for Each Screen



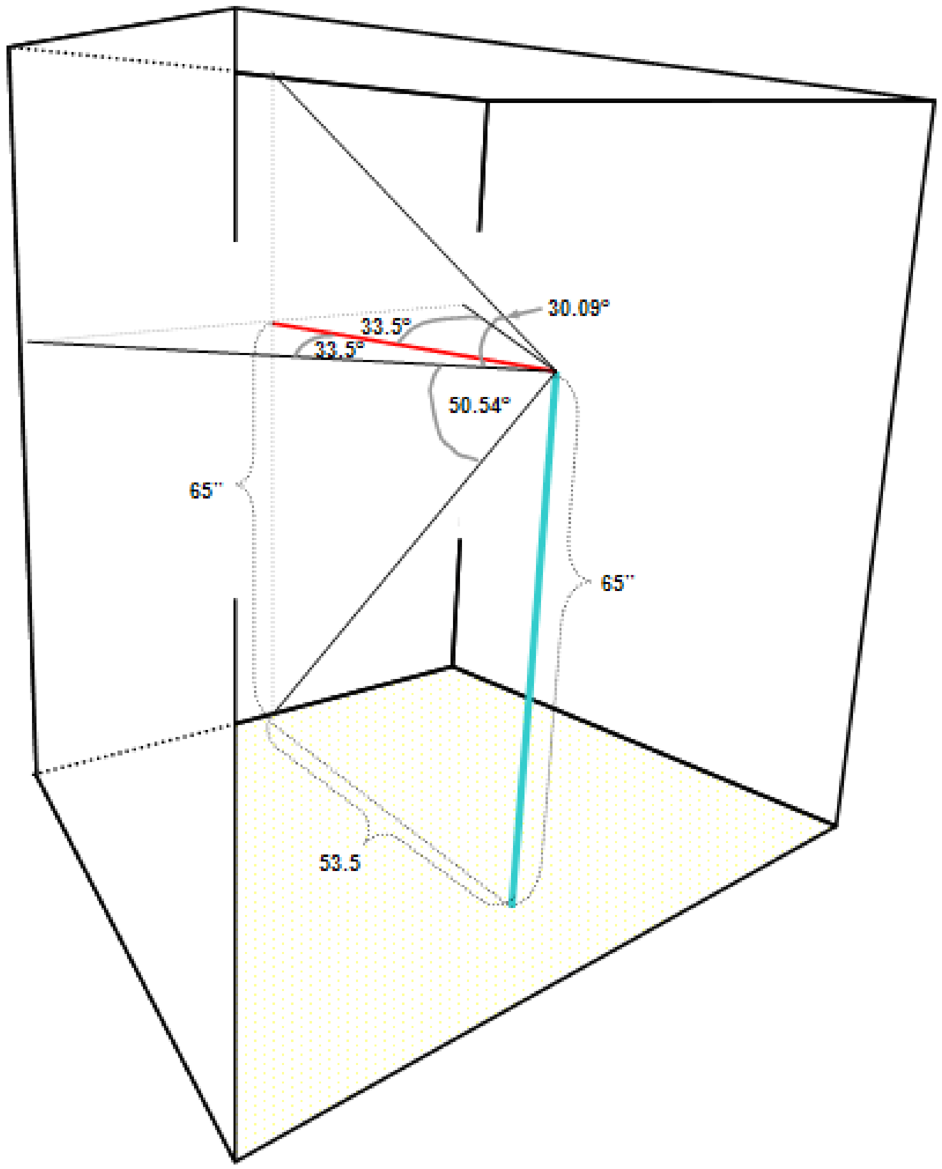
Go back to your diagram for your multiscreen display, add the new sweet spot, and compute the FOV angles using trigonometry. For example, the image on the right shows a diagram of the BNAVE and the measurements for the right screen. The blue line roughly represents the viewer standing in the BNAVE, where the top of the line is the sweet spot. The Left/Right/Bottom/Top markings on the right side screen show what the game engine on that PC *thinks* is going on.

