Writing VR Games In Go

Things I Learned Writing Infinigrid: Escape
18 October 2017

Timothy Bogdala

What is virtual reality?

- Head mounted display (HMD)
- Controllers
- Room Scale

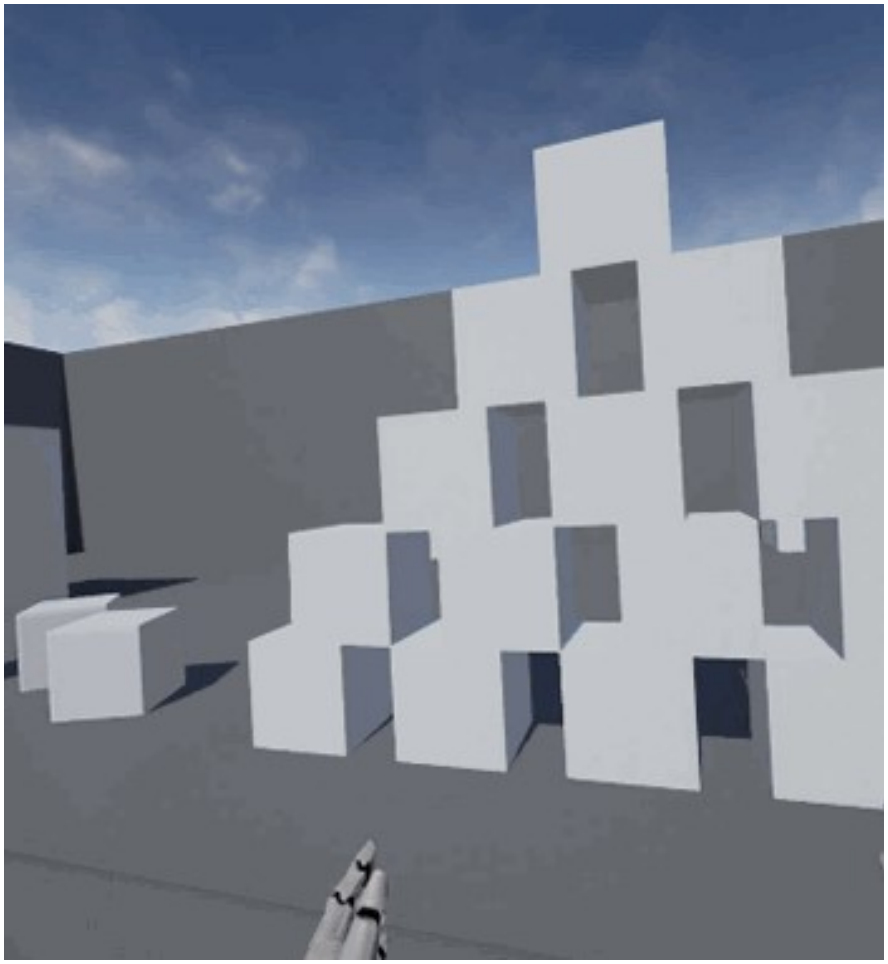
The best experience (IMO)

HTC Vive because it has all three!

HMD-only can be done (even with a PocketCHIP) but ...

How can you go about developing for VR?

Unreal Engine 4



How can you go about developing for VR?

Unity 3D

How can you go about developing for VR?

Build your own 3D graphics engine!

It can't be too bad, can it?

posts sorted by: **relevance** links from: **all time**

Trying to render a basic, single triangle to the screen. The triangle renders fine, but only on the very first frame, then the screen blanks out.

7 points • **9 comments** submitted 1 year ago by cow_co to r/opengl

So yeah, the triangle is great for the first frame, and then it just disappears. I have no idea why.

My code for the drawing of the mesh (runs once PER FRAME) is

more

I messed up passing the info to GLSL buffers. Results in **blank** screen. How to correctly pass info to the shaders?

4 points • **5 comments** submitted 2 years ago by Hexorg to r/opengl

I have a "world" where everything including the vertices stays the same from frame to frame. The only thing that changes is the position of objects. So I wanted to upload most of the data prior to

more

Why isn't anything drawing?

3 points • **6 comments** submitted 1 month ago by Bk4180 to r/opengl

Hi. Im very new to OpenGL and have to draw things for a school project. Im just trying to draw lines, and they aren't showing up at all. When I run the program, it just shows the window with the title

