



JOE LUDWIG  
VALVE

VIRTUAL REALITY AND STEAM

Hi everybody

For the last half hour of our VR Mega-session I'm going to tell you about what we're doing with VR in Steam.



# VR and Steam

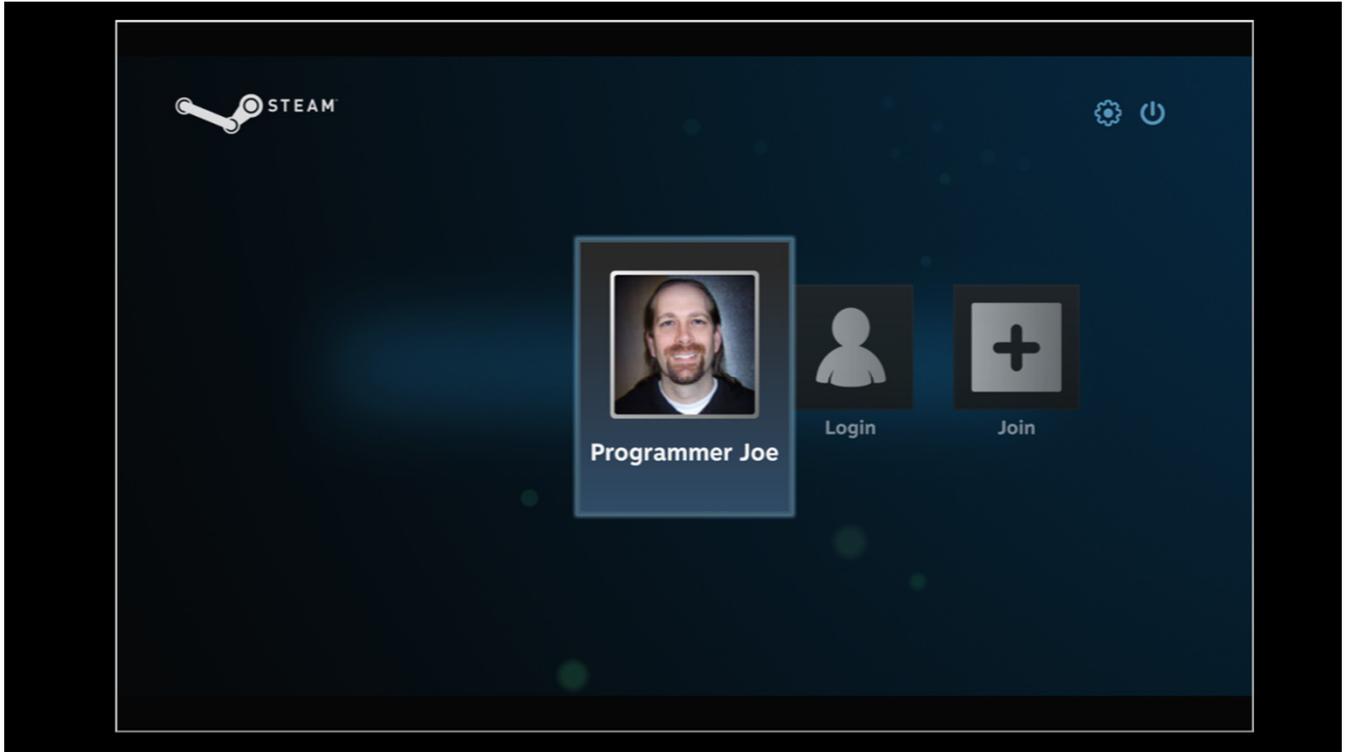
- What are our goals?
- What exist and what does that mean for you?

I'm going to talk about our VR work in Steam in two phases. The first is to describe our goals and where we are headed. Then I will go into detail on where we are on our work against those goals and what it means for you.