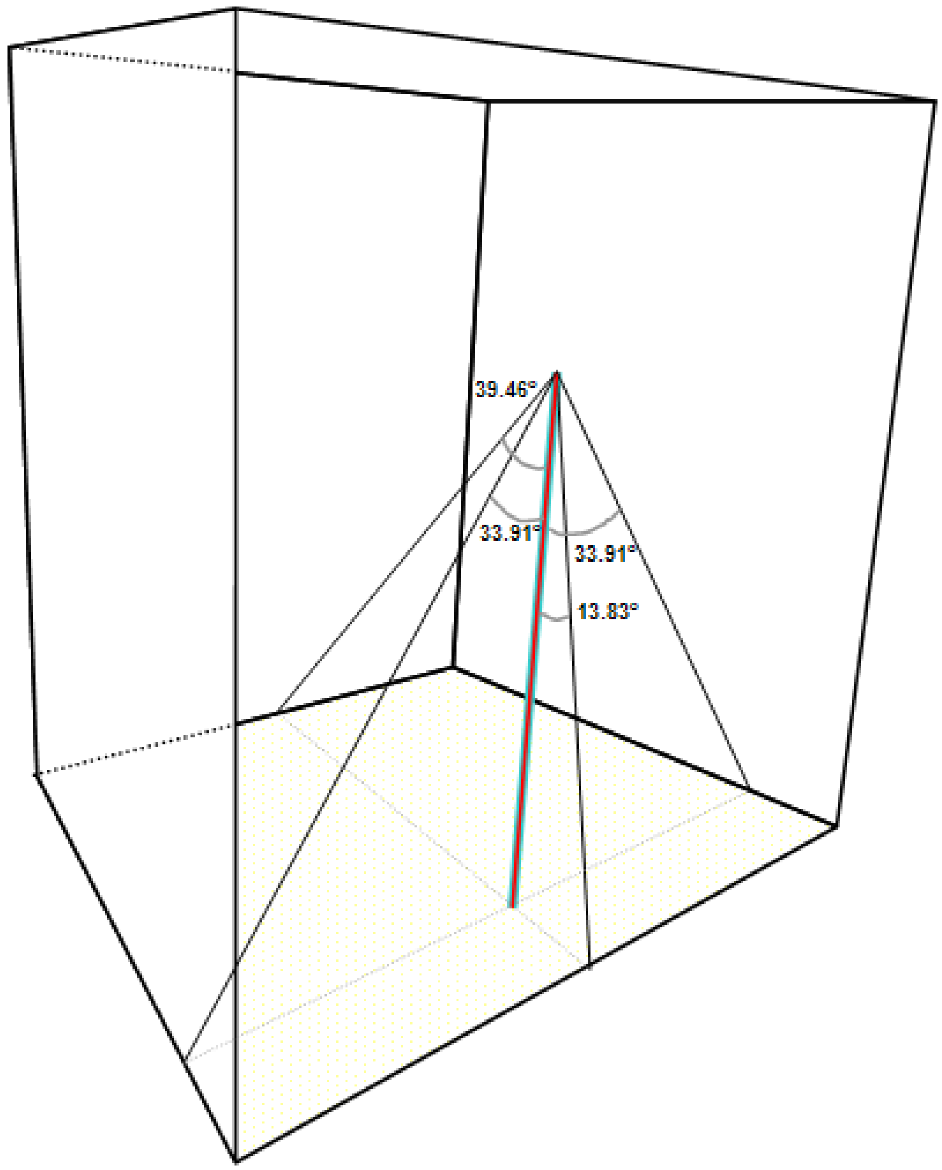
As you can see, the display is physically rotated so the "Right" side is actually on top, etc. This notation is somewhat confusing, but allows us to keep track of what amounts to disinformation we are providing to the game engine -- disinformation which must remain consistent.

The shortest perpendicular from the sweet spot to screen is shown in red, 43.7" long. That angle between it and the rightmost edge of the screen is 20.11 degrees. (21.11 deg = atan(16/43.7)) That edge is the "bottom" of the normal view screen, as far as the game engine is concerned, so in CaveUT.ini, `FOVbottom=20.11`. Similarly, `FOVright=35.59, FOVleft=56.09, FOVtop=47.` Study the diagram and make sure you understand why.

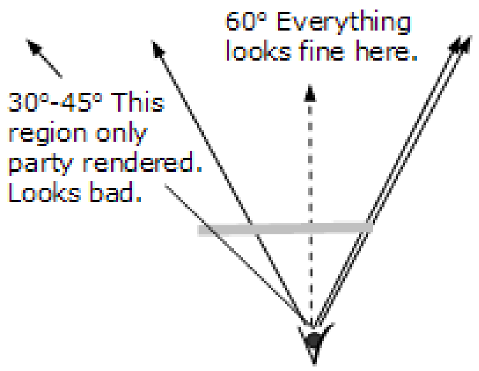
For the BNAVE, the perspective parameters are much the same, but with the parameters reordered or their signs reversed. It depends on which way the screen was rotated.