more

Depthtesting not working

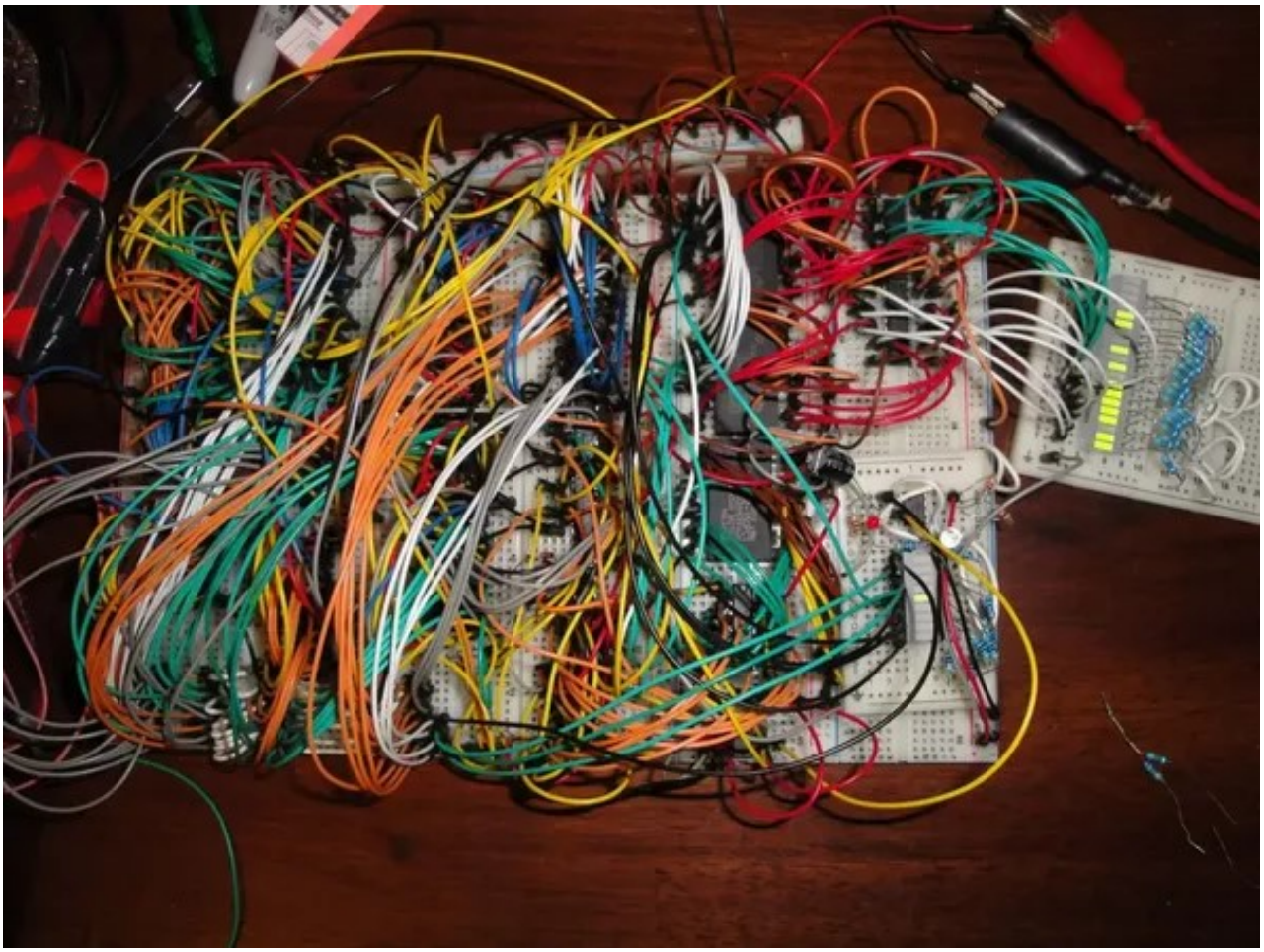
4 points • **3 comments** submitted 1 month ago by Karl__Barx to r/opengl

Hev.

A short list of needs and wants for a VR game:

- rendering pipeline (OpenGL, DirectX, Vulkan)
- VR device library support
- asset loading for textures, models, etc ...
- level design tools
- collision detection
- networking support for multiplayer
- audio support
- cross platform support

Even if you have all of these parts, how can you tie them together?



Before we get started on VR ...

Lets take a little time to review some core concepts of OpenGL programming.

What does it take to render a single triangle in OpenGL 2+ (overview)

Initialize:

- Create a window and get an GL context
- Compile a basic vertex and fragment shader into a GL program
- Create a VAO (Vertex Array Object) -- required on 3.2+ core profiles
- Create a VBO (Vertex Buffer Object) to hold the triangle vertex data
- Buffer the vertex info for the triangle into the VBO

Render Loop:

- Set the GL viewport and clear the screen
- Draw the triangles with DrawArrays

Create a window and get an GL context

- Easily done with GLFW (http://www.glfw.org/)
- Can request a specific version of OpenGL
- Go wrappers for the library and OpenGL already exist for GLFW (https://github.com/go-gl/glfw) and for OpenGL (https://github.com/go-gl/gl)

(The above OpenGL library wrapper doesn't support GL ES 3.0, however ...)

```
glfw.Init()
defer glfw.Terminate()
glfw.WindowHint(glfw.Samples, 4)
glfw.WindowHint(glfw.ContextVersionMajor, 3)
glfw.WindowHint(glfw.ContextVersionMinor, 3)
glfw.WindowHint(glfw.OpenglForwardCompatible, glfw.True)
glfw.WindowHint(glfw.OpenglProfile, glfw.OpenglCoreProfile)
window, _ := glfw.CreateWindow(640, 480, "Testing", nil, nil)
window.MakeContextCurrent()
gl.Init()
```

One small gotcha for those following closely ...

Make sure to lock the goroutine doing the GLFW calls to the main OS thread. This is needed for GLFW's event dispatching. Without this you get **random crashes**.

```
func init() {
    runtime.LockOSThread()
}
```

What are shaders?