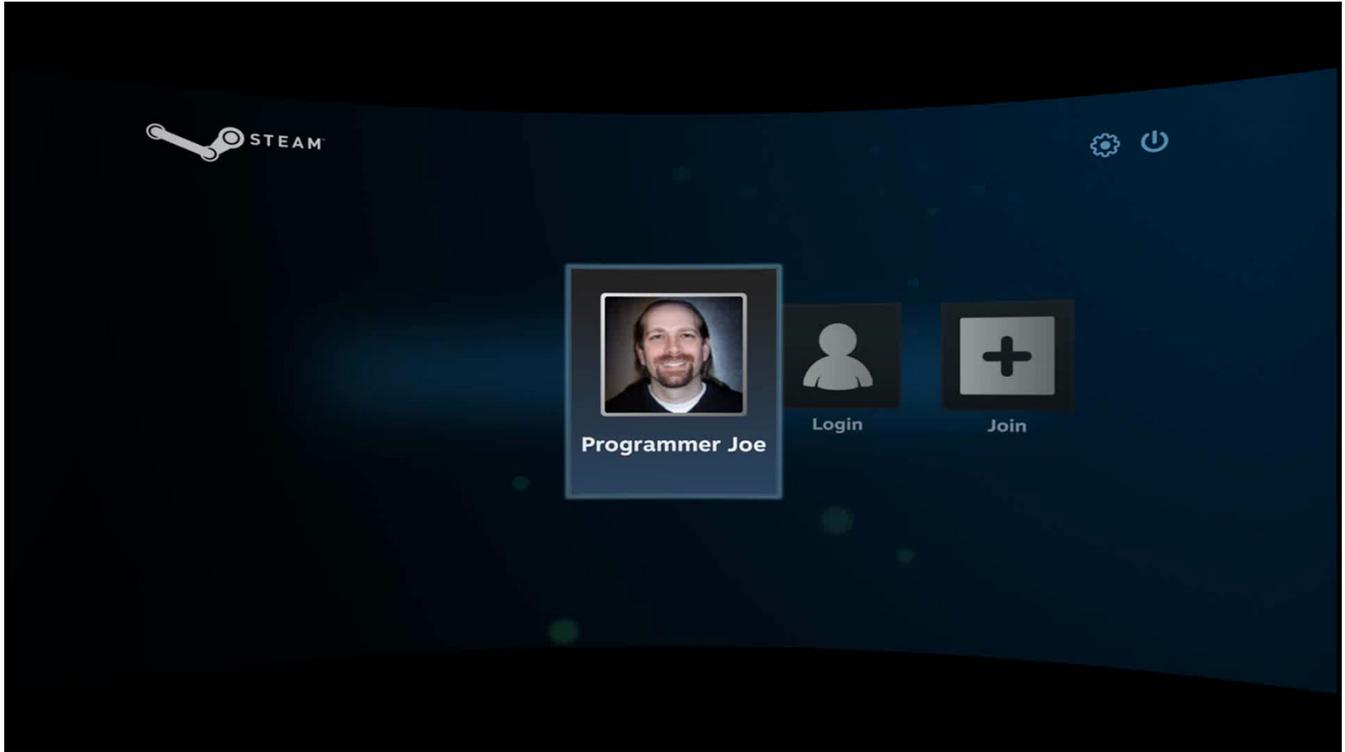


But I will start with where we are headed. Our goal is to provide a seamless VR experience while running in Steam.

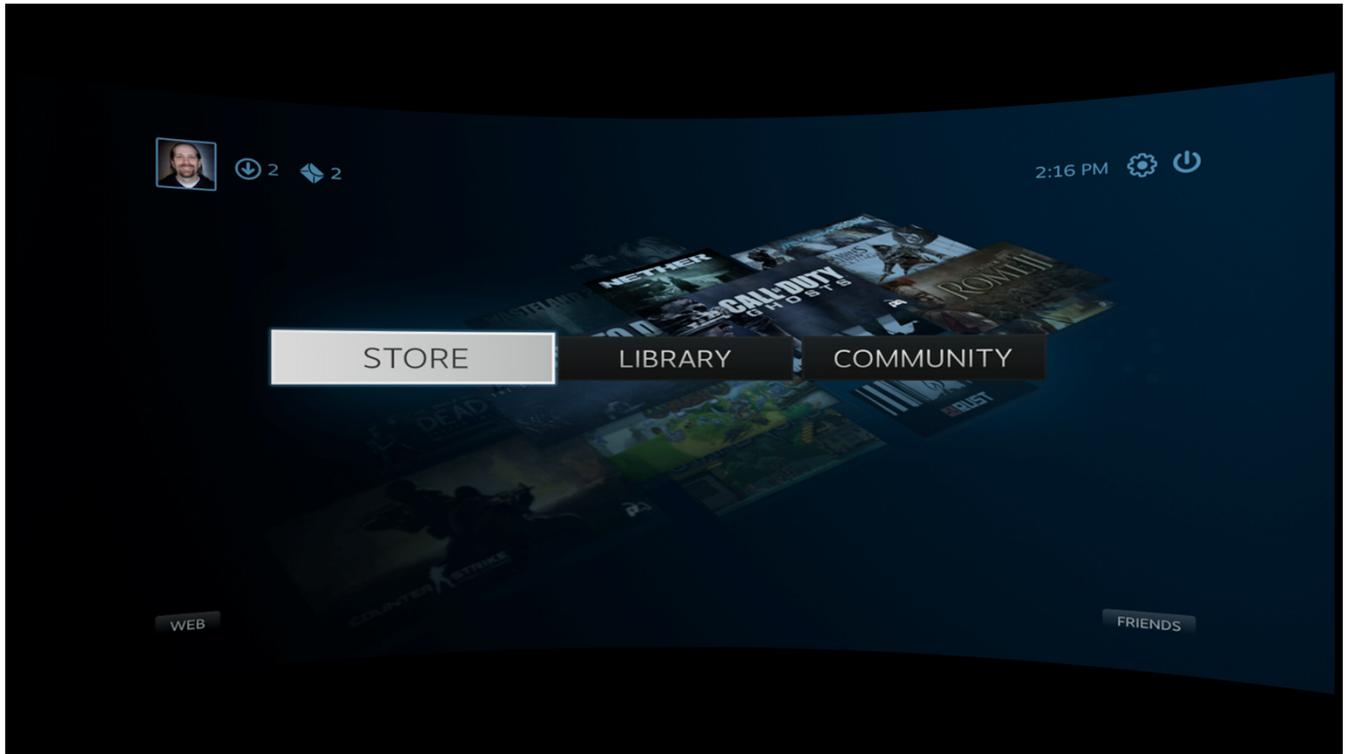
The first step is to launch Steam and put on your display.



