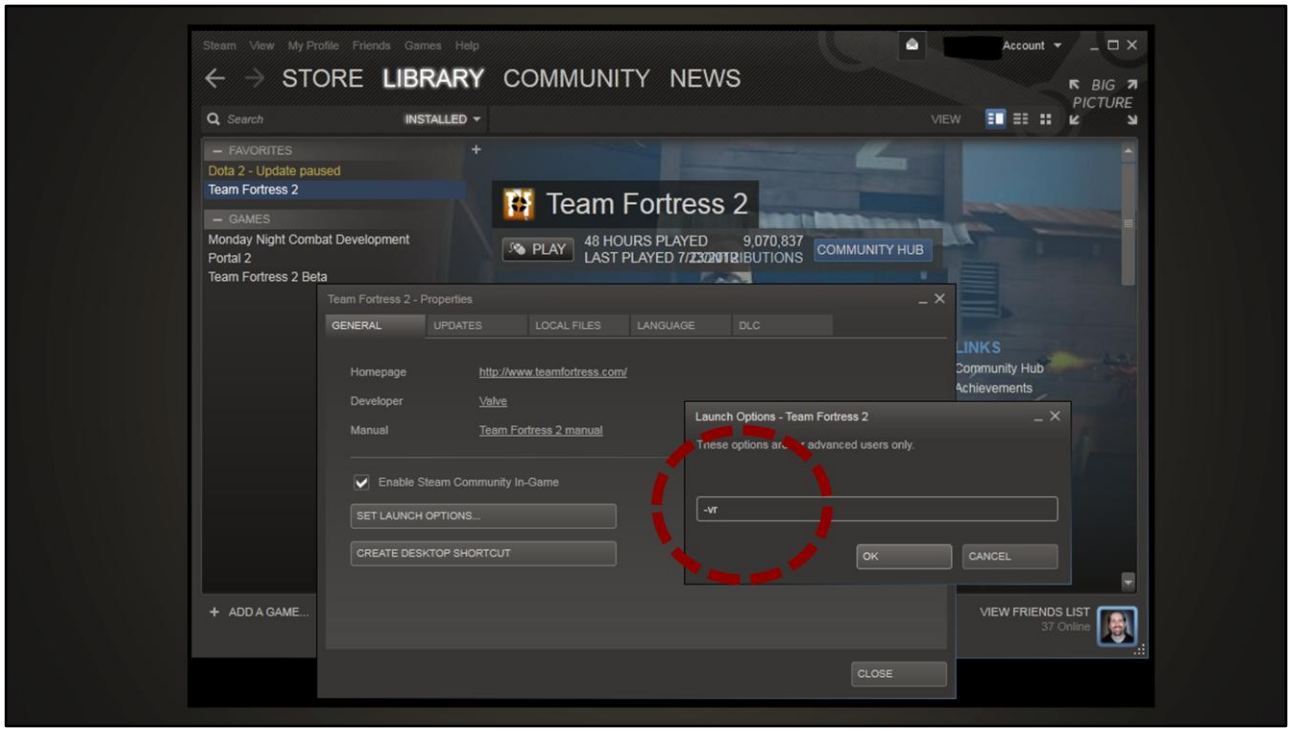
Lessons learned porting Team Fortress 2 to Virtual Reality

Joe Ludwig – joe@valvesoftware.com

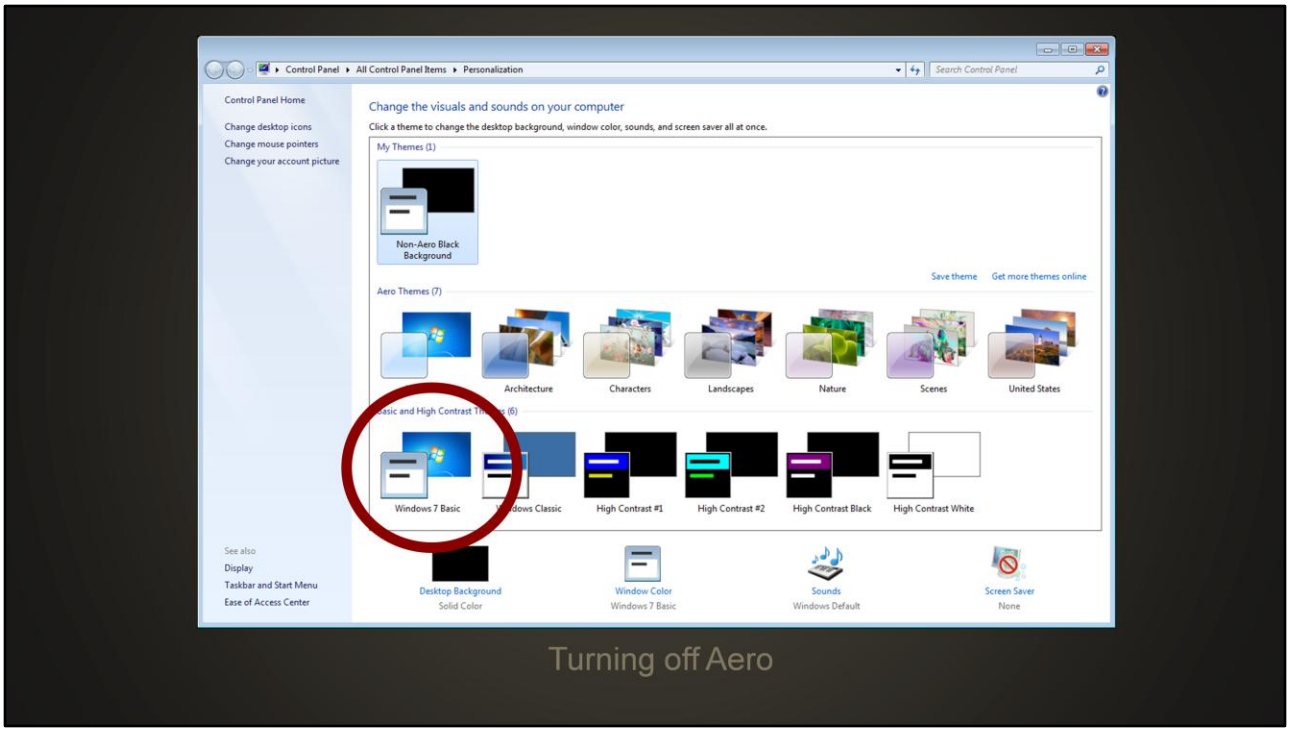
Since there's a lot to get through and not much time, I'm going to move through most of this fairly quickly. I'm also not going to talk much about the game itself, so if you have no idea what this image is about, this talk isn't going to help. There will be some time at the end to ask questions, and both Michael and I will be around for a bit after the talk.



What we did is add support for VR to TF2. If you have an Oculus Rift dev kit, just edit the Properties for Team Fortress 2 in Steam and add –vr to your launch options, and you'll be in VR mode. This is an update to the game that shipped last week, and TF2 is free to play. I'm going to describe what we did in TF2 as best I can with words and pictures, but I encourage you to go check it out for yourself.

Most of this development happened on either an nVis ST-50 or on one of the foam-core and duct-tape Oculus Rift prototypes that Palmer provided in the fall. Be grateful that you'll get to work with the dev kits they're shipping now instead of either of these options.

I'm going to talk quite a bit about the Rift today, but everything I say here also applies to other VR head-mounted displays. It's just that most of you are going to have the Rift and probably won't have an ST-50 or other display. We also experimented with other tracking system, but you will have the Rift's gyro-based tracking, so that's what I'm going to talk about.

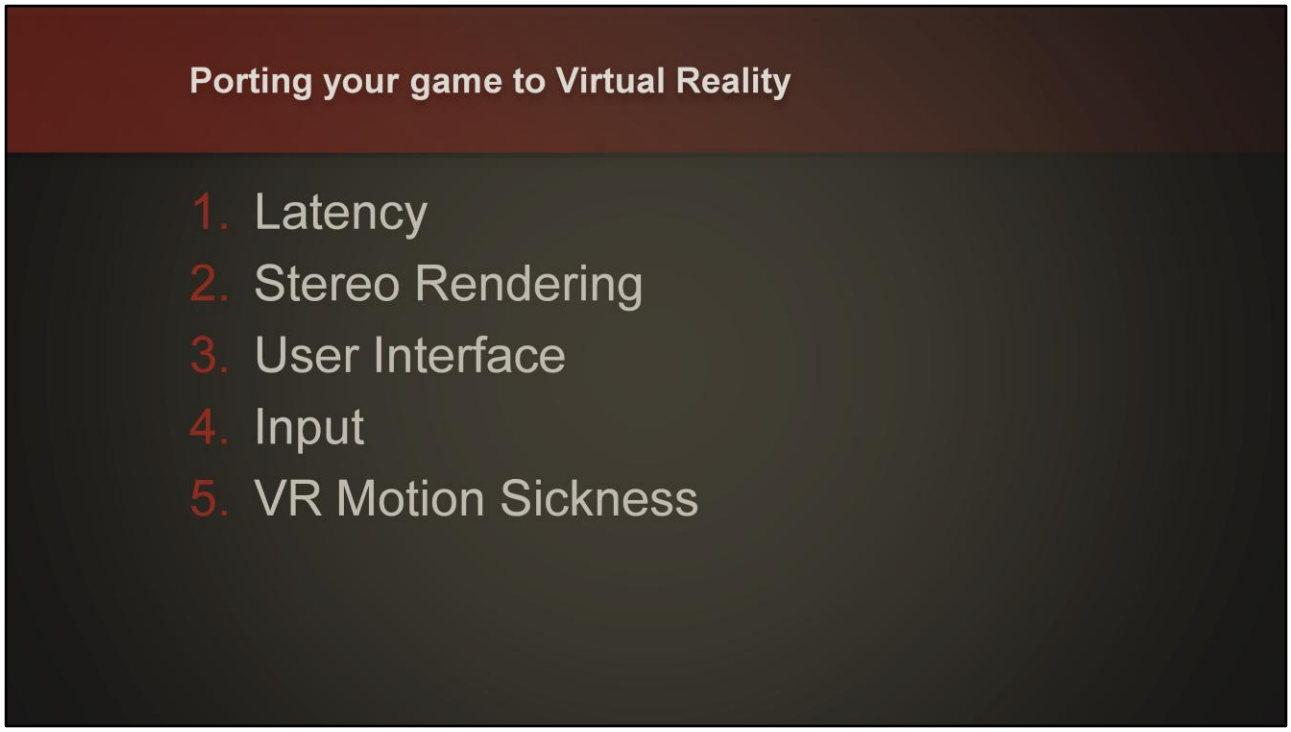


Before we get to the in-game stuff there are a couple of development quality-of-life things I wanted to mention. The first is that you really want to turn off Aero. The compositing that Windows does in Aero adds at least a frame of latency when you are running in a window. Do this in the “Personalize” option on the desktop context menu. Just pick any non-Aero theme.