Two more diagrams complete the example. The first one diagrams the angles for the front screen and the second diagrams the floor screen.



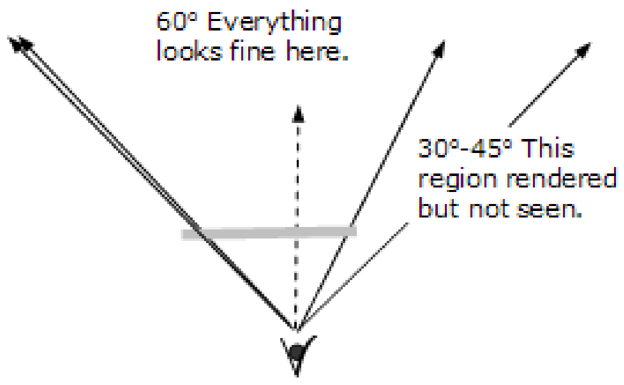


Step 4: Compute and Set the CaveFOV Parameter


60° Everything looks fine here.
30°-45° This region only party rendered. Looks bad.

The **CaveFOV** parameter tells the UT2003 game engine how much of the virtual world to render (that is, to calculate what game-world shapes should look like and make the information available to the display.) With the perspective effects turned off, CaveFOV it also determines the dimensions of the user's view frustum from the real and into the virtual world, so the engine only has to render just enough to accurately fill the viewer's display.

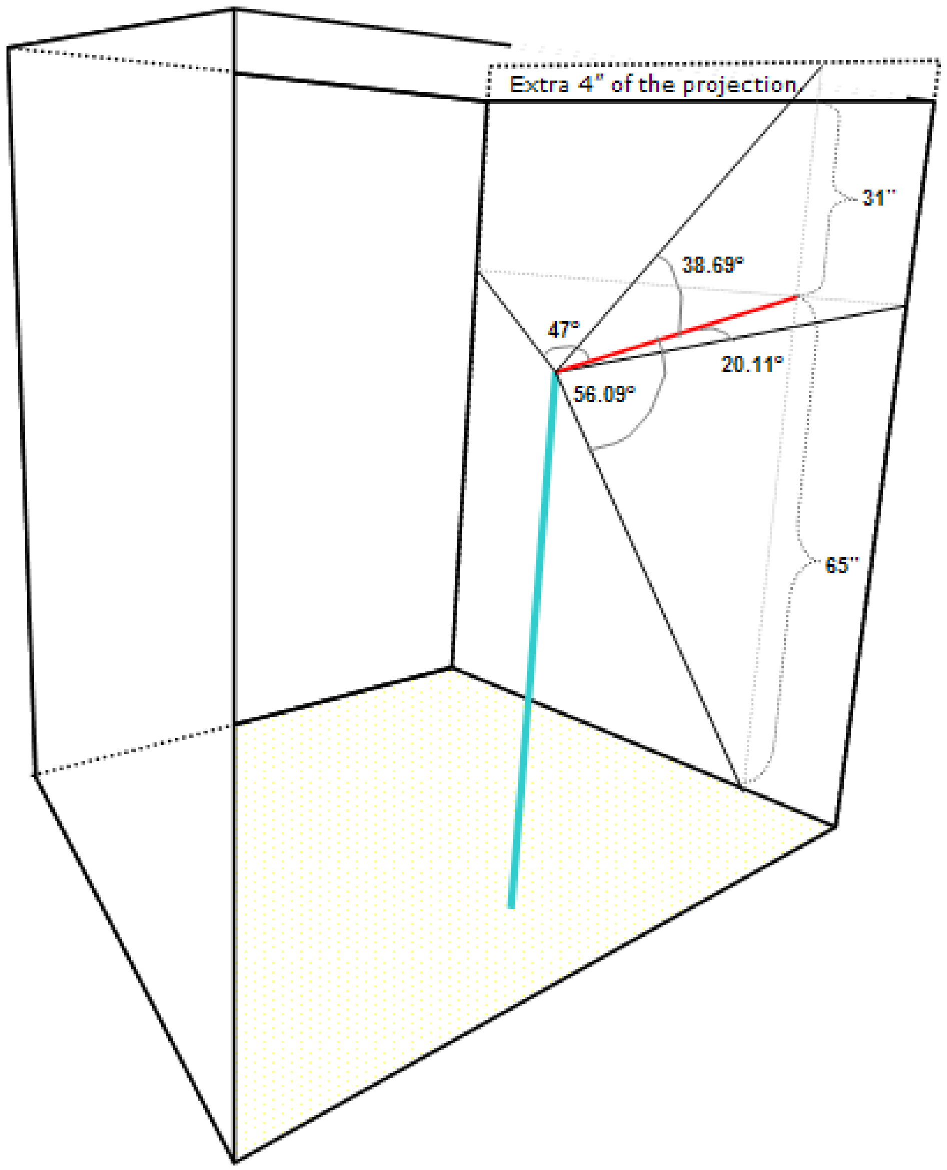
But when the perspective functions are active, (`OffAxis=Yes`), CaveUT directly controls the dimensions of the main view frustum. It is easy to show more of the virtual world than the engine thinks the viewer is seeing. This results in gaps in the display, where some or all of the visible objects are missing, as shown in the diagram above and to the right. In that example, `CaveFOV=60, RightFOV=30` (these numbers are correct), but `LeftFOV=45`, leaving a 15-degree area not fully rendered.