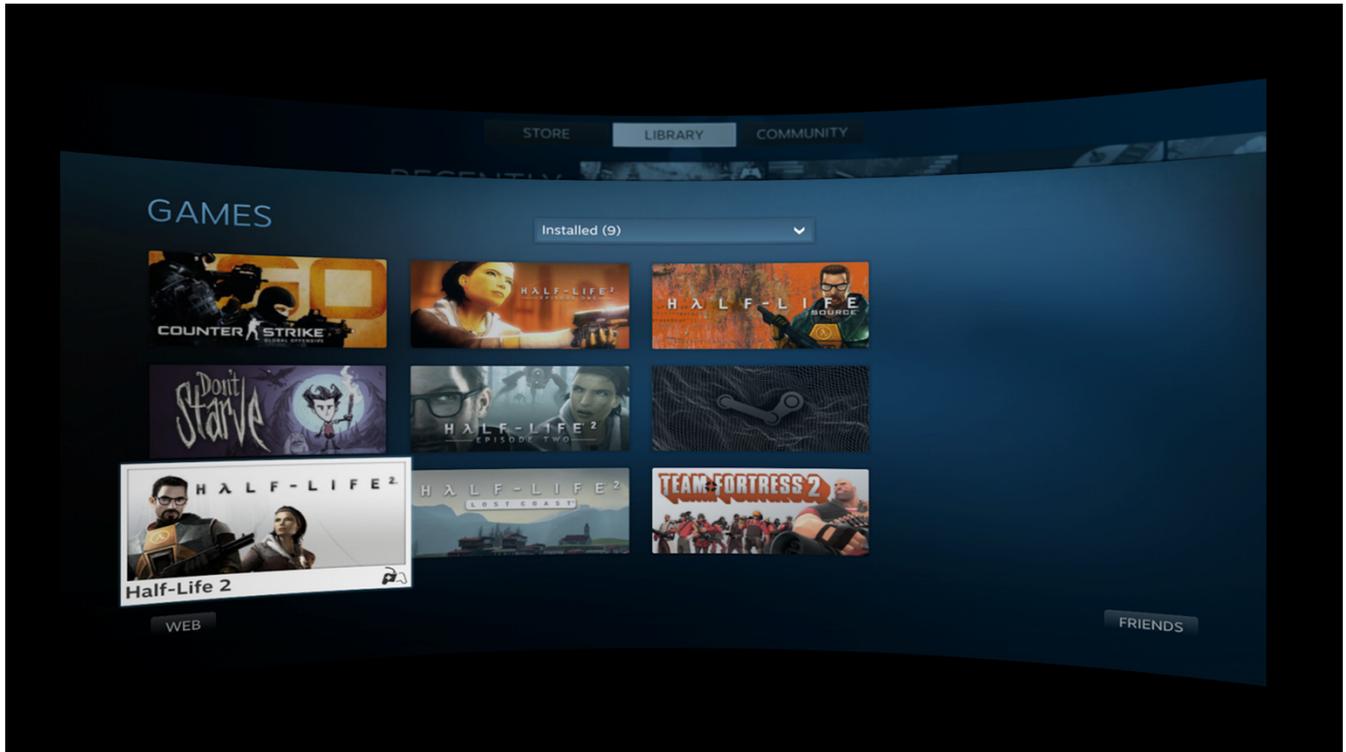
First stop is the login prompt. VR mode in Steam is the same UI as Big Picture, just wrapped onto a curved virtual screen that sits several feet in front of the user.



This screen is at a fixed point in virtual space. If you are facing forward it is centered in front of you. When you look around the virtual screen rotates to match.



Once you login you can do anything that you can do in Steam.



In the library it's easy to filter down to games that support VR and then launch one of those games.