The other thing I wanted to mention… DVI or HDMI splitters can be very useful when you're testing a VR game. Unless your game can render the game view to a second window, they will be the only way to see what the person wearing the display is seeing.

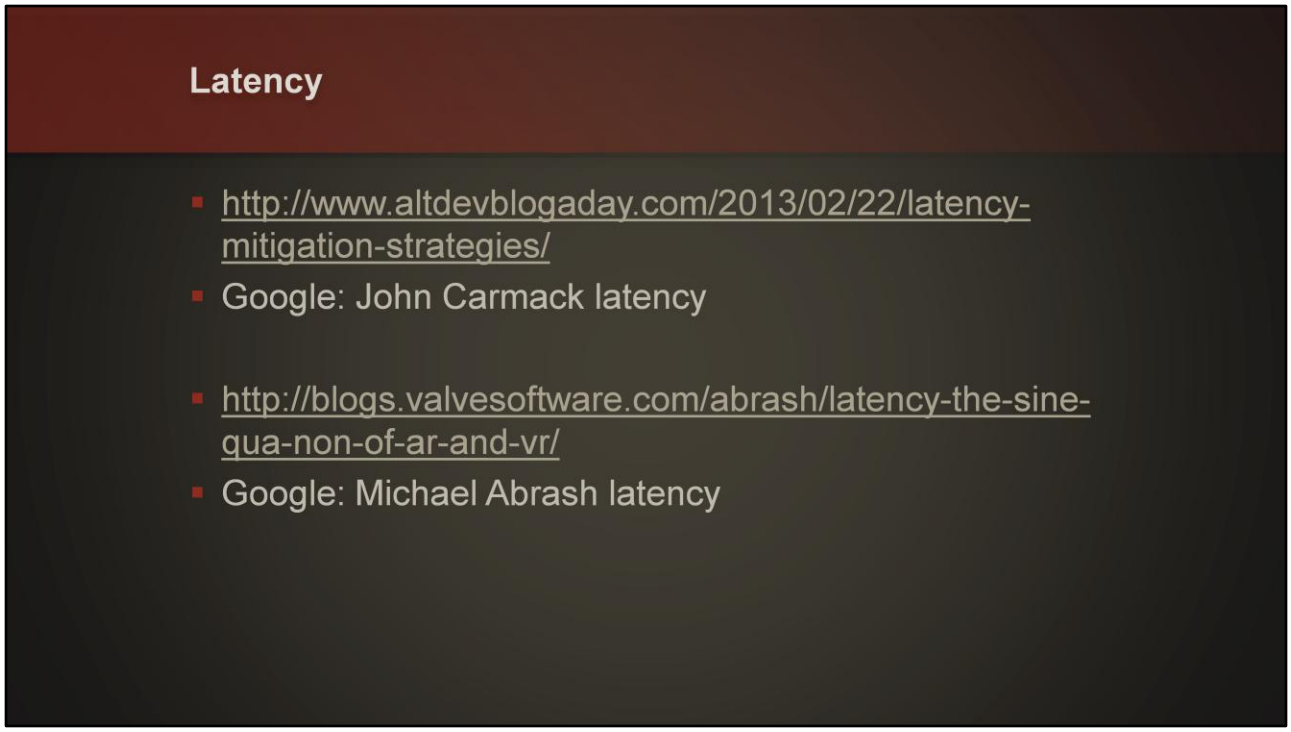
We use a DVI splitter by Aluratek that works pretty well. Amazon has them for 80 bucks.

One thing to watch out for is the display ID that the splitter reports to Windows. You may need to plug and unplug the inputs and outputs of your splitter in a particular order to get the ID of the head mounted display to get passed through to Windows. If the wrong ID is reported the Rift SDK won't know where to put the window and will refuse to recognize the device.



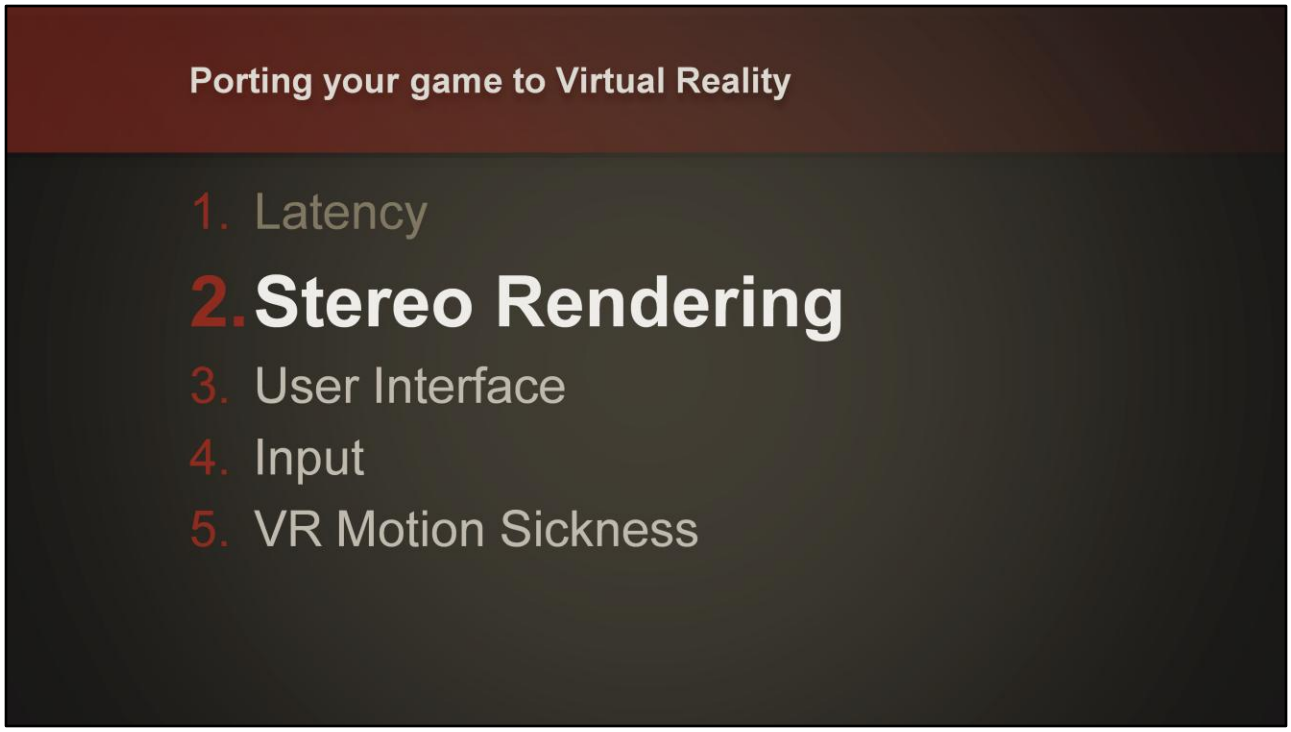
The actual process of getting an existing game running in VR is pretty straightforward. You will probably have the basics up and running within a week. Then you only have to do the other 95%. Hopefully what we've learned will help you save some of that time. We think that our second game on the same engine will take about a man-month.

At a high level, what I'm going to talk about today is latency, stereo rendering, how to deal with the UI in stereo, approaches to integrating mouse and keyboard input in the game, and some quick ways to make people sick and how to avoid them.

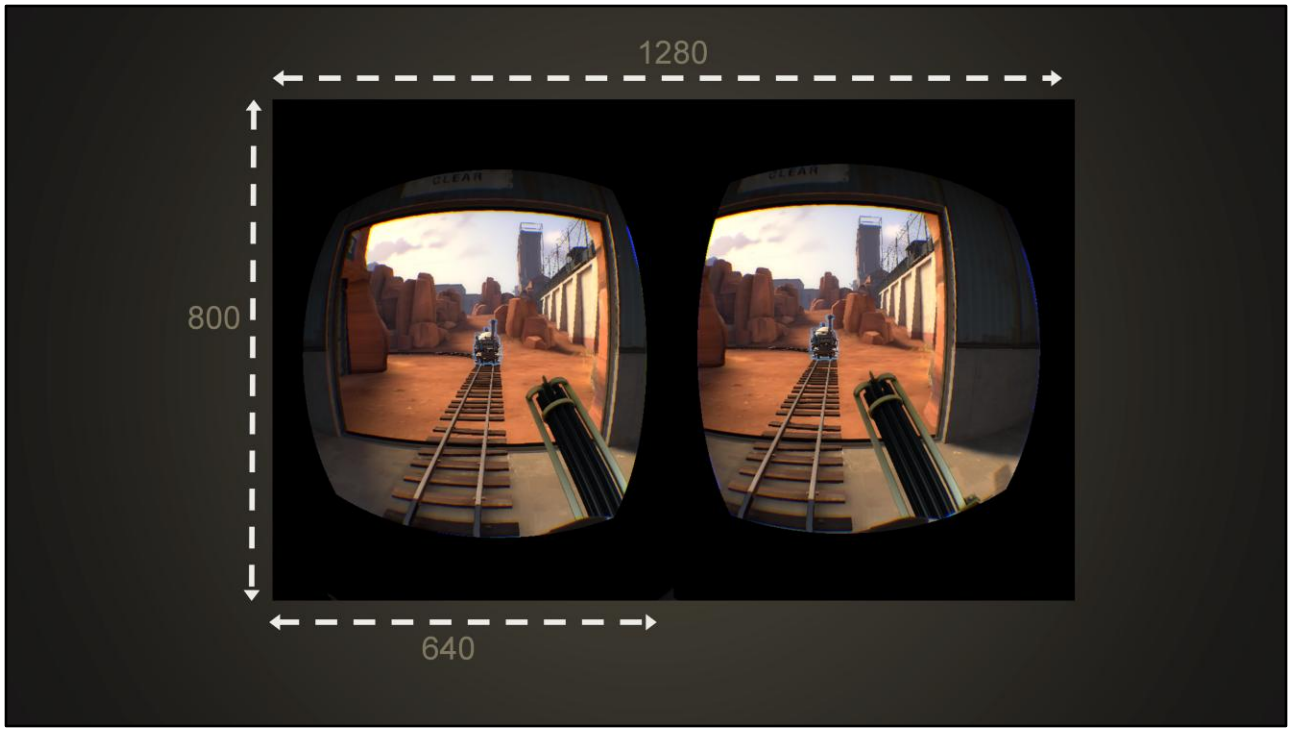


Do you remember about ten seconds ago when I said I was going to talk about latency. Well I'm not. Having too much latency in your system is the best way to make people sick, so it's incredibly important. But giving latency the time it deserves would take the whole rest of the session, so what I'm going to do instead is point you at a couple of blog posts by a couple of smart guys. John Carmack and Michael Abrash have both written posts that get into the details. I recommend reading both of them.