60° Everything looks fine here.
30°-45° This region rendered but not seen.

The solution to this problem is to simply make CaveFOV large enough to accommodate the CaveUT-specified view frustum. In the example on the left, `CaveFOV=90`, so the left half of the rendered view is 45 degrees -- large enough to the accommodate the part of the display specified by FOVleft. Note that the right half is also 45 degrees, which is more than what is needed. The engine renders an additional 15 degrees of the world that will not be shown. There's no way to avoid this, because the rendering engine will only render for a symmetric view frustum.

So why not set CaveFOV to some very high value and don't worry about it? Because the more the engine has to render, the harder it has to work and the greater the load on the computer -- a waste of rendering speed. In practice, you can have several displays with significantly different CaveFOV values and not notice a difference in performance, as with the BNAVE. However, when performance starts to lag because of a very large virtual word or too many (virtual) people in it, problems should show up first in the displays with high CaveFOV settings.

Step 5: Fine-Tune the Perspective Effects



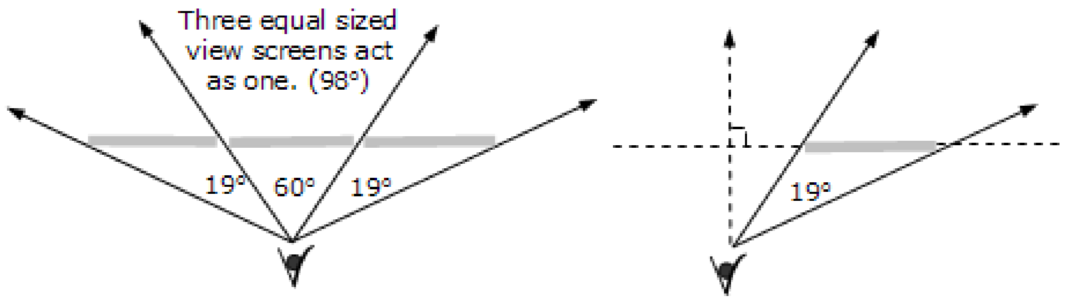
When you finally install and run CaveUT on your multiscreen display, CaveUT's perspective parameters will probably need some tweaking. You need to carefully measure the physical dimensions of the physical display and account for any discrepancies between that and your original diagram. For example, the image projected onto the right side of the physical BNAVE is elongated by a good four inches, an unfortunate side-effect of the optical setup. To make it fit, the current operators of the BNAVE simply run that extra four inches off the top of physical screen. The effect is a negligible loss of resolution on the right side and a small but important lengthwise stretching of the image.

The best way to compensate for this error in CaveUT is to recompute the FOVtop parameter, using the area of the projection itself, rather than that of the physical screen, as shown in the diagram on the right. The original FOVright was `atan ( 31"/43.7") = 35.59` , while the new one is `atan( (31"+4")/43.7" ) = 38.69"` Doing this makes the image look right, missing only a little bit of resolution.

Side Note: This is a small demonstration how CaveUT can handle rectangular displays of arbitrary dimension and placement.

It is very important make these types of perspective adjustments, because you will not be able to compensate for them with the rotations or offsets. In fact, if you are absolutely unable to make the screens line up using the keyboard fine-tune adjustments, it is a sure sign that you need to recalculate the perspective correction. There are no keyboard incremental adjustments for the perspective parameters -- you have to recalculate them and change the values in CaveUT.ini.