Now, many games that support VR don't necessarily support it for every part of the experience. It turns out that Half-Life 2 is one of those games. The bits of code that understand VR don't run until after the splash screen goes away.

So for those games, Steam will show whatever the game is drawing on the same virtual screen. We are calling this "Legacy Mode", and in theory you could play an entire game in this mode using Steam's upcoming streaming feature.



As Half-Life 2 works today, the game will start up in “desktop” mode, with an option on the main menu that switches to VR.



Instead of doing that, though, the game asks Steam whether it is already running in VR and switches VR mode on automatically.



From there it's the game that is out there already. Most of you have probably already played VR mode in Half-Life 2, but if you haven't go check it out.

Fighting manhacks and playing catch with D0G are a totally different experience than doing the same things on a screen. They are definitely worth trying.

While you are playing catch, you can see incoming chat messages from your friends.



You can also bring up the Steam Overlay, which appears on the same sort of curved screen layered over the game. All the features of the overlay work in VR mode.

In other words, our goal is to make VR a first-class platform feature on Steam and provide users with the best possible VR experience in the process.

# NICE MOCKUPS. WHAT ACTUALLY WORKS?

So if that's what we are shooting for, what of that is actually finished. You may have noticed that every image I've shown you up to this point was a mockup. That's not an indication of what works and what doesn't. It's just that VR screenshots with the distortion turned on and rendered in stereo actually make it difficult to see much of anything when you put them on a slide.



Steam Big Picture in VR Mode

Here is Steam Big Picture running in VR mode. The basics are all working now, though there is still work to do.

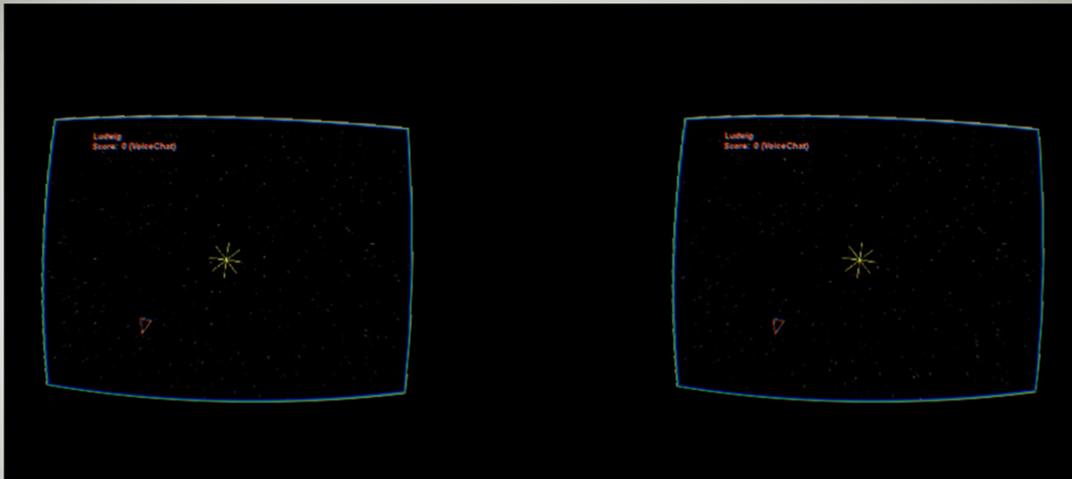


Steam Overlay and Half-Life 2 in VR

And here is the Steam Overlay running in VR Mode on top of Half-Life 2.

A beta version of both of these shipped in the Steam Client Beta that came out last week. They are not entirely where we want them to be, but we would appreciate your thoughts. Just run Steam with `-vr` on the command line to try them out.

Please tell us what you think. These are both in a very early state and we need your feedback to make them better.



Steamworks Example (aka Spacewar) in VR

Even more important than either of those is that the Steamworks example game, Spacewar, runs in VR now. If you are looking for an example of how to use anything I'm about to talk about, this would be a good place to look.



# VR and Steam

- What are our goals?
- What exist and what does that mean for you?

So now that you know what we want to accomplish, let's talk about what exists today and what you can do with it.