Here are the URLs, but if I were you, I would just Google their names and "latency" and these posts will be the top hit.



Next up is stereo rendering. What should you actually draw to the screen to have your game work in VR? Until you get this right, looking through the display is going to break your brain.



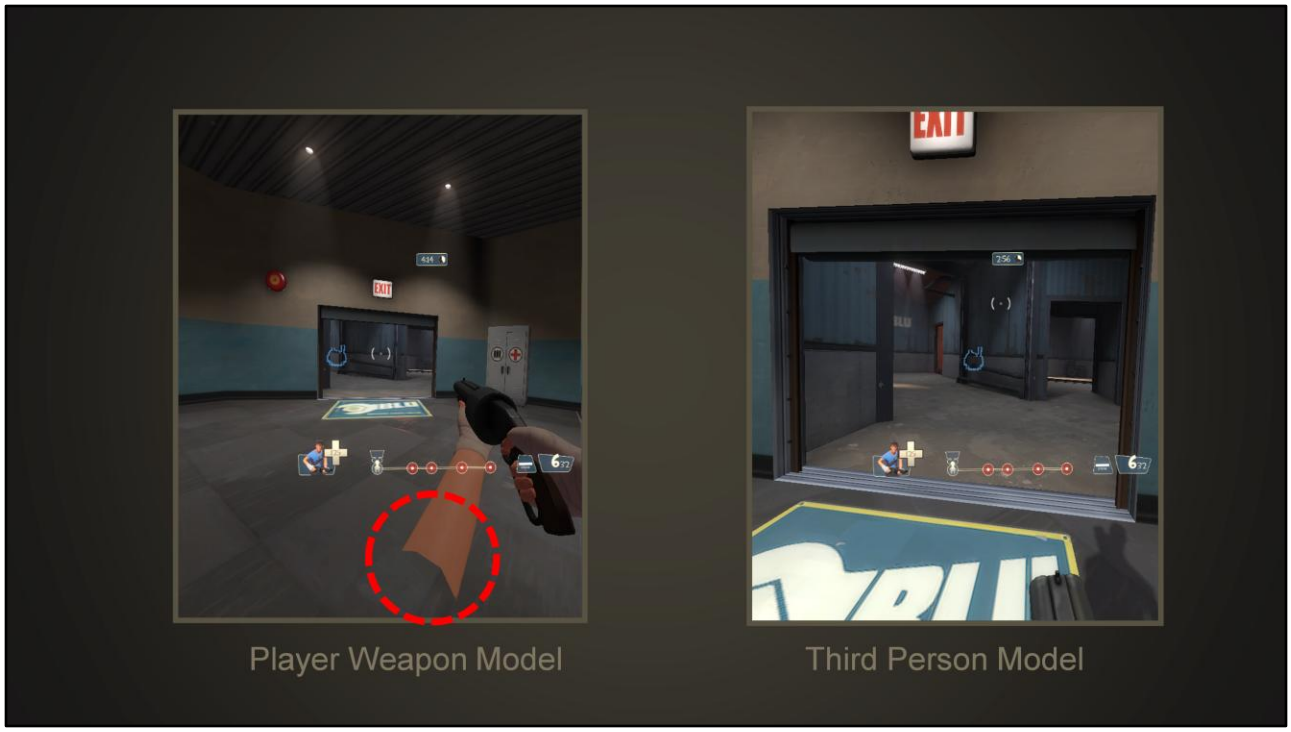
When you get your display connected it will appear to windows as an extension of your desktop. You need to get the game's window up in the right spot on the desktop or run full screen on the right adapter.

Once you do that you will need to split the window for the two eyes. On the Rift that means you divide the 1280x800 display in half horizontally and end up with one 640x800 viewport for each eye.

Exactly how to get your engine to draw the two views is something that will vary wildly from engine to engine. In Source it meant adding what were essentially "for each eye" loops in the top-level rendering routines then fixing everything that was broken by that. One of the biggest sources of stereo bugs for us were screen-space effects. Things like the "I'm on fire" effect or the "I'm underwater" effect. I don' t think there was a single screen-space effect in TF2 that worked the first time.

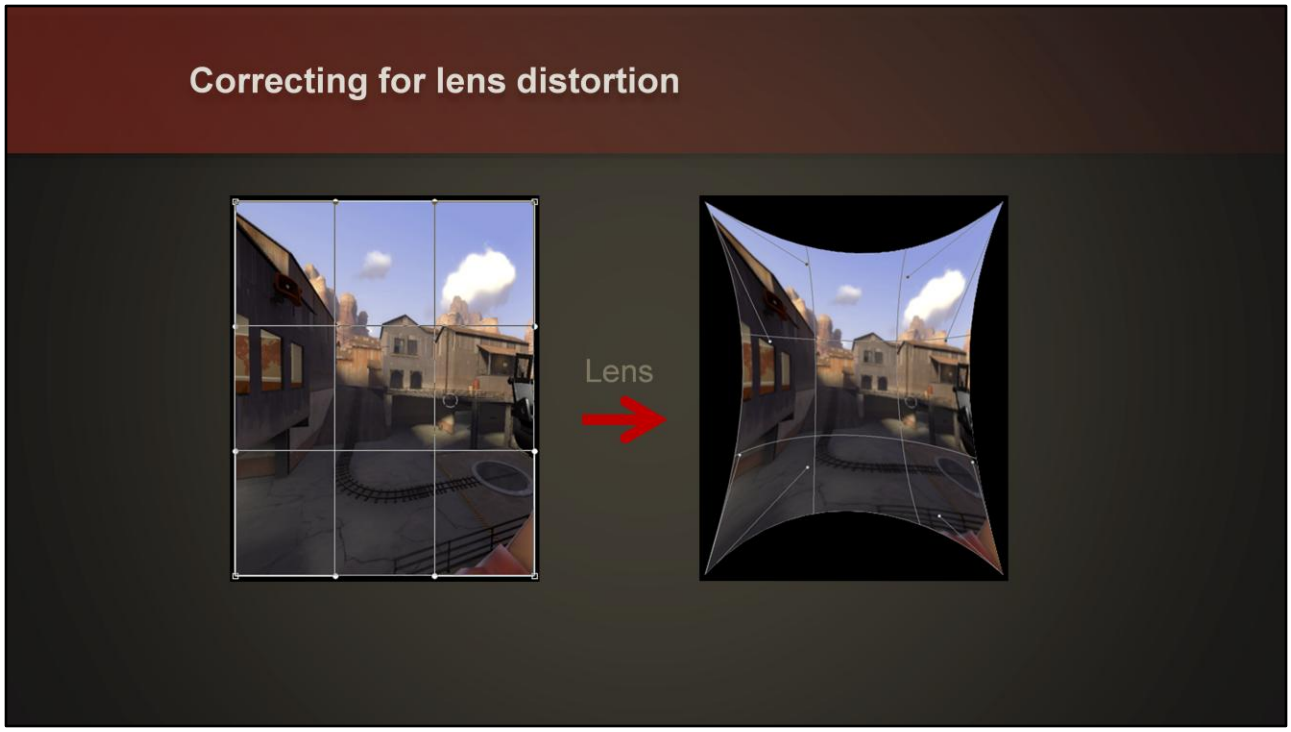
These two viewports form a stereo pair, so they have slight different perspectives on the world. We found it useful to use the concept of a “middle” eye that was the player’s traditional viewpoint in the game and then have the left and right eye positions be offset from that.

The projection matrix for each eye will be driven by what display you’re using. To avoid VR motion sickness it is very important to use transforms that match the physical attributes of the display. Anything else will cause mismatched signals in the user’s brain and cause problems. In the case of the Rift, SDK will tell you what need to know, so take a look at the SDK documentation for specifics. I’ll get more into VR motion sickness in a bit.