Advanced Usage: Making a Very Skewed View Frustum



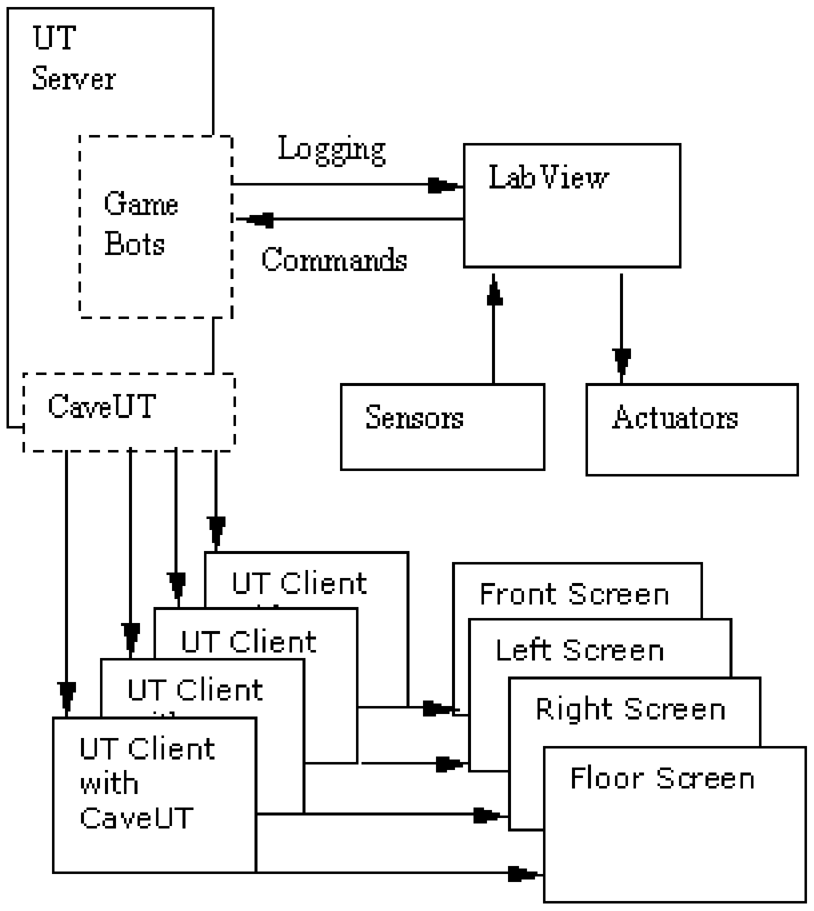
Situations crop up in which the main view axis for a screen does not intersect the screen itself.

In the example diagrammed on the left, three screens are used to make one, flat, composite display. The two side screens can only be viewed an oblique angle. To make them show the virtual world in a visually naturalistic way, the perspective correction has to be rather severe. All of the FOV parameters are calculated as before, using the shortest perpendicular from the sweet spot to the *plane in which the view screen lies*. In fact, it was never necessary for this perpendicular to intersect the screen itself, it just happens to be that way for the screens in the BNAVE example, above.

In this example, FOVleft will be *positive* thirty degrees, while FOVright a positive 49 degrees. FOVleft can be positive, just so long as FOVright is larger. FOVtop and FOVbottom could be manipulated in exactly the same way to skew the view frustum vertically.

Next: **Controlling CaveUT From a TCP Socket**
Previous: **V. Configure View Rotations and Offsets**
Start: **Back to Page Index**

Controlling CaveUT From a TCP Socket



CaveUT can be controlled from LabView to instrument psychophysical experiments.

GameBots is another freeware *Unreal Tournament* modification intended for scientific research. Once installed, it allows the researcher to open a standard TCP connection to *Unreal Tournament* server which causes a human figure, a "bot", to appear in the virtual world. This bot can be controlled through commands sent to GameBots over the TCP connection from some sending program. The sending program can be written in any language capable of opening TCP sockets. One could even use Telnet, though this is recommended only for debugging purposes.

Spectator mode in UT can "watch" a bot created by GameBots. Therefore, you can install GameBots on your UT server along with CaveUT and effectively control the display over the TCP connection just by moving the bot! You can also create more bots (which can look like anything) and can be controlled over their own TCP connections.

This allows you to write your own software to present a wide range of visual experiences and interactive designs.