This is a basic vertex shader written in GLSL (which looks a lot like C)

```
#version 330
in vec3 position;
out vec3 out_pos;
void main()
{
    out_pos = position;
    gl_Position = vec4(position, 1.0);
}
```

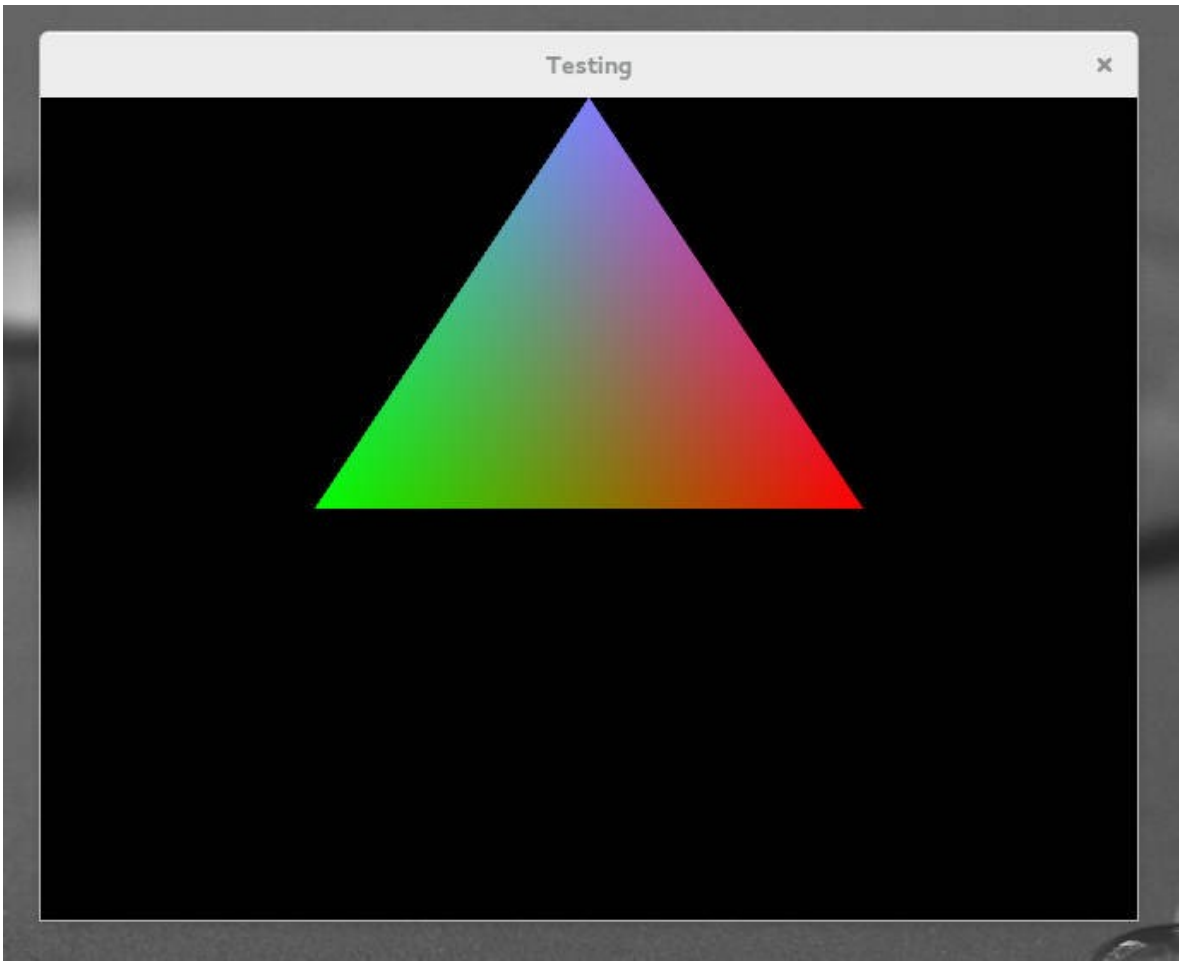
Takes a vertex position in and writes a vertex position out. No transformations. No fancy fancy. This out parameter then goes to the fragment shader.

Fragment shaders make things pretty

```
#version 330
in vec3 out_pos;
out vec4 colourOut;
void main()
{
    float red = out_pos.x + 0.5;
    float green = (-0.5 + out_pos.x) * -1.0;
    float blue = out_pos.y;
    colourOut = vec4(red, green, blue, 1.0);
}
```

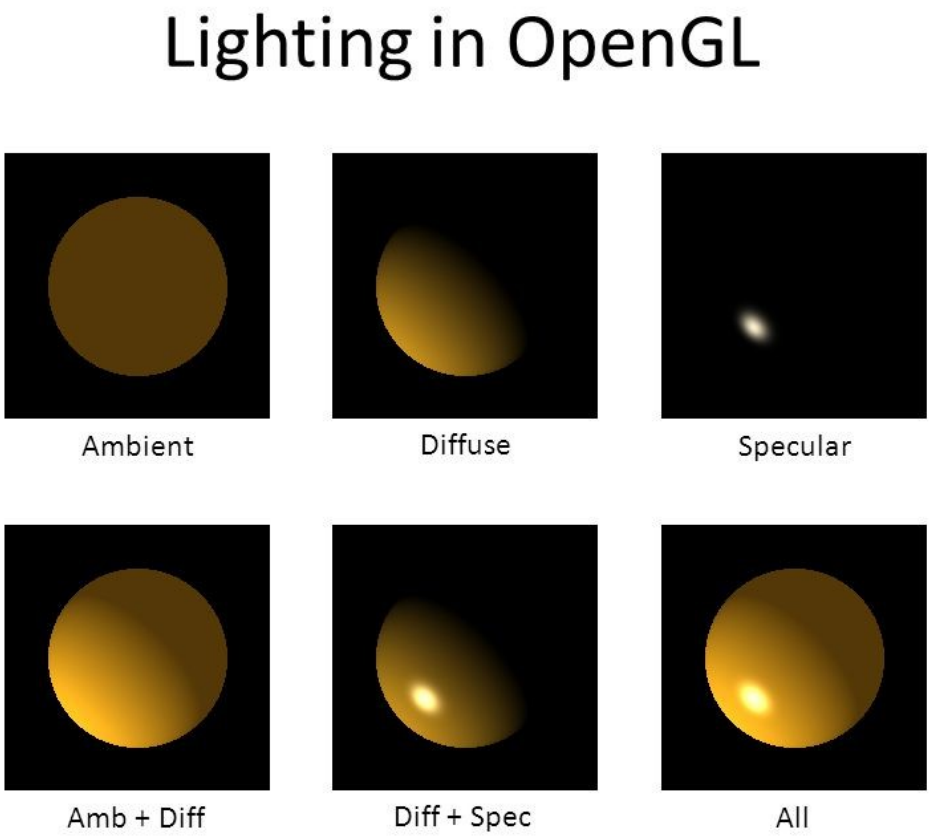
This takes the vertex position from the vertex shader output and derives a color from it. The color is then written to the out parameter `colourOut`.