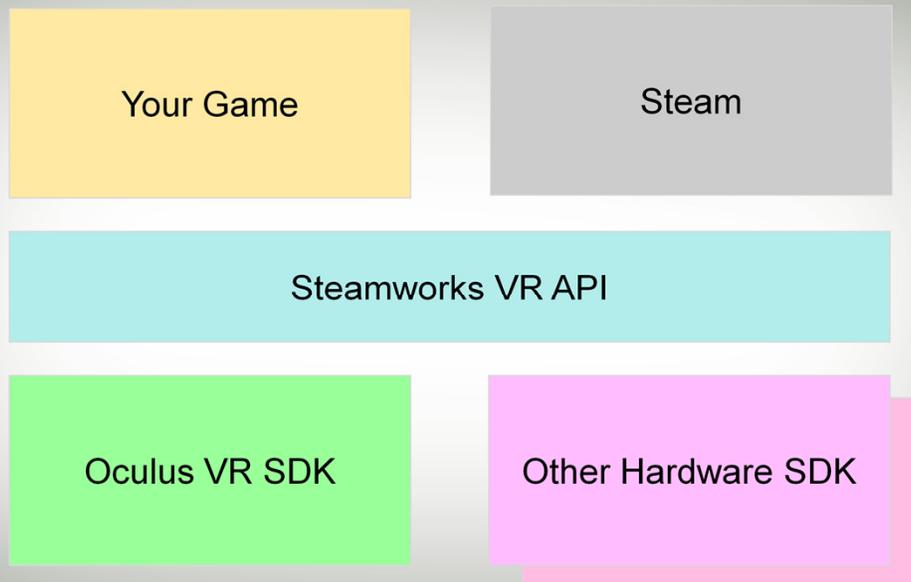
# VR Mode in Steam

New ISteamUtils member:

```
bool IsSteamRunningInVR();
```

That same client beta from earlier this week also includes a new Steamworks call that a game can make to determine if Steam is running in VR mode. If your game is anything like our games and supports a user-switchable VR mode you should make this call and automatically activate your VR mode. That is an important part of providing a seamless experience to users.

Just look at ISteamUtils for the IsSteamRunningInVR() method. We will release a new Steamworks SDK within the next few weeks that includes this call and the other stuff I'm about to talk about. Keep an eye out for that.



The other thing that makes all of this work is the Steamworks VR API. This is a new addition to the Steam platform that we have written to expand on the services provided by the Oculus SDK and other hardware-specific APIs.

The Oculus SDK as it exists today is essentially a device driver. It allows one process to connect to a piece of hardware and provides all the services required to make that work.

The Steamworks VR API expands on those capabilities by adding platform-level features that apply to all hardware device drivers.



# VR in Steamworks

## Future Proofing

- New Hardware
- Updated Software

## One Device, Multiple Apps

We decided to build this API because we felt like we could help VR emerge more quickly by addressing a couple of issues. We aren't shipping our own hardware, but providing software that is used by all of you is something that we have some experience with, so that is where we've put our energy.

The first of those issues is making your continue to work well in VR months or years from now. That includes both support for new hardware and software updates.

The second issue we are addressing with the Steamworks VR API is that unlike every hardware-specific SDK we have encountered, it allows multiple apps to talk to a single piece of hardware at the same time.



# FUTURE PROOFING

Future Hardware

Future Software

I'll start with future-proofing.

If you don't have a team providing ongoing support for your game, this is very important. You need support for new hardware, software updates, and more to keep your customers happy, and Steamworks can provide that.

Even if you do have a live team, using the Steamworks API lets your team focus on the game and not worry about ongoing VR support.

# Future Hardware



The biggest kind of future proofing that the VR API in Steamworks helps with is new hardware. These are some of the HMDs and tracking systems we've been able to run our games on over the past year and a half. At the moment none of these is consumer ready, though the Oculus Rift is the closest. That is all going to change over the next year or two, though. A number of other VR displays have emerged in the last few months.

If you use the Steamworks VR API you will get support for any of these devices that catch on without making any changes to your game.

Of course it is likely that eventually new hardware will include features that aren't in the API yet. When that happens we will rev the API so new games can take advantage of the new features. However, older games will continue to work without modification like they do with other Steamworks features.



# Future Software

- Improved calibration
- Improved filtering for the tracker
- New platforms

Even without new hardware, though, there are still reasons to write to Steamworks instead of directly to a hardware vendor's API, and that is software updates. All sorts of features can be enabled or improved on a bit of hardware by upgrading its driver.

Unfortunately moving to a new driver for the Rift currently means linking against a new version of a static lib. Even if the API were in a DLL, the interfaces aren't versioned or backward compatible, so it wouldn't be possible to swap out that DLL with a new one.

When you use the Steamworks VR API we can update the implementation without requiring you to ship your game again.



# ONE DEVICE, MULTIPLE APPS

There is also another, more immediate, reason to prefer the Steamworks API. It is often the case that multiple apps need access to the hardware at the same time.

# Traditional Shared Hardware



This is a common problem with hardware devices. For many devices like keyboards, mice, and monitors the operating system provides standard APIs and abstracts away the hardware completely.

For video cards it's the job of OpenGL or DirectX to manage access to the device.

# Shared VR Hardware



The Steamworks VR API fills the same niche for accessing the VR hardware that OpenGL does for video cards. And part of what it does is allow any number of VR apps to get display geometry and head tracking data from a single head-mounted display.