For example, UT/GameBots/CaveUT and LabView have recently been used to instrument an experiment in the **BNAVE**, with help from Dr. Patrick Sparto and Leigh Mahoney. Lab View is a software package used by many experimental scientists to perform data acquisition and instrument control for lab experiments. It can be used to control actuators, read sensors, and execute programs written in its built-in scripting language.

At the appropriate times during the experiment, the experimenter ran a Lab View script that sent commands to the operator PC's UT installation. The diagram to the right summarizes the setup.

The only drawback is that the TCP connection is slow compared to the graphics rendering, so there's a small time lag from when the command is issued to the bot performing its action. This is fine for general commands like "follow this person" or "go find the flag," but using it for animation can be a bit clumsy.

The UT2003 version of GameBots is still under development, but it is serviceable.

Next: **Interface Options**
Previous: **VI. Configure Perspective Correction**
Start: **Back to Page Index**

Interface Options

The two-walled UT-Cave setup was demonstrated at the **Ultra Unreal** gamer convention (19-21 July 2002). This was the first time the CaveUT 1.2 software had any serious use and the first time CaveUT had been used by large numbers of gamers. There, expert players such as **Ian MacKechney** helped experiment with different control strategies.

Interestingly, most of the UT players at Ultra Unreal use keyboard and mouse, but that's not an ideal option for a CaveUT setup. For normal usage, it would be necessary to construct a platform to hold the interface devices, and the platform's physical presence would partially separates the player from the visual display, degrading the experience.

Standard implementation of the UT-Cave used a MacAlley USB Airstick for the PC bought in the summer of 2001. This is essentially a joystick which works normally, but without a physical base. Instead, it uses built-in accelerometers to detect tilt, allowing the player to simply hold it in mid-air. The control settings for to enable this device are (*Unreal Tournament Main Menu* | *Preferences* | *Controls*):

```
Fire: Joy-5
Alternate Fire: Joy-6
Move Forward Joy-4
Move Backward Joy-1
Strafe Left Joy-3
Strafe Right Joy-2
Jump: Joy-7 or Joy-8
Crouch/Down JoyPovDown
Joystick X-Axis Turn Left/Right
Joystick Y-Axis Look Up/Down
```

Previously, the Gravis Destroyer game pad worked reasonably well, and seemed to be easier to learn. But it lacked the analog control a joystick offers, making targeting a little bit more difficult.

Obviously there are many other interface options.

Next: **Tips and Tricks**
Previous: **Controlling CaveUT From a TCP Socket**
Start: **Back to Page Index**

Tips and Tricks

When setting up CaveUT, try turning off spectator mode, turn on the HUD and make sure the crosshairs are visible. The crosshairs mark the vanishing point of the screen image. Use that for calibrating your off-axis projections and/or view rotations.

Instead of the crosshairs for targeting, which are turned off in CaveUT, tape a penny or similar marker onto the screen where the weapons fire is focused. In a typical two-walled V-Cave, this is about two-thirds of the way up from the bottom along the central seam.

One of the things about CaveUT that amazes many programmers familiar with multiscreen displays is that CaveUT makes no attempt to synchronize the updates in the multiple screens, commonly known as "genlock". Each screen just updates as fast as it can. As long as all the screens can update at 30 frames per second or faster, the viewer will never notice. If performance begins to lag, however, there will often be situations where one screen has updated because the view moved or rotated, but another did not, causing a visual disconnect between the two screens. In practice, this will not be a problem as long as the machines are offering fast performance. (Note that genlock is required for stereoscopic displays, and CaveUT/UT2003 is not yet stereoscopic.)

Next: **Packing A Portable CaveUT For Air Travel**
Previous: **Interface Options**
Start: **Back to Page Index**

Packing A Portable V-Cave For Air Travel

The picture to the right shows the entire two-walled V-Cave, computers projectors, screens and all, packed into travel bags. It can be carried as standard airline luggage, with 50 lbs. (about 23 kg) and several cubic feet of extra capacity in the bags for personal items. Everything fits into the standard dimensions of check-through and carry-on luggage for airline flights in the continental United States, without incurring an extra fee for oversize or overweight luggage.

Roughly, the four pieces are:

Green Bag

This bag contains everything for the portable screens: the screen materials, the PVC pipes, the joiners, the clamps, the carpet seam binders, and the rope. Altogether it weighs about 65 lbs. (about 30 kg). On the last occasion the V-Cave was transported this way, in April of 2002, all of the U.S. air carriers had a 70 lb. (about 32 kg) weight limit for check-through bags and a 62" (about 157 cm) limit for total bag dimensions (height, width and length of the check-through bag could add up to no more than 62"/about 157 cm).