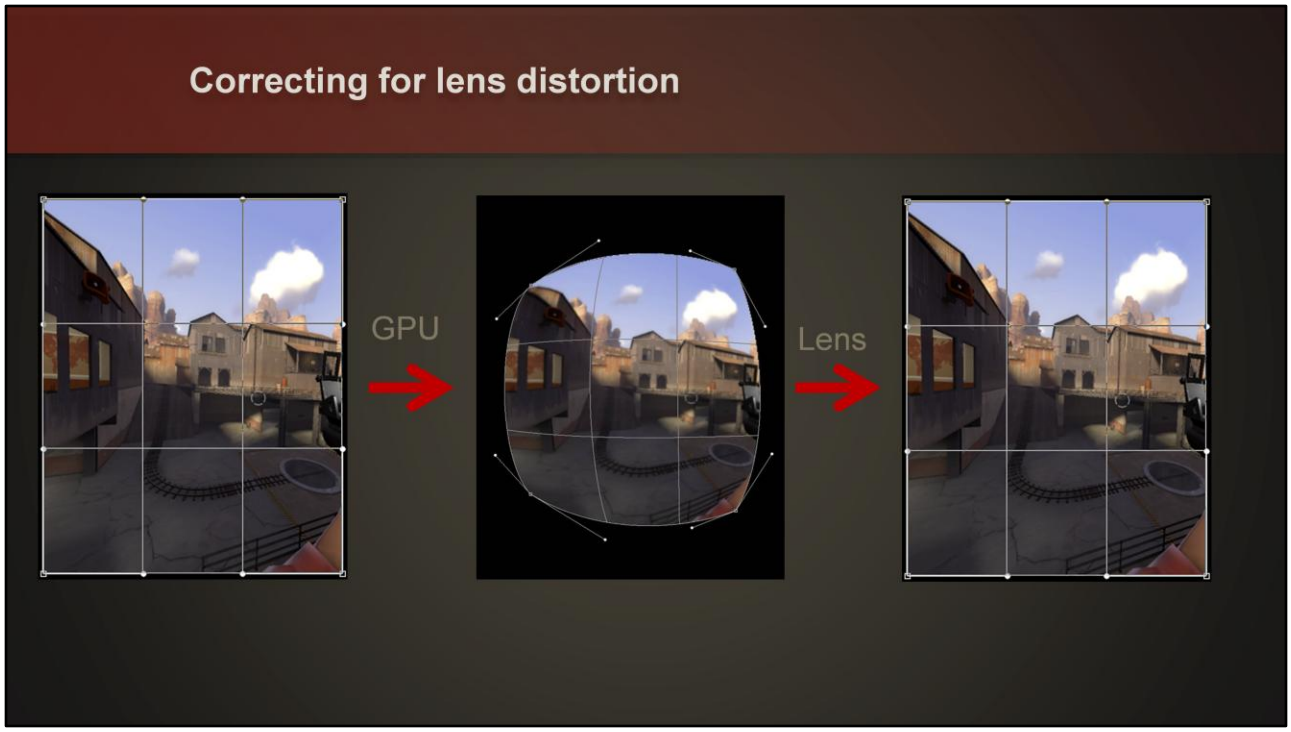
Another thing you may have to deal with when rendering in stereo is player weapon models if your engine draws different custom models for those. In the case of TF2 the player weapon models end at about the elbow. That caused problems for some models just because of the larger vertical field of view. Also, several of the input modes we tried result in the weapon moving up and down within the player's view, so this exposed the edges of the model to the player. In VR mode in TF we don't use these custom models at all.

Instead we draw the third person model (with its head removed) under the user's view position. As the user moves their head around the model is moved around to match so that the weapons are pointed in the right direction.



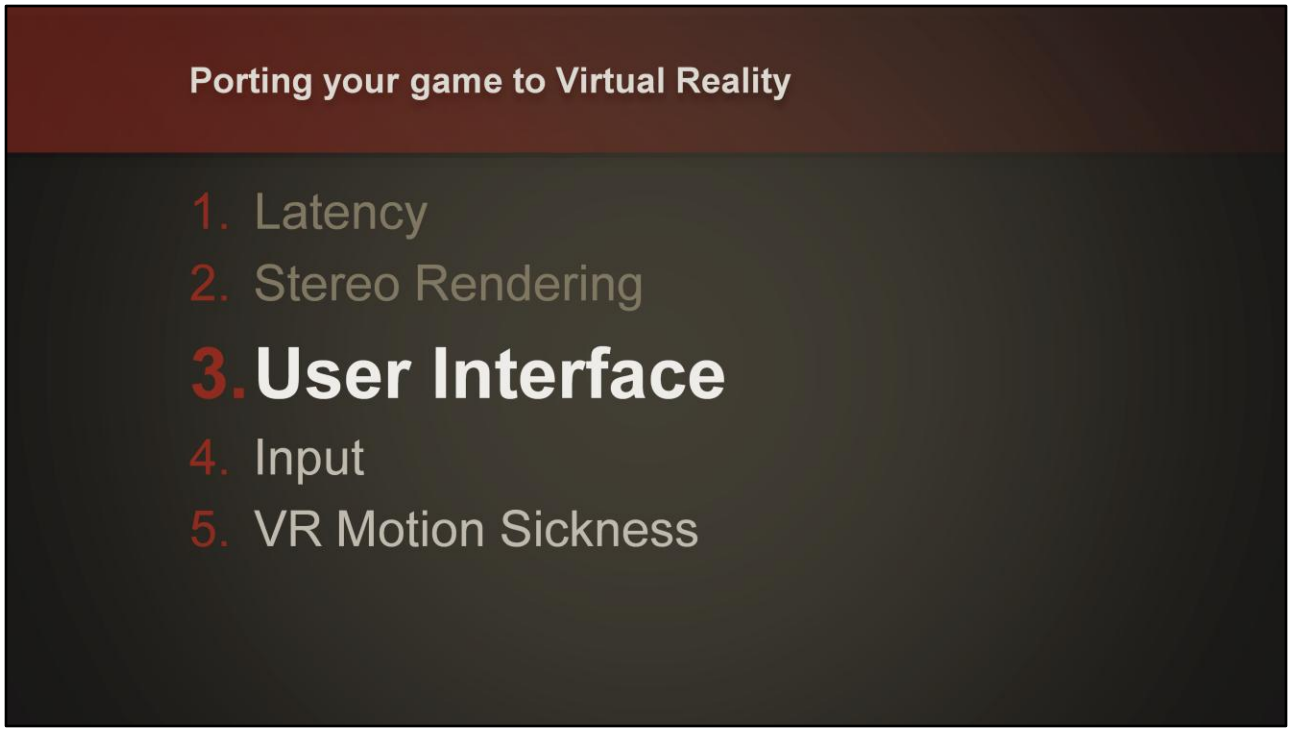
One last thing I want to mention before we move on to UI…

Unless you pre-distort the images you're displaying in the Rift they are going to come out looking something like this. The Lenses in the Rift apply significant pincushion distortion.

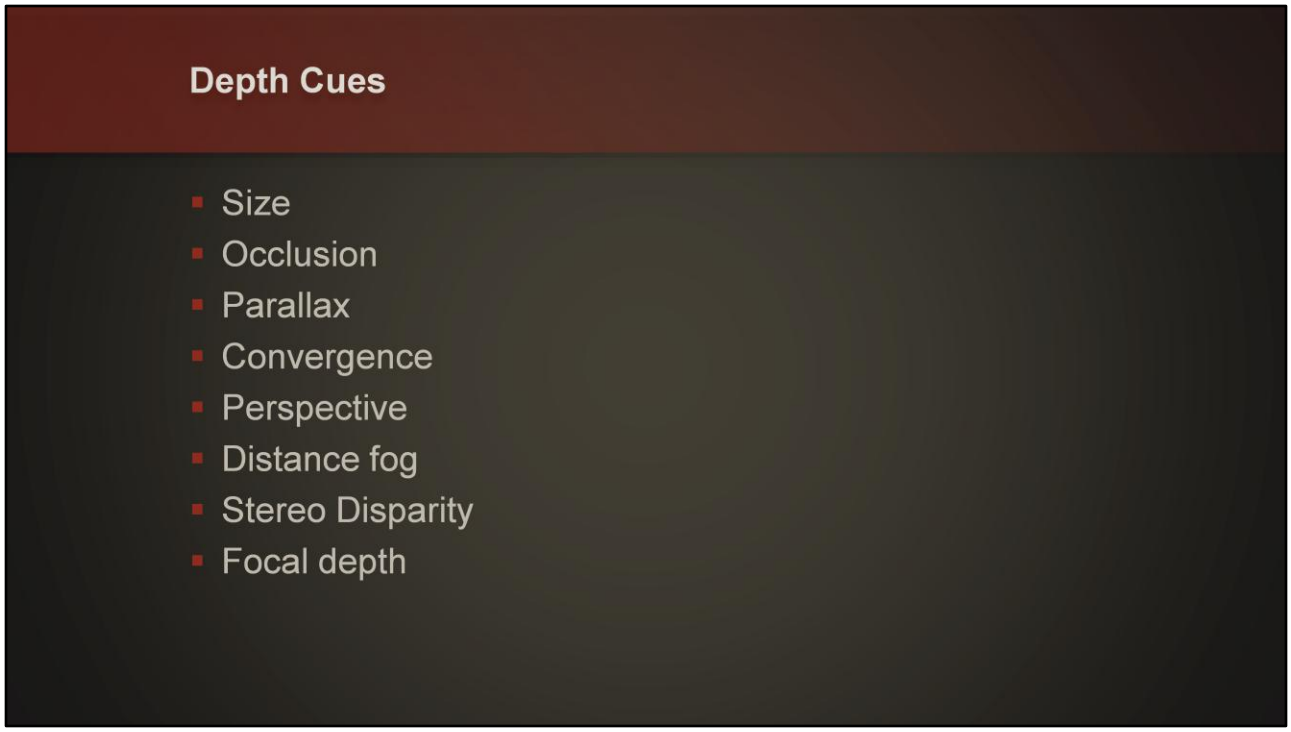


The way to deal with that is to apply a barrel distortion to the image. Then the lenses will back out the distortion you added and your lines will be straight again. The SDK contains the necessary shader code to do this, and will report the values to plug into the distortion function since they vary from display to display.