There's nothing magical here, it's just a little IPC communication, but asking every game or every hardware vendor to provide this service is not realistic. Thanks to the Steamworks VR API when you run Steam in VR mode and then launch a VR game the game can actually connect to the hardware and doesn't just fail to init the vendor-specific APIs.



# VR in Steamworks

- **Display geometry**
- Distortion compensation
- Head tracking

So how does this API actually work? I will go through it quickly. I am assuming here that you've interacted with the Rift SDK or some other VR hardware. If not, your best bet is probably to take a look at the Steamworks example app to see how to build a VR app.

Well let's start with the first call you are likely to make, which tells you where to put your window...



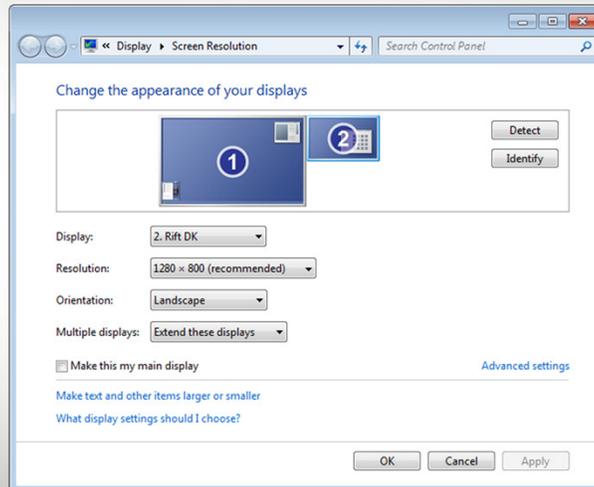
# Display Geometry

```
bool GetWindowBounds(  
    int32_t *pnX,  
    int32_t *pnY,  
    uint32_t *pnWidth,  
    uint32_t *pnHeight );
```

BTW, this is all included in the Steamworks documentation, so don't worry about writing any of it down. I'm just going to buzz through the functions in the API quickly so that you can ask questions while we're still here at Dev Days. If you run into problems hooking any of this up, please contact your publishing tech or shoot me an email.

GetWindowBounds returns exactly that information. X and Y are in desktop coordinates and Width and Height are the size of the display.

# Display Geometry



These are the same values you set in the Screen Resolution panel on Windows or the Display Options dialog on Linux. `GetDisplayBounds` is just a way to get them programmatically in a cross-platform way.



# Display Geometry

```
void GetEyeOutputViewport(  
    Hmd_Eye eEye,  
    GraphicsAPIConvention eAPIType,  
    uint32_t *pnX,  
    uint32_t *pnY,  
    uint32_t *pnWidth,  
    uint32_t *pnHeight );
```

Next up is `GetEyeOutputViewport`. This is the area in the window that you will need to draw into to for each eye.

This is the first of a few calls that include an API Type parameter. This is either DirectX or OpenGL and just transforms the viewport coordinates into something appropriate for that platform before returning them.

# Display Geometry



For some displays you could just divide the window in half horizontally and assume that the left half is the left eye and the right half is the right eye. That might work today, but there is no guarantee that the two eyes will be positioned that way in the frame buffer for every future device. You are better off making a quick call to get the bounds of each eye.

This is what the two viewports would look like on a Rift.



# Display Geometry

```
HmdMatrix44_t GetProjectionMatrix(  
    Hmd_Eye eEye,  
    float fNearZ,  
    float fFarZ,  
    GraphicsAPIConvention eProjType );
```

You also need to fetch the projection matrix that is used for an eye. This will be an off-center projection that is appropriate for whatever HMD you have connected.

# Display Geometry



This matrix will represent a slightly wider field of view than is really visible in the final output. Rendering these extra pixels into your pre-distorted image maximizes the final Field of view for the user. The stretched projection is backed out again by the distortion function itself.



# Display Geometry

```
HmdMatrix44_t GetEyeMatrix(  
    Hmd_Eye eEye );
```

Next up is GetEyeMatrix. The eye matrix adjusts the in-world cameras for the user's interpupillary distance or IPD. This is what gives you a stereo effect when you are wearing the display.

# Display Geometry



The API knows what the user's IPD is, so all you have to do is fetch the matrix and multiply it in between the view and projection matrices.



# Display Geometry

```
void GetRecommendedRenderTargetSize(  
    uint32_t *pnWidth,  
    uint32_t *pnHeight );
```

The final display geometry function I'm going to mention is `GetRecommendedRenderTargetSize`. This function tells you how big your pre-distortion render target needs to be to minimize stretching pixels. Using this size will tune your texture to match the projection matrix and distortion function.