The result is a simple triangle being rendered



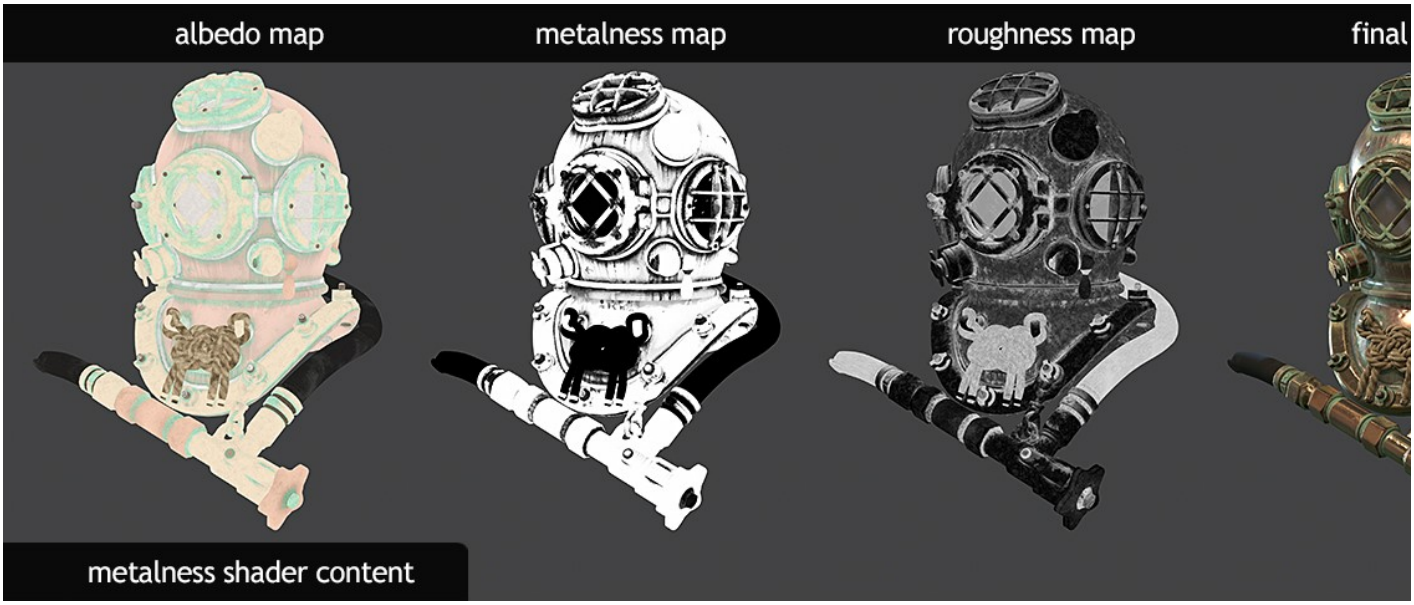
What can be done with shaders?

On a simple level, you can implement an ambient, diffuse and specular shader.



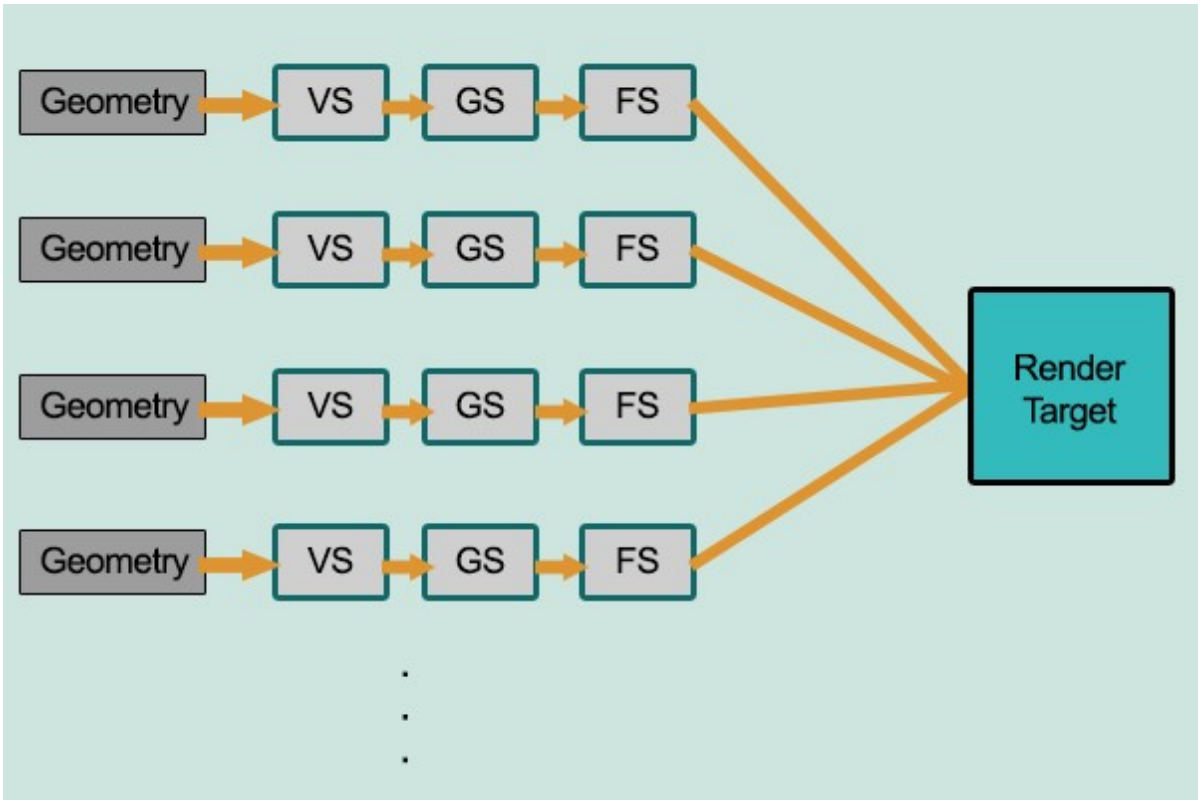
With enough work you can implement physically based lighting