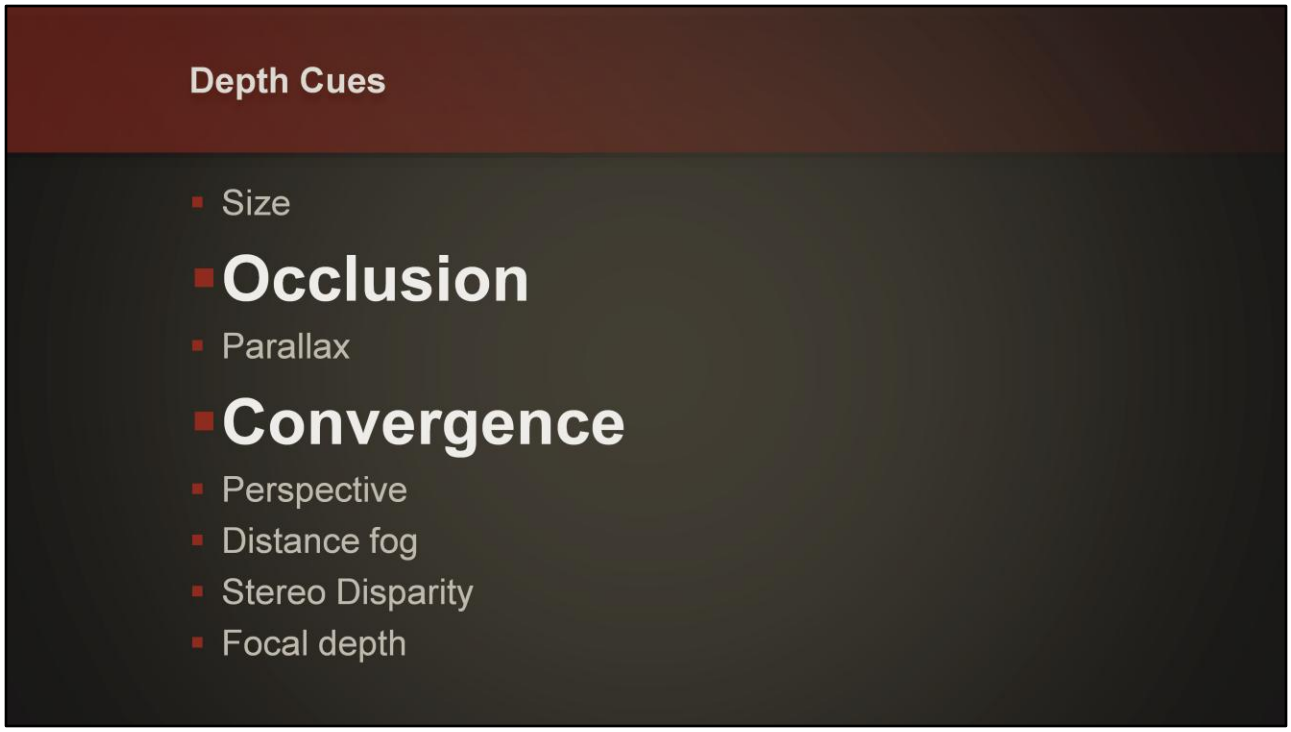
It might seem like this is a bug, but it's actually a huge feature of the Rift. One of the things that held VR back in the 90s was the difficult optics involved in letting your eye focus on a large fov display that it's so close to. Modern video cards are more than capable of dealing with this in the GPU. We can trade significant weight and complexity in the optics for a little more software, which is definitely a good trade.



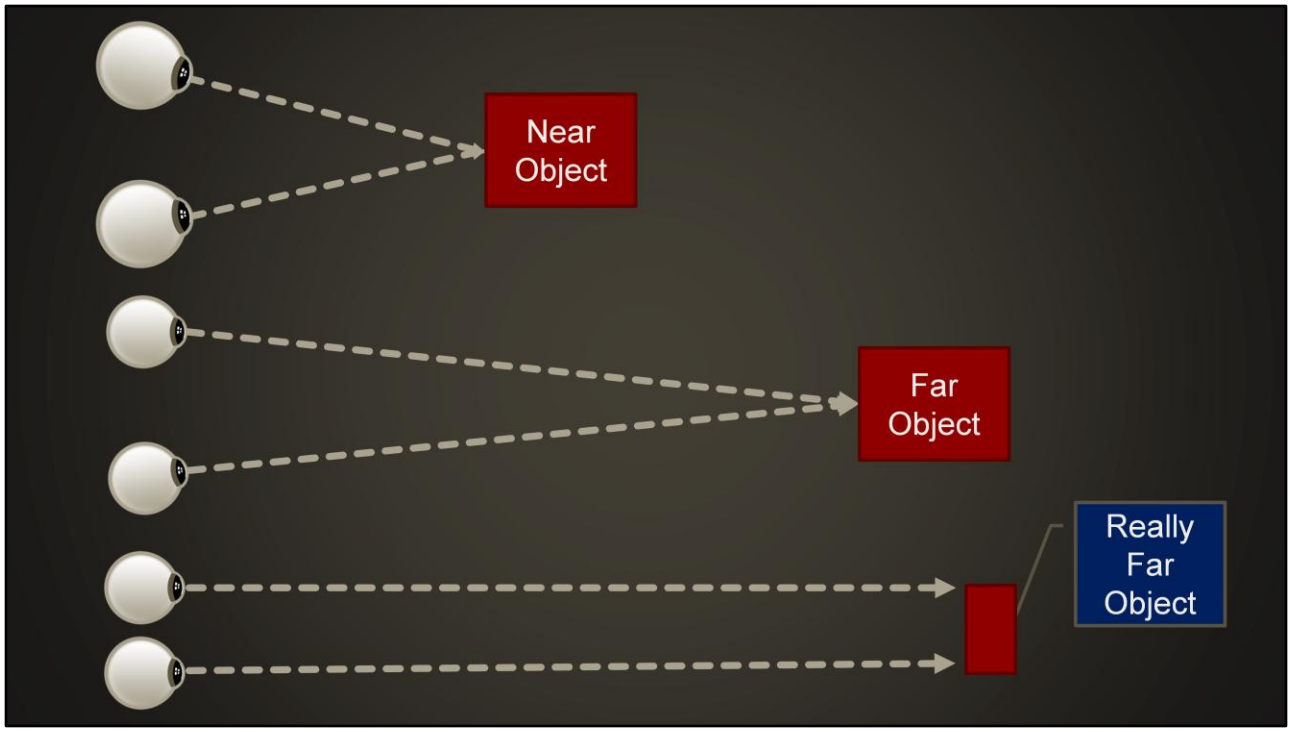
Ok, so on to UI. First up, is why this is actually complicated. You can't just draw your UI to the frame buffer at half width and expect it to work.



Your eyes and your brain use a bunch of different factors to figure out how far away something is. Here's a partial list roughly ordered from most significant to least significant. Many of these are cues that your games already provide as part of 3D rendering. I want to focus on two of these.



By Occlusion, I mean “what stuff blocks out what other stuff”. I can tell that my hand is closer than that gentleman in the 4th row because my hand is blocking out part of him.

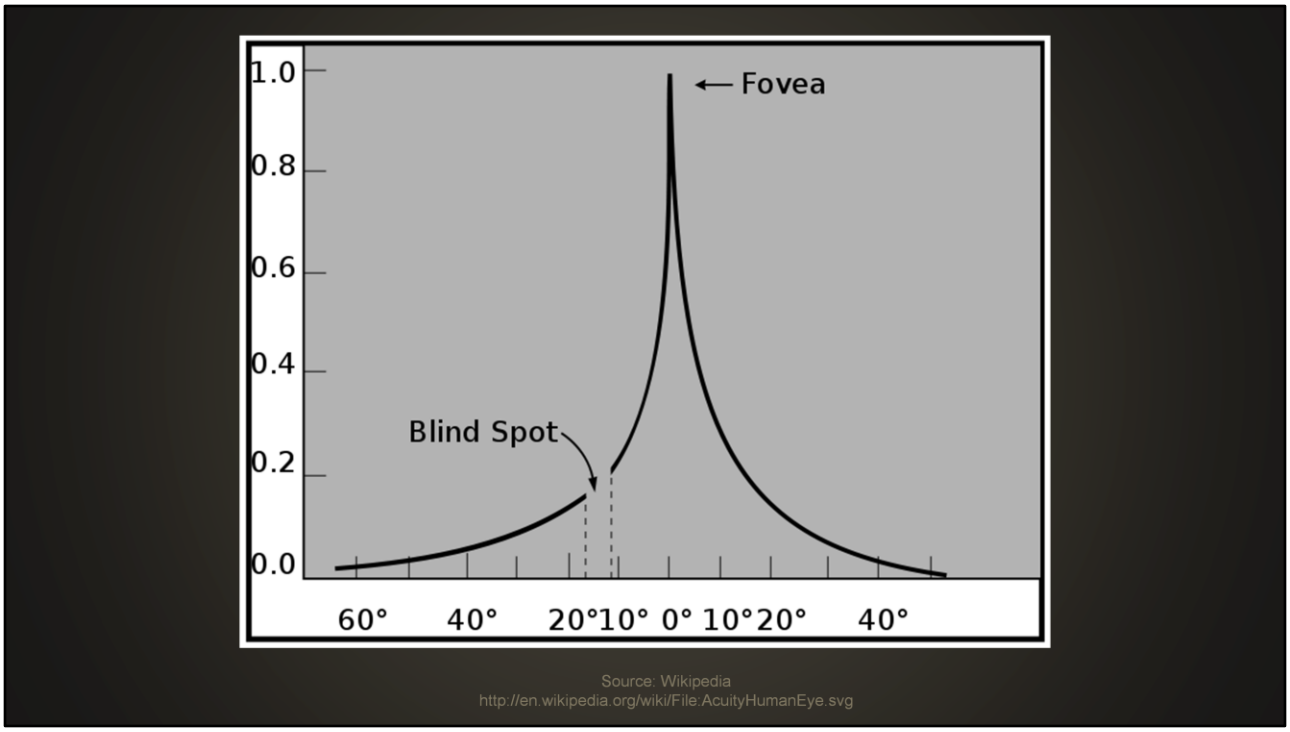


Another powerful depth cue is Convergence. Convergence is how far your eyes have to physically rotate to focus on the object in question. For near objects they rotate more than they do for farther away objects, relative to being parallel.

For objects that are really far away your eyes are basically parallel. That's what you are simulating when you show the same image to each eye.

This is why it's a bad idea to draw the UI to the views directly. When two views of an object are identical it tells your brain that they are at infinity, at least in terms of convergence and stereo disparity. Each person reacts a little different when that cue doesn't match the others (like occlusions and size), but nobody has a good reaction. Some users react badly enough that they can never actually get the two views of the object in question to "fuse" and be treated by their visual system as one object. At the very least you're going to cause a bunch of eye strain and headaches if you do this.

The other problem is the distortion shader and just the layout of pixels on the display in general. The outer edge of the viewport is actually entirely hidden from the user. Then there is a band that is significantly warped which causes visible filtering artifacts. What you really want to do is get your UI into the least warped part of the display, which is in the center.