Black Bag

This bag contains the tripods, network hub and cables, power cables and strips, tape, glue, plastic hammer, and personal effects not related to the V-Cave. It weighs in at 50 lbs. (about 23 kg). Clothes make excellent packing material, especially for the tripods.

Green Suitcase

This bag contains the two projectors and their laptops, and, because of the delicacy of this equipment, was a carry-on bag.

Because the laptops were large ones, the case was just barely within the limits for a carry-on bag for the air carrier being used (those limits, in April 2002, being 10"x13"x22" or about 25 cm x 33 cm x 56 cm) and a 50 lb. (about 23 kg) limit.

Unfortunately, the rules for carry-on baggage vary from one air carrier to the next. Check with the air carrier you intend to use before transporting a V-Cave this way. It's also possible to select your components in order to minimize the likelihood of problems: Buy smaller laptops, compact projectors, and so on.

Black Shoulder Bag

This bag contains the third laptop and some reading materials. It's small enough to constitute a "personal item" rather than a "carry-on bag" as defined by most U.S. air carriers, which is useful, as many air carriers permit a total of one carry-on and one personal item.

Important Note

It's useful to remember that you should choose luggage that is rated for the weight of the gear you'll be carrying. Low-cost luggage, especially soft-side luggage, is far more likely to split when being used to haul heavy loads across long distances. We've had good luck with hockey bags, lately.

Next: **Safety And Motion Sickness**

Previous: **Tips and Tricks**

Start: **Back to Page Index**

Safety and Motion Sickness

It shouldn't surprise anyone that players and operators of a CaveUT setup can experience motion sickness. Players who are particularly sensitive should be warned that they can develop a headache or nausea which could last a while.

Players who are unsteady on their feet for any reason should be advised to sit while using the cave.

When showing a CaveUT setup to anyone who has not used immersive virtual reality before, the operator should stand near to them initially in order to assist them if they become dizzy.

In an institutional or other lawsuit-conscious setting, operators may wish to implement stronger safety measures, such as the safety harness in the BNAVE, and legal waivers may be advisable.

Next: **Known Issues**

Previous: **Packing A Portable CaveUT For Air Travel**

Start: **Back to Page Index**

Known Issues

The CaveUT Mutator tends to write a lot of extra information into CaveUT.ini when it saves the view rotation and offset adjustments. This may be annoying, but it is but harmless. With luck, future implementations of CaveUT will eliminate this issue.

Animated 2D images that depict 3D shapes (billboards) don't look right in CaveUT. Examples include the background in CTF-Face, fires, and weapons when they are spinning on the ground, waiting to be picked up. These images and animations are pasted onto 2-D surfaces located *in* the 3D virtual world. As a result, the CaveUT hacks cannot change the way the objects in that image look, because they are already rendered. In fact, if such an image falls across the intersection of two screens, its two parts will not line up. This is not so much a bug as an inherent limitation with billboards in 3D engines, and remains a problem for all versions of CaveUT.

There is a long standing bug (or feature) in UT's spectator mode (v432, v436, UT2003) which makes rotations in CaveUT jerky. From the perspective of many players, about 1/3 of a second passes between the time a player starts to rotate and the time the spectator's view begins to rotate. Then the spectator's view jumps into the correct position, creating a visual jerk. Strangely, when the player uses an analogue input device such as a mouse or joystick, the rotations are much smoother, but still not as smooth as they should be.

Always be sure to use 32-bit color depth in UT's video preferences. For some reason, this approach greatly reduces problems with coplanar polygons in some virtual worlds. When this parameter is changed, UT will typically crash, but when restarted the program will function correction with the new parameter in place.

Next: **Improvements Needed**

Previous: **Safety and Motion Sickness**

Start: **Back to Page Index**

Improvements Needed

Care to help? CaveUT is a completely open-source effort, protected under Epic Games' GPL agreement. It remains a work in progress, and the community developing CaveUT hopes that people in the Unreal community are interested in developing CaveUT further.

At some point, the CaveUT team will be ready to setup a Source Forge or equivalent project. In the meantime, we are happy to post patches and/or additions to CaveUT, and to help anyone who wants to work on them.