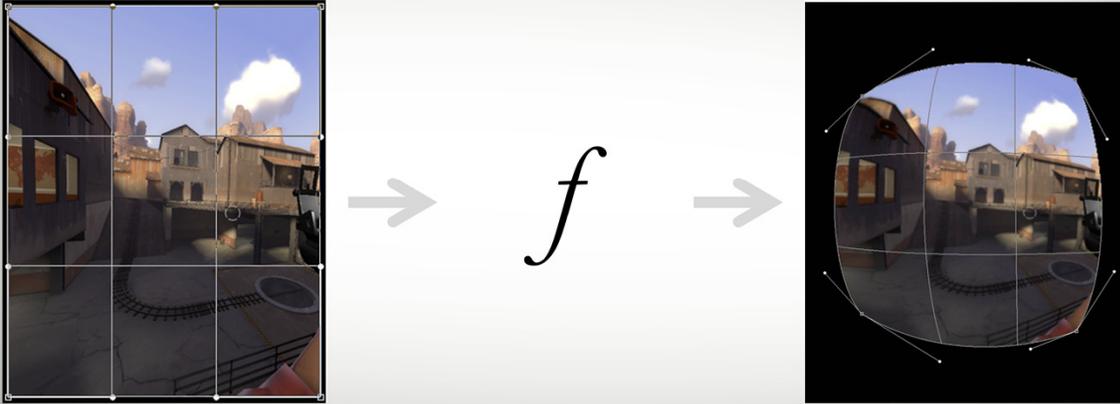
# VR in Steamworks

- Display geometry
- **Distortion compensation**
- Head tracking

There are a couple other display geometry functions, but those are all the ones that are required to get up and running. See the Steamworks documentation for details on the others.

Next up is the distortion step that is required to compensate for the distortion of the lenses in the display.

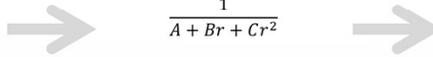
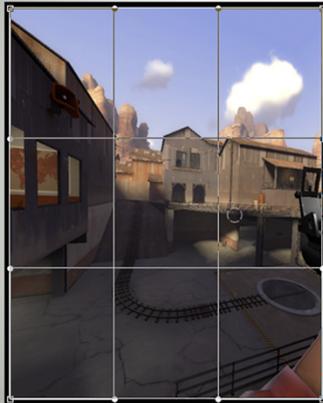
# Distortion Compensation



The way that this usually works is that you render an undistorted scene into an offscreen render target and then run that through some function in a pixel shader.

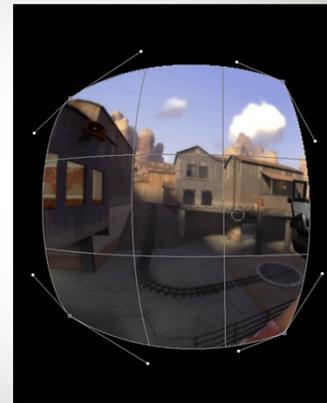
The problem with this approach is the function.

# Distortion Compensation



$$A + Br + Cr^2$$

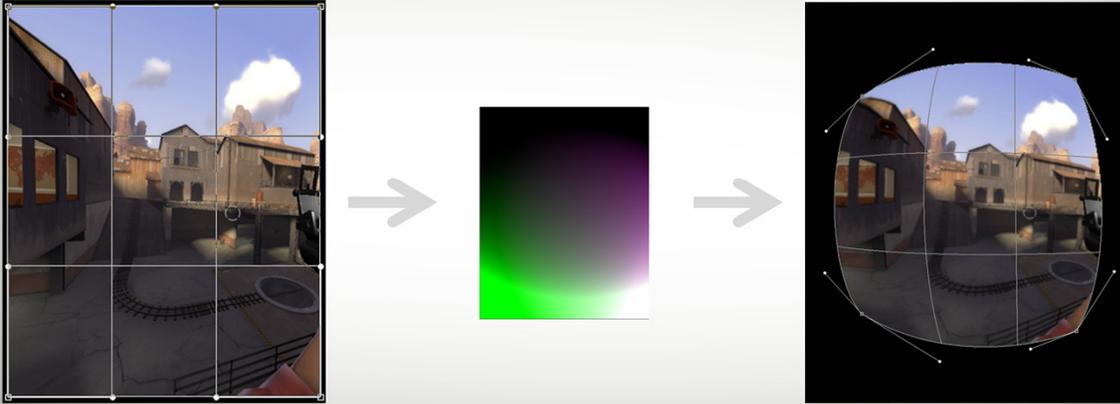
$$\frac{1}{A + Br + Cr^2}$$



$$\begin{aligned} &(2t^3 - 3t^2 + 1)p_0 + (t^3 - 2t^2 + t)m_0 \\ &\quad + (-2t^3 + 3t^2)p_1 \\ &\quad + (t^3 - t^2)m_1 \end{aligned}$$

It turns out that there are a whole bunch of them. Every combination of lens and display panel has different distortion characteristics. Most of them are radial and operate on the distance from the center of the lens, but even that isn't universal. Games usually want every shader to be something that is under their control, but expecting every game to support every function that could exist is unreasonable.