This can be based off of a metal/roughness workflow or a specular/roughness workflow.



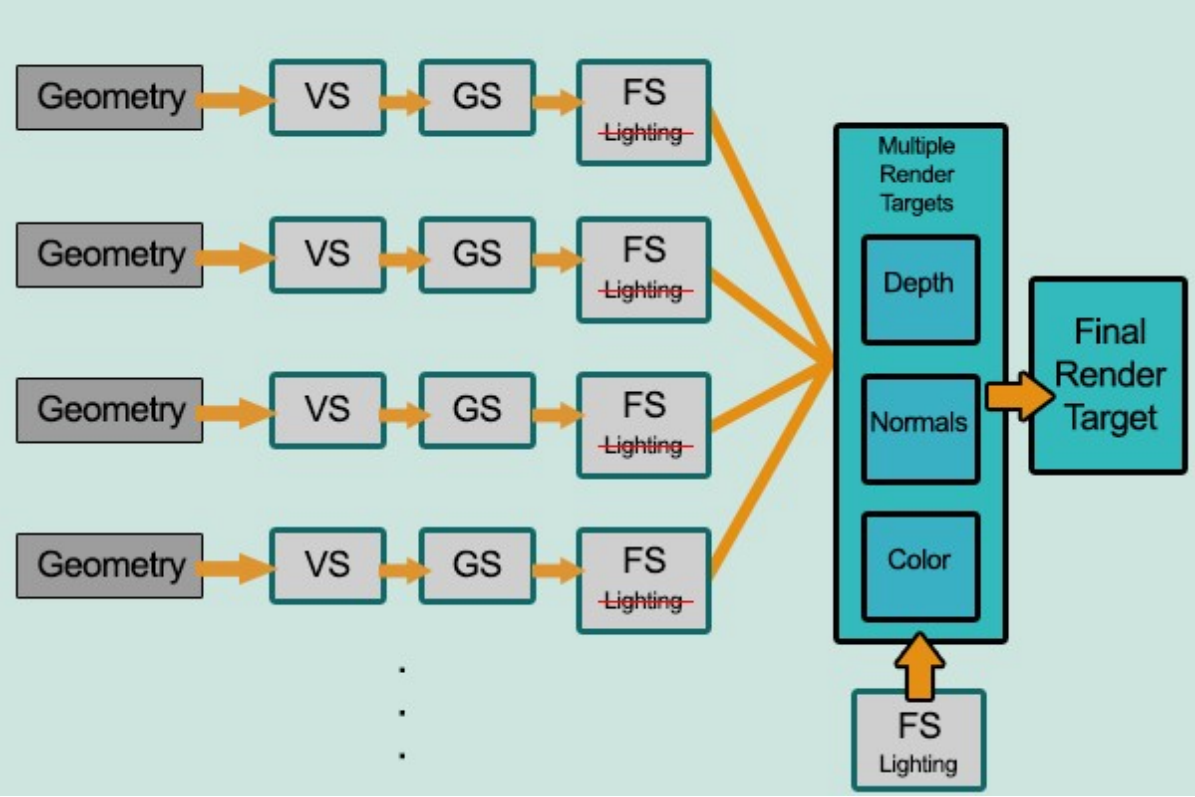
Forward rendering

A forward render will draw an object with each pixel (if lighting is done in the fragment shader) gets processed one time per light.



Deferred rendering

A deferred renderer writes information out to three or more separate display size textures and then runs another shader at the end to combine them and do all of the lighting in the last step.