It's also easier to see the middle of the display than the edges. The center couple degrees of your vision fall on a part of your retina called the "fovea". About half the information collected by your retina is collected at the fovea. If you want somebody to be able to read something you need to either make it gigantic or put it where the fovea can reach it.

Nobody finds it comfortable to turn their eyes far enough to aim their fovea at the edges of the Rift display to read some bit of HUD.

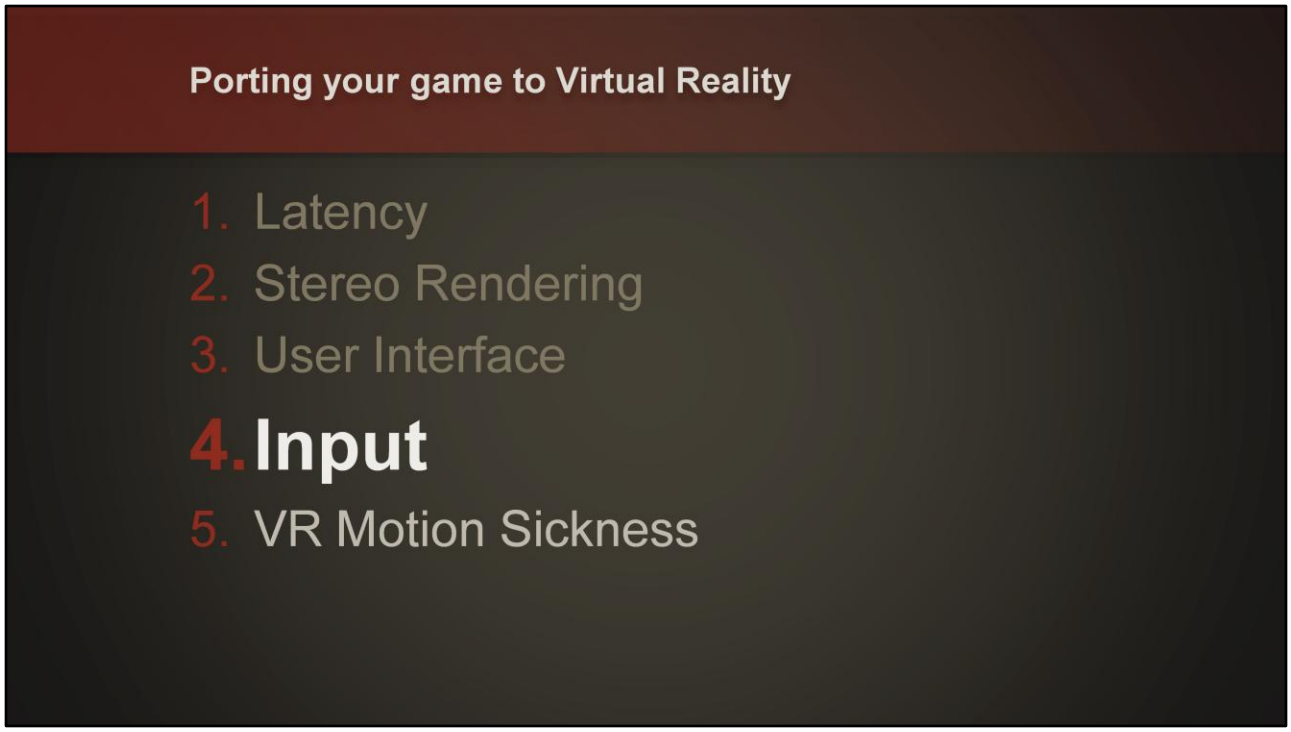
So the HUD is a pain. If you can get away with no HUD at all, that's probably your best bet. I expect most new-for-VR games will go this way.

Unfortunately that wasn't an option in TF2, so here's what we did. We render the HUD onto a render target then draw that texture on a quad that sits about 10 meters in front of the user. The quad itself is about 60 degrees wide in terms of FoV. There are still depth cue conflicts at that depth, particularly with the nearest part of the floor and the player's weapon. Fortunately that seems to resolve them enough for most players to be comfortable, though. Even those numbers are debatable, though, so they are configurable by ConVars, and we hope to learn what they ought to be from users.

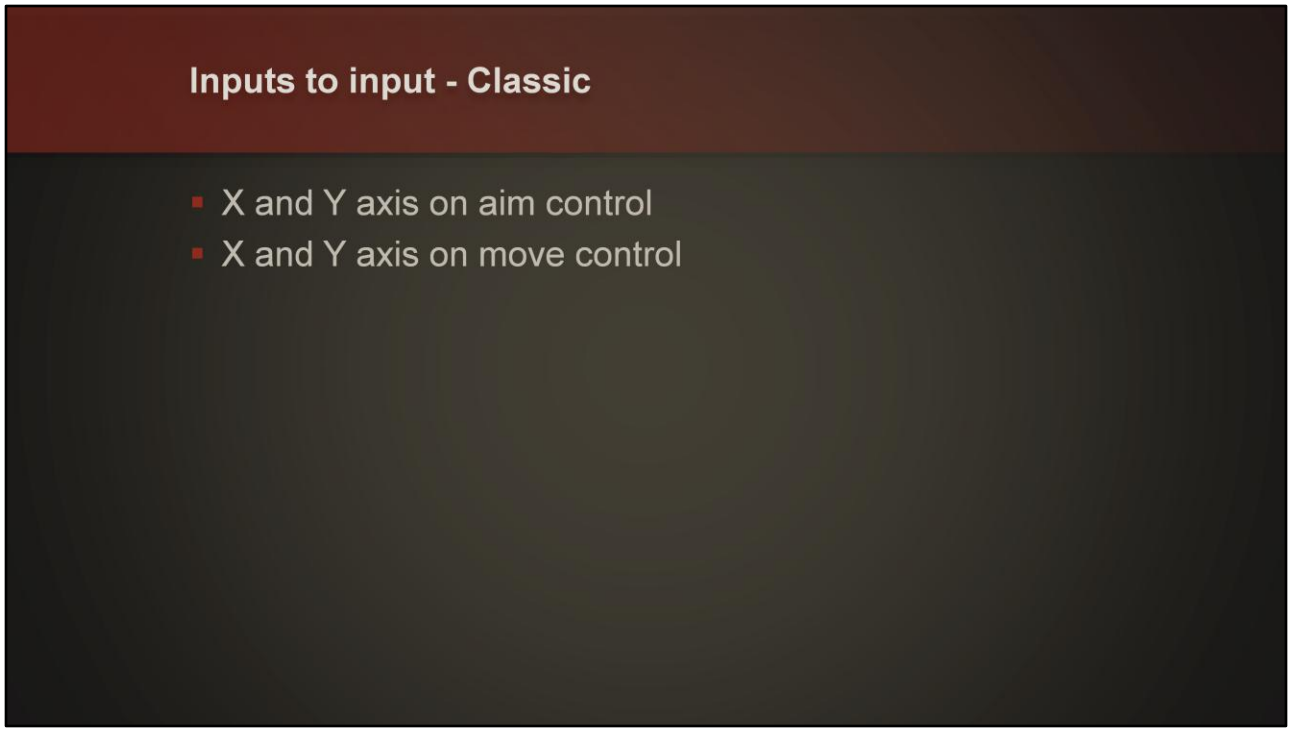
In our case we also draw the non-HUD UI on that same quad, which seems to work fairly well.

One exception to all of this is the crosshair. This we do draw directly into the frame buffer, but to figure out its stereo separation we cast a ray into the world and see what you'll hit. Then we draw the crosshair at that depth. This is a little hard to demonstrate with images… you should really see it in the game to see what I'm talking about. The key thing is that the edge of the crosshair is always going to line up in both eyes with the edge of an object near the aim depth.

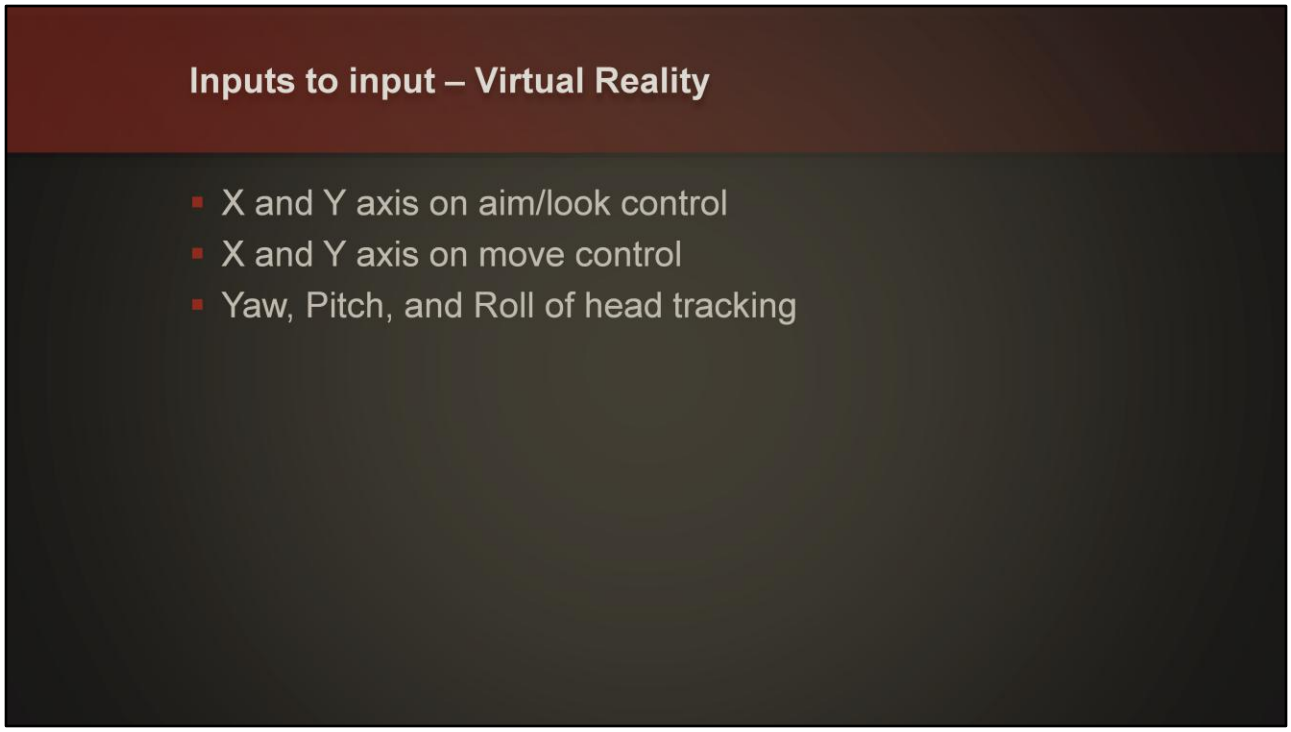
It's a little odd to see the crosshair depth change, but it avoids convergence mismatch and eye strain on the UI element that you spend the most time looking at.