Among CaveUT's needs:

Develop More and Better Control Interfaces

There can never be enough testing and innovation on controls for CaveUT. Our most pressing need is to to separate firing direction from movement direction; options that have been suggested include using a head tracker, moving with a separate device (perhaps hand-held or foot-controlled), and performing weapon targeting with a third control.

Introduce a Spherical Correction

It's not readily apparent from the lovely shot of the **Earth Theater**, but CaveUT has a problem with that display, because the theater's screens are curved rather than flat. Currently, CaveUT works by simply carving out a section of UT's (flat) display.

01/22/04 Willem De Jonge has just completed preliminary code for a spherical correction. We hope to include it in CaveUT2004

Real Time Head Tracking

This task would require a real-time data stream from the head tracker to both VRGL and the UT2003 code. In the game code, the tracker could simply move the player as any other game peripheral would. The hacked OpenGL library (VRGL) would have to open a socket to receive the offsets from the tracker driver. Simple, in theory, but not a small amount of programming....

Stereographic Display

One of the ways we perceive depth in the real world is through the fact that each eye sees the world from a slightly different angle. Out to a distance of about eight feet, the human brain is able to calculate the distance from your eyes to any object using this disparity.

Stereographic displays effectively project both the right and left eye views onto some virtual scene. Special glasses separate the images for the viewer so that each eye sees the scene meant for it.

Marc LeRenard got preliminary stereo working for CaveUT 1.2, recently (01/15/04). Once it is debugged and doucmented we will make it available on this site.

Next: **Ownership and Distribution**
Previous: **Known Issues**
Start: **Back to Page Index**

Ownership and Distribution

CaveUT

Use of the CaveUT 2003 Mutator is covered by the End User Licencse Agreement distributed with the game itself.

Generally, anybody can use the mutator and anybody can change it, but all versions of the mutator must remain available to the public. Epic Megagames holds the exclusive right to generate revenues from use of the mutator and all its modifications, either through direct sale or paid use.

That said, Epic has traditionally been enthusiatically supportive of academic and scientific research, in general. They have been particularly friendly to the CaveUT project.

For more information on licensing issues, contact **mrein@epicgames.com**.

This Documentation

The web page titled **VRGL** is not covered by this copyright notice. Though CaveUT 2003 makes use of VRGL, VRGL is a separate package covered by its own LGPL license. It is in no way to be considered a part of UT2003. Information about rights related to that page appear on the page itself.

Next: **Credits**
Previous: **Improvements Needed**
Start: **Back to Page Index**

Credits

Personal Contributors in reverse alphabetical order:

Joe Manjlovich: Designed the CaveUT2003 Mutator and the Linux version of CaveUT 1.2.

Willem de Jonge: Willem wrote **VRGL**, a modified OpenGL library package upon which CaveUT2003 depends. VRGL accomplishes view rotation math for the CaveUT Mutator.

Jeffrey Jacobson: CaveUT was his big idea; general design and project management, testing and integration; provided content and wrote the rough draft of this web site; performed view rotation mathematics and wrote OpenGL code for CaveUT 1.1.; performed the physical design of the V-Cave design and provided instructions for replicating it.

Zimmy Hwang: Wrote game code and provided general help with CaveUT version 1.1.

Aaron Allston: Edited the web-based documentation.

Institutional contributors in reverse alphabetic order:

Ken Sochats and his **Visual Information Systems Center:** Sochats' lab's contribution to CaveUT lies mainly in the fact that they have used it extensively, becoming very helpful in debugging CaveUT. In addition, they built the desktop-sized MiniCave and their own portable V-Cave, demonstrating them all over the Pennsylvania, have loaned Jeffrey Jacobson equipment, and continue to pursue a number of CaveUT-based projects.

Dr. Mark Redfern and his **Medical Virtual Reality Center:** Permitted

Zimmy Hwang and Jeffrey Jacobson to use the BNAVE for development and testing. By extension, the creation of CaveUT is partially sponsored by NIH P30 Grant DC05205 through the Department of Otolaryngology, University of Pittsburgh.

Dr. Michael Lewis and his **Useablity Lab**: Provided extensive personal, academic, and material support. By extension, the creation of CaveUT is partially sponsored by AFOSR contract F49640-01-1-0542 through the Department of Information Sciences at the University of Pittsburgh. Thanks also go to everyone else in the department who have helped in other ways and generally tolerated Jeffrey Jacobson's creative process.

Epic Games: Wrote an excellent game engine and associated open-sourced game code.

Start: **Back to Page Index**