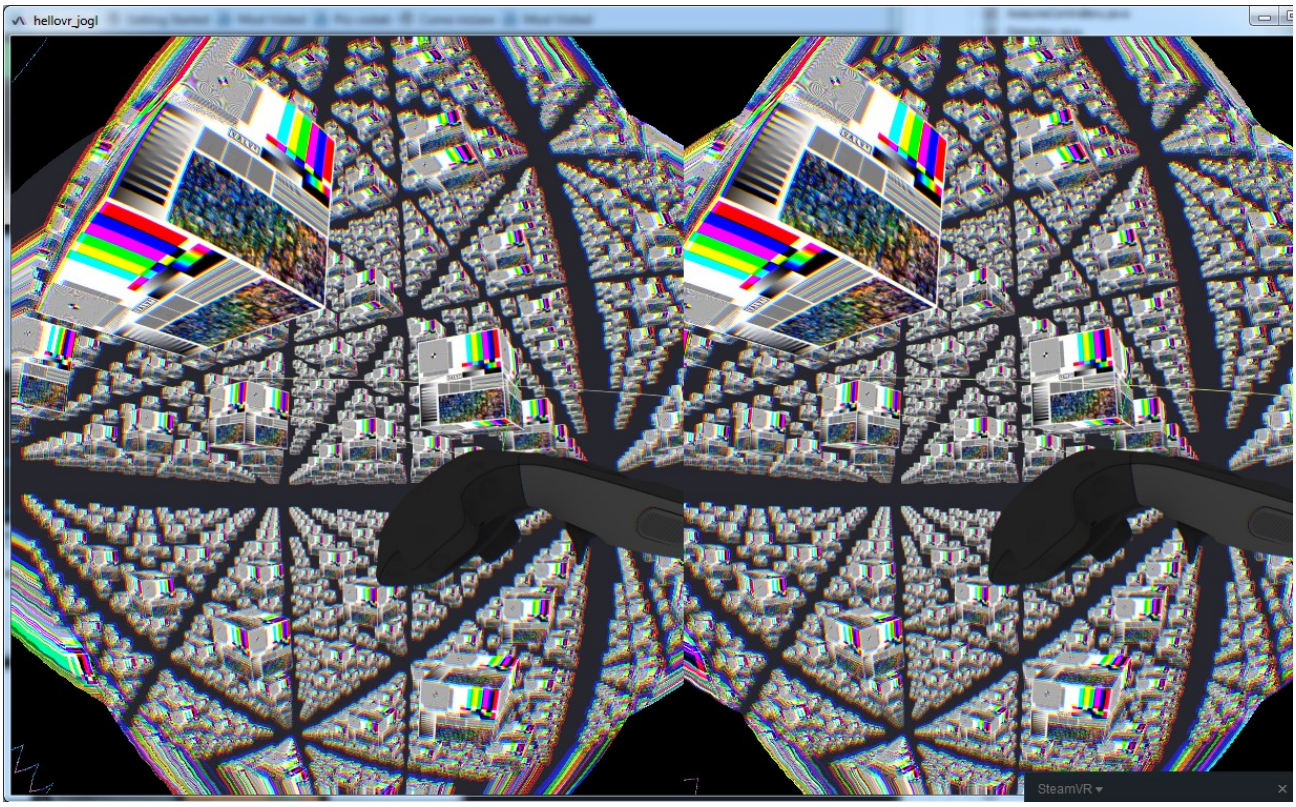
Where to go for more learning resources:

- The excellent Learn OpenGL (https://learnopengl.com/) website!
- Tom Dalling's Modern OpenGL (https://www.tomdalling.com/blog/category/modern-opengl/) blog series
- opengl-tutorial.org (http://www.opengl-tutorial.org/)
- The OpenGL Red Book (http://www.opengl-redbook.com/)
- My own article (https://animal-machine.com/blog/141218_getting_started_with_go_and_opengl.md) on starting basic OpenGL programming with Go

So why the detour into writing 3D OpenGL code?

Knowing OpenGL will help understand Valve's OpenVR SDK

- Permissive license to use the binaries and can be found here (https://github.com/ValveSoftware/openvr)
- Contains compiled binaries for the libraries and header files
- Also contains examples!



The API surface for OpenVR is daunting but you don't need most of it!

In order to make a playable VR game, you only need to use two or three interfaces and only a few methods on each interface:

- IVRSystem (https://github.com/ValveSoftware/openvr/wiki/IVRSystem_Overview): provides transforms for the HMD location/rotation as well as the eye offsets. This interface is also used to poll connected devices and get their transforms.
- IVRCompositor (https://github.com/ValveSoftware/openvr/wiki/IVRCompositor_Overview): takes a render texture for each eye so that the frame can be rendered on the HMD. It also is the sync point in the API to lock the framerate to 90 fps.
- IVRRenderModel (https://github.com/ValveSoftware/openvr/wiki/IVRRenderModels_Overview): can be used to pull models and textures for the player's controllers if you wish to render the controller on screen.

With IVRSystem you can:

- Get the recommended size for the OpenGL render targets for each eye when initializing the game.
- Get the eye offsets from the HMD to correctly render stereo images.
- Use ComputeDistortion() to generate an object the rendered textures for each eye can be projected on to mirror the look of the HMD device.
- During the game loop, tracked devices can be polled for ther state using GetControllerState()

With IVRCompositor you can:

- Render a frame to the HMD by submitting a render texture to each eye.
- After submitting the render textures, IVRCompositor:WaitGetPoses() is called to trigger the frame update as well as updating the HMD pose and location.
- With the HMD pose, all 3D objects in the scene can be projected correctly for the view.

What does it mean to render a texture to each eye?

- It's a similar process to the deferred rendering process mentioned in the previous brief OpenGL tutorial.
- Instead of rendering directly to the 'screen', you bind OpenGL texture objects to an OpenGL framebuffer object. Each eye then gets it's own framebuffer object.

Oh Emmmm Geee, what does this all mean?!!

It means that with a little abstraction you can do both 3D and VR in the same game!