Next up is user input.

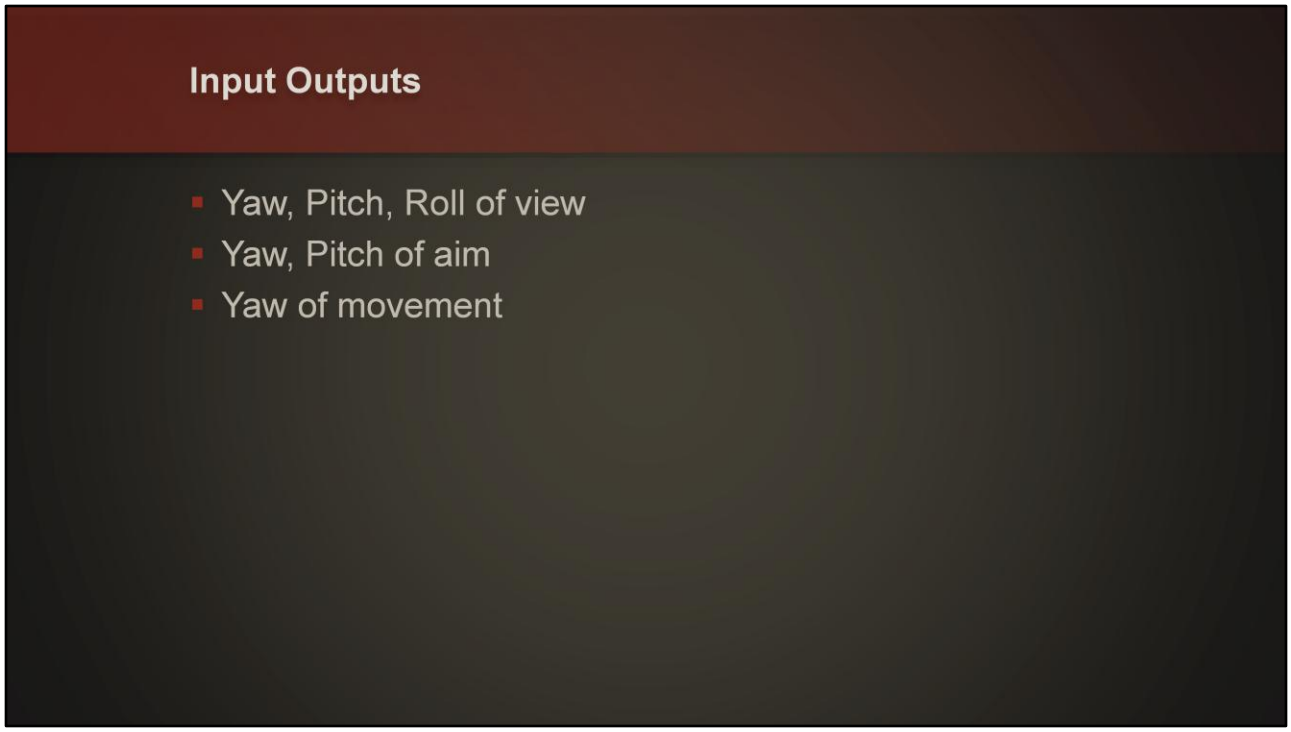


Most first-person shooter input boils down to just a couple of things. You have one 2D device (like a mouse or the right thumbstick) to control your view. Then you have another 2D device (like WASD on your keyboard or the left thumbstick) to control how your character moves.



VR complicates this by adding three more axes of input from head tracking. The yaw, pitch, and roll you get from head tracking need to be folded in with your existing view controls to let the user tell you how they want to control their character. This complicates both aiming and looking significantly because the two are no longer directly tied together.

Another complication is the cord on the Rift itself. If you don't give the user some way to turn around with having to turn their heads, they are going to wrap themselves up in the cable and not be able to turn any more.



Those 7 axes of input need to map to six axes of output. One of these is pretty simple… you need to figure out which direction you're going to move when you hit W. Users expect strafing and reverse controls to be derived from the forward control, so there's really only one output for those two axes.

I also skipped Roll on the aim direction here. Maybe you have some weapons that aren't radially symmetric in your game, but we didn't have to deal with that in TF2 so we didn't spend any time thinking about it.

One last thing before I get into the input modes we built: in our case, pitch and roll of the headtracking input are always mapped 1:1 to pitch and roll of the view output. And yaw is always just the "torso yaw" plus the head tracking yaw. Any time we messed with headtracking we made people sick, so we don't do it anymore.

Input Mode 0

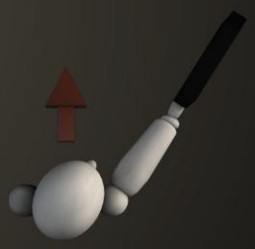
- Torso Yaw controlled by Mouse X
- Look YPR = Torso + Head
- Aim YPR = Look YPR
- Move where you're looking

The first mode we built for VR was pretty simple. Torso yaw rotation was controlled with side-to-side motion on the mouse. Torso Pitch and Roll technically exist, but they're always zero in all of these modes. On top of torso rotation we add data from the head tracker to get the view and aim angles, which are the same angles. You are essentially aiming with your nose.

This worked ok, but we heard a couple complaints: One was that it's actually pretty difficult (and tiring) to aim by moving your head.

Input Mode 1

- Torso Yaw controlled by Mouse X
- Look YPR = Torso + Head
- Aim YPR = Look YPR
- *Move with torso yaw*



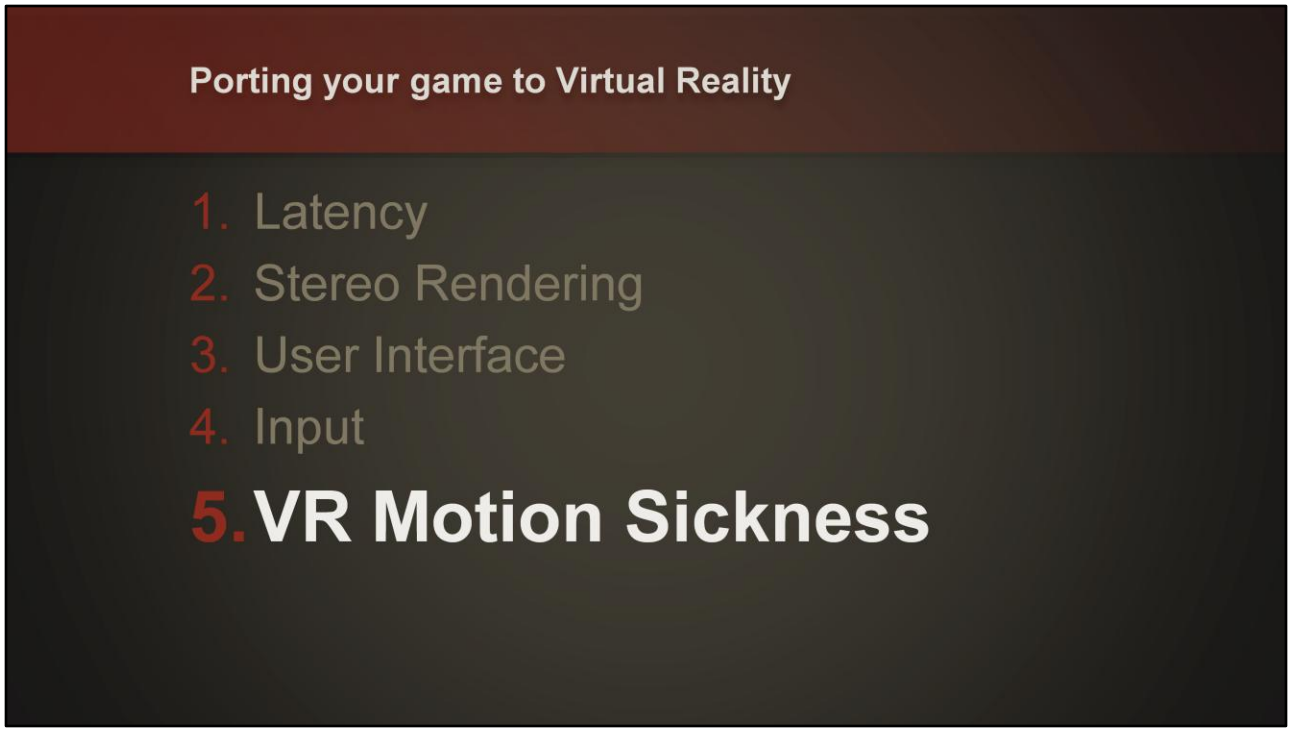
The other complaint about mode 0 was that some people found it strange to press W and not move in the direction that their torso was facing. It was of that complaint that mode 1 was born. Torso yaw was still mapped 1:1 to the mouse X axis. Aim and look are still the same angles and still Torso+Head tracking. The difference is that you move in the torso direction instead of the look direction. To help you keep track of where your torso is, we also displayed the HUD in front of your torso so you could look away from the HUD.

This all worked well enough with some of our other tracking systems, but as of right now there is some yaw drift on the Rift dev kits. That means your torso and head drift apart and eventually you can't see the HUD or "forward" direction anymore.

Mode 2 was an attempt to address the aiming precision issue. This was the first mode that involved unlocking the aim point from the center of the display. You can move your aim point freely up and down in mode 2 as long as you keep it on the screen. If you move left and right in the middle blue zone you do move your crosshair, but you also apply 50% of that rotation to the torso. If you push against the edge of that 15 degree region you are moving the torso 1:1 with the mouse X axis. This mode also works pretty well, but people complained that the view dragged the crosshair around and changed your aim point. It also has some bugs that made head tracking not match exactly in yaw.