# Distortion Compensation



Instead the Steamworks VR API computes the distortion values once at startup uses a lookup texture in the actual shader. This keeps the shader very simple and supports any distortion function without any game-side changes.



# Distortion

```
DistortionCoordinates_t ComputeDistortion(  
    Hmd_Eye eEye,  
    float fU,  
    float fV );
```

The ComputeDistortion function returns the distortion values to the game. Just call it in a loop to populate your texture. We find that a 128x128 texture is plenty to represent the distortion lookup values, though you will want at least 16 bits per channel in the texture.

You can find working examples of this in the Steamworks example and the Source SDK.



# VR in Steamworks

- Display geometry
- Distortion compensation
- **Head tracking**

The last area covered by the Steamworks VR API is head tracking. There are a couple of functions involved in this...



# Head tracking

```
bool GetWorldFromHeadPose(  
    float fPredictedSecondsFromNow,  
    HmdMatrix34_t *pmPose,  
    HmdTrackingResult *peResult );
```

And the first one is `GetWorldFromHeadPose`. This function takes the number of seconds into the future to prediction motion (which should generally be the time your takes to render one frame) and returns a pose and information about that pose.

Note that this pose is a 3x4 translation/rotation matrix. Including translation support in your game is a key piece of future-proofing. The first Rift dev kit doesn't support it, they started showing off their positional tracking system last week at CES.

# Head Tracking



This returns the "head" pose, which is a point centered between the user's eyes in a right handed coordinate system with negative z facing forward, positive Y facing up, and positive x facing to the user's right.

# Head Tracking



I mentioned that the pose provided by the Steamworks VR API includes position, but didn't really get into why that's important. I mean, it's important for the totally mundane reason that it eliminates one cause of VR motion sickness, but other than that.

There is an entire class of game that is enabled by translation that won't work on a display that has only rotation tracking. Any game where the object of gameplay is something that the player will orbit around needs to know where the player's head is located. That would include any game that is situated in front of the player in virtual space. One example would be something like a virtual Jenga game where the player moves their head left and then looks right to see the left side of the stack. We think there are a bunch of possibilities for new kinds of VR games once this tracking is commonplace.



# Head tracking

```
void ZeroTracker();
```

The other tracking function your game will likely use is ZeroTracker. This sets the user's current position and rotation as the origin of the every future pose returned by the API. This is important for games that are applying head tracking information on top of some sort of in-world player position.

If you never call ZeroTracker all poses will be returned in a tracking space that is determined by the display's tracking system. This is usually the camera or base station for the system or in the case of rotation-only systems the point between the user's eyes.

One thing that won't change with Zeroing the tracker is that positive Y is always up. Zeroing affects yaw and position, but pitch and roll depend on which way the tracking system says is up.

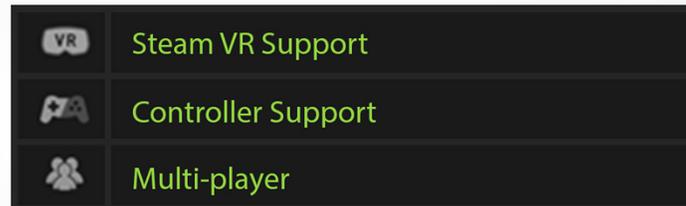


# VR in Steamworks

- Display geometry
- Distortion compensation
- Head tracking

I realize that was quite a lot to take in. To learn more, check out the Steamworks documentation and example. Or find me sometime after this and I'd be happy to answer your questions.

# VR in the Steam Store



The last thing I want to talk about is what we are doing with VR in the Steam store.

In addition to the fact that you can now use the store in VR, we are also adding a VR support indicator similar to the Controller Support indicator. That includes puts the game into a category for games that have Steam VR support.

Checking the box in the Steamworks site that makes the Steamworks VR API available for your game also turns on this feature flag to make it easier for people to find your game.

# WHEN?

So when is all this going to be out in the world?



# WHEN?

Within a few weeks

Much of what I talked about today is finished and is in the current Steam Client beta. That includes VR mode in Steam and in the overlay. Sometime in the next few weeks we will release a new version of the Steamworks SDK that contains the rest.



**JOE LUDWIG**

**JOE@VALVESOFTWARE.COM**

<http://tinyurl.com/steamworksvr>

<http://tinyurl.com/tf2vrslides>

<http://tinyurl.com/steamvrhub>

In the meantime, here's a link to the VR API documentation on the Steamworks site. And another link to the slides from my GDC talk from last year about our experience getting VR mode working in TF2. The third link here is to the Steam Community hub for Steam VR. That's where people are talking about VR mode in Steam.