You just need to adapt the following:

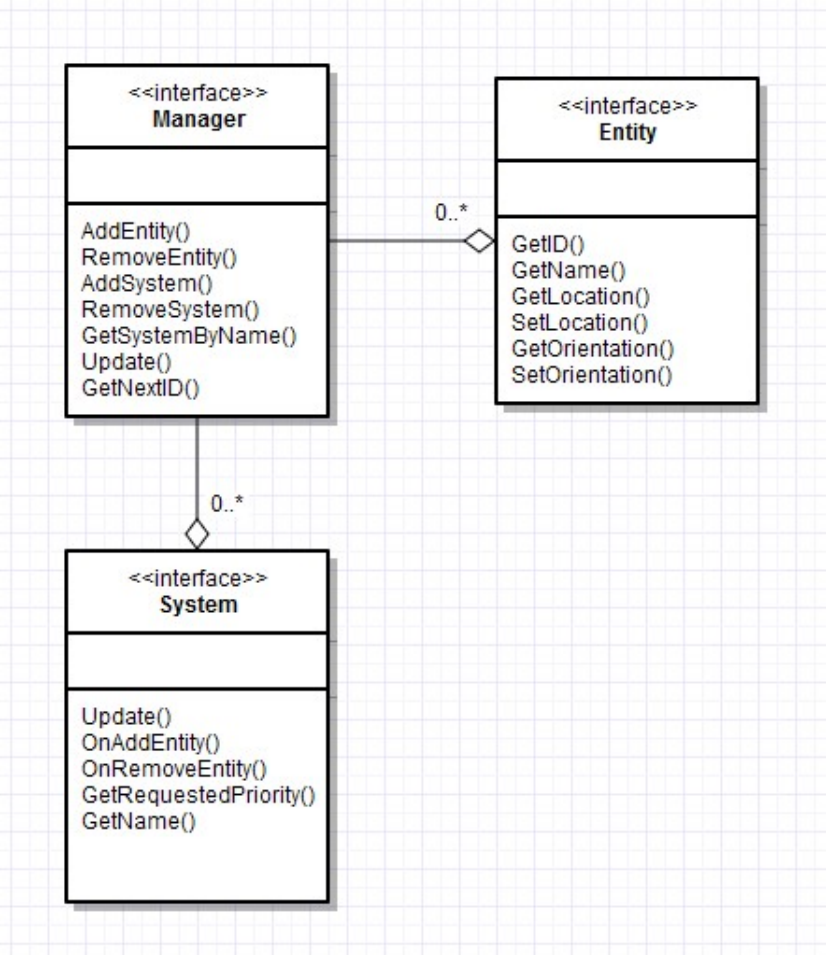
- Initialization of OpenGL rendering structures (e.g. make those framebuffers or not)
- Input devices for the VR controllers instead of mouse/keyboard or gamepad
- Any particular camera control / movement that might be affected by HMD motion
- Rendering out normally or once per eye and then combining them

A little word about our dependencies:

- Demo code in openvr-go (github.com/tbogdala/openvr-go) as well as infinigrid: escape (https://github.com/tbogdala/infinigrid) use my open source Go 3D graphics library called fizzle (github.com/tbogdala/fizzle). This is an experimental engine and I haven't pinned the API down yet.
- My work-in-progress OpenGL user interface library eweygewey (github.com/tbogdala/eweygewey) is also used. Same warnings apply.
- My last own dependency project is a small collision library called glider (github.com/tbogdala/glider). It handles axis-aligned bounding boxes and spheres and is quicker than importing a physics library (like my own cubez (github.com/tbogdala/cubez) and using collisions from there.)

Fizzle's 'scene' package

This is the solution I've adopted to keeping a game scene organized and flexible.



fizzle.scene.Manager

- registers systems and entities for the game scene
- keeps a priority sorted list of systems and updates them in order every frame
- sends events to all systems such as AddSystem or RemoveSystem
- handles requests for new Entity ID numbers

fizzle.scene.System

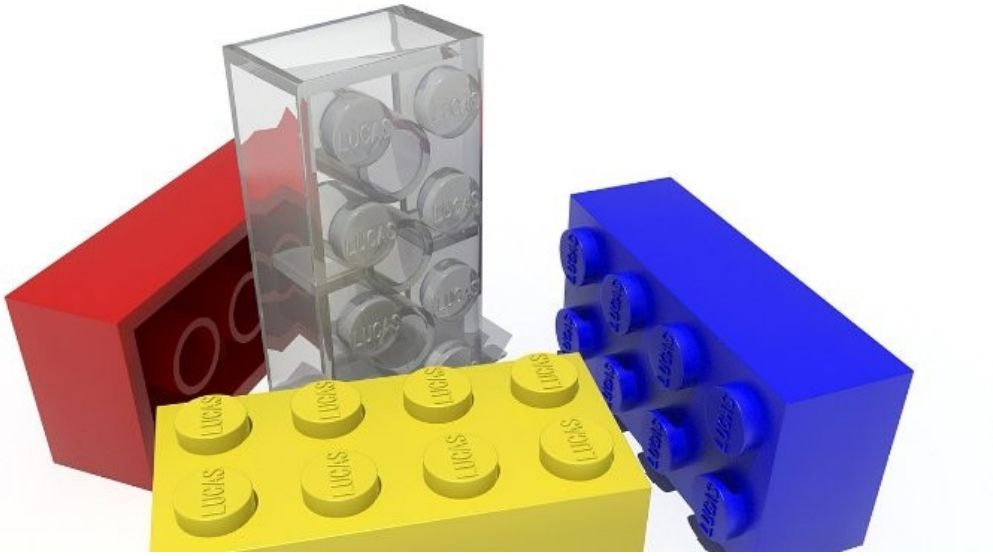
- a 'controller' like interface meant to be implemented by game systems that will need to update on a given frame update cycle (but not necessarily every frame).
- examples of the fizzle.scene.System interface in the demo projects include the forward renderer, user interface system and the input systems.

fizzle.scene.Entity

- simply an object in the game -- not necessarily even able to move or be rendered.
- the BasicEntity implementation and Entity interface only contain the following information: ID (within the scene.Manager), Name, Location and Orientation.
- this means that a user defined struct or interface can be defined by client code and that Entities do not have to have a renderable object.