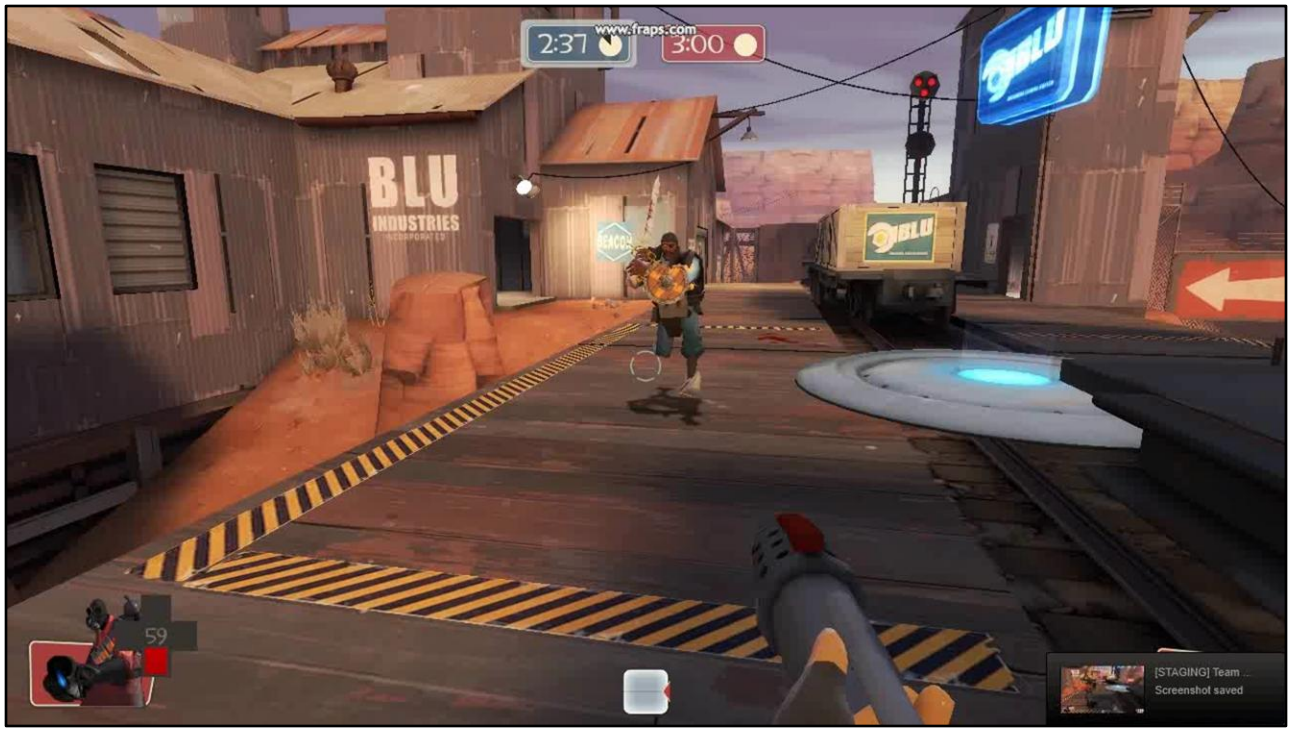
From those complaints, mode 3 was born, and that's what we're shipping as the default. In mode 3 you can move your crosshair freely within about a third of the screen. If you push against the edges in yaw you will rotate your torso. If you bump against the limits in pitch, you just can't aim any higher or lower. The aim point is also completely unaffected by head movement, so if you are aiming at something you can look away and look back and your aim point will not have changed. Mode 4 is just mode 3 but with your "move forward" direction set to your aim point instead of your view point.

While we are happy with mode 3, we're under no illusion that it is the final answer input. VR is at the same point that first person shooters were before everyone standardized on mouselook. It's likely to take a few more years to settle on a standard.



And now we come to the part of the presentation where we talk about motion sickness. Hopefully none of you have sensitive stomachs.

VR motion sickness is a lot like seasickness in a couple respects: The first is that the symptoms are similar and include headaches, nausea, and in some cases cold sweats. The other is that the motion sickness induced by VR seems to be something that people get used to over multiple days in the same way people get their sea legs.



This clip actually demonstrates two things you should not do…

Don’t change the user’s horizon line, ever. You can see here how the camera follows the motion and rotation of the character’s head and so it rolls. Your actual head isn’t going to roll when you get killed by an Eyelander, so the mismatch will make you sick.

The other thing this clip shows is the freeze cam. When you die in TF2 you get a nice shot of whoever killed you. This causes significant problems for some people. All we do is show the same image on the screen for a few seconds and ignore head tracking, but some people are convinced that we’re actually moving the view backwards. It was very strange.

These are specific cases of a more general rule: Don’t mess with headtracking. If the user turns their head 27 degrees to the right and rolls it 3 degrees their view in the game needs to turn 27 degrees to the right and roll 3 degrees. Anything else is going to make people sick.

This video can be found here: http://www.youtube.com/watch?v=LPtHCeb_HPc

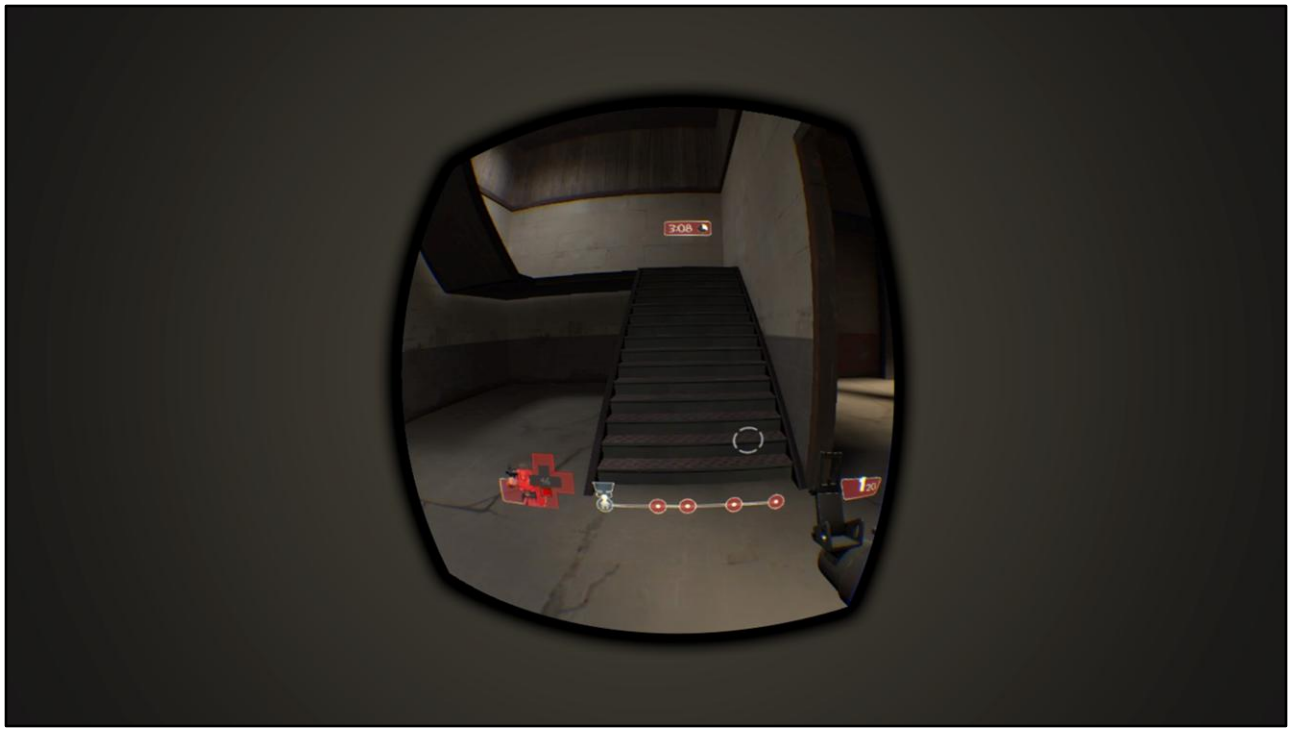
In fact, involuntary motion of any kind seems to be bad. We had a bug for a while that slid the camera to the side while in spectator mode, and for 25-50% of players that was another wave of motion sickness.

The exception is moving the camera in the direction that it is pointing. Once we fixed the spectator camera so the player was moving forward they had a much easier time. And that same motion seems to be fine with just about everyone when they're the ones pressing 'W' to move forward.

There are a few things that happen in TF2 that trigger motion sickness, but aren't things we can remove for gameplay reasons.

Rocket Jumping, for one. The fact that you're looking down when you do it seems to be part of the problem. People don't seem to have the same issues with Sticky jumping.

Then there's the Scout. He runs at something like 22 miles per hour, and moving around at that speed causes issues for many people. TF2 is a game with a frenetic pace in general. Many of these issues would be less prevalent in a slower paced game.



The one that surprised us the most is stairs or ramps, which is what stairs in TF2 actually are. Some people get a wave of motion sickness whenever they move up or down stairs in the game. We think this is related to the “move forward” rule. When a player runs up stairs they’re actually moving up and forward at the same time, and those two can be significantly mismatched if they are actually looking down.

So that's most of what we learned with TF2. There is still plenty to figure out, though.

For instance, how will pulled back ¾ view games like Dota work in VR?

How will third person avatar games (like Arkham Asylum) interact with head tracking?

We're just getting started on figuring out what VR can do, and how to interact with it. I'm excited to see where we end up.

Questions? joe@valvesoftware.com

- Eliminate latency
- Buy a splitter
- Fix your screen-space effects
- Fix your player weapon models
- Pre-distort in a shader
- Eliminate the HUD if you can
- Draw the HUD in stereo if you can't
- Draw the crosshair at the aim depth
- Include a way to turn around on the mouse
- Give people some aiming without head motion
- Don't mess with the horizon. Ever.
- Keep view rotation 1:1 with head tracking
- Don't slide the camera sideways