By using the fizzle.scene package a pair of render and input systems can be created!

- specialized systems for forward rendering and keyboard/mouse input can be initialized by default
- if a flag is set, then a VR renderer and VR input system pair can be added to the scene.Manager instead



Advantage of Go's interfaces

- it's easy to create small interfaces such as `CollisionEntity` that can be implemented by all Entity types that support collision detection.
- an example of this in `infinigrid` is the `ShipEntity` type which, while implementing `scene.Entity` also can implement `CollisionEntity` just by defining a method for the `ShipEntity` type.
- this feels very natural and keeping these interfaces small means there's less to worry about when deciding whether or not to implement the interface, because you can make your selections *a la carte*.

Speaking of Go... How do we interface with the OpenVR SDK?

- it's a C++ library that also can work with C interfaces (the openvr_capi.h is auto-generated).
- time for cgo!

How to interface with C: the short version

After you declare the package and just before you `import "C"` you can put a block of code in C style comments (/* ... */)

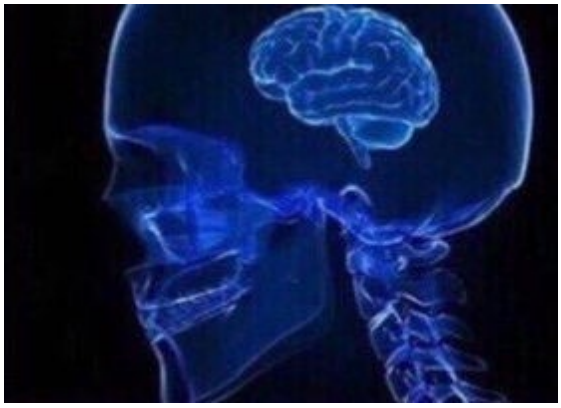
```
package openvr
/*
#include <stdio.h>
#include <stdlib.h>
#include "openvr_capi.h"

#if defined(_WIN32)
    #define IMPORT __declspec(dllimport)
#else
    #define IMPORT
#endif
IMPORT void VR_ShutdownInternal();
*/
import "C"
```

And then you can call it in your Go code by using the `C` package:

The C package makes it easy to access the functions and data declared on the C side of things.

```
func Shutdown() {
    C.VR_ShutdownInternal()
}
```



Okay we can import DLL functions ... what does OpenVR export?

Answer: not a lot.

E	Ordinal ^	Hint	Function	Entry Point
C	1 (0x0001)	0 (0x0000)	LiquidVR	0x00001BA0
C	2 (0x0002)	1 (0x0001)	VRCompositorSystemInternal	0x00001BF0
C	3 (0x0003)	2 (0x0002)	VRControlPanel	0x00001C40
C	4 (0x0004)	3 (0x0003)	VRDashboardManager	0x00001C90
C	5 (0x0005)	4 (0x0004)	VROculusDirect	0x00001CE0
C	6 (0x0006)	5 (0x0005)	VRRenderModelsInternal	0x00001D30
C	7 (0x0007)	6 (0x0006)	VRTrackedCameraInternal	0x00001D80
C	8 (0x0008)	7 (0x0007)	VR_GetGenericInterface	0x00001DD0
C	9 (0x0009)	8 (0x0008)	VR_GetInitToken	0x00001E00
C	10 (0x000A)	9 (0x0009)	VR_GetStringForHmdError	0x00001E10
C	11 (0x000B)	10 (0x000A)	VR_GetVRInitErrorAsEnglishDescription	0x00001E10
C	12 (0x000C)	11 (0x000B)	VR_GetVRInitErrorAsSymbol	0x00001E30
C	13 (0x000D)	12 (0x000C)	VR_InitInternal	0x00001E50
C	14 (0x000E)	13 (0x000D)	VR_IsHmdPresent	0x00001EE0
C	15 (0x000F)	14 (0x000E)	VR_IsInterfaceVersionValid	0x00001F50
C	16 (0x0010)	15 (0x000F)	VR_IsRuntimeInstalled	0x00001F80
C	17 (0x0011)	16 (0x0010)	VR_RuntimePath	0x00002090
C	18 (0x0012)	17 (0x0011)	VR_ShutdownInternal	0x00002190

How can we access those sweet, sweet VR functions then?

We define our own C function above the import command to do the following:

```
struct VR_IVRSystem_FnTable* _iSystem;

// paraphrased & error checking skipped
int initInternal(int appTypeEnum) {
    EVRInitError error = EVRInitError_VRInitError_None;
    intptr_t _iToken = VR_InitInternal(&error, appTypeEnum);

    // IVRSystem_Version is defined in openvr_capi.h as
    // static const char * IVRSystem_Version = "IVRSystem_017";
    bool icheck = VR_IsInterfaceVersionValid(IVRSystem_Version);

    char interfaceFnTable[256];
    sprintf(interfaceFnTable, "FnTable:%s", IVRSystem_Version);
    _iSystem = (struct VR_IVRSystem_FnTable*) VR_GetGenericInterface(in
}
```

What does that C code do?

- we call the `VR_InitInternal` function first
- then we check to see if the DLL supports the requested IVRSystem_Version
- finally, we request the interface by name through VR_GetGenericInterface

What's inside this VR_IVRSystem_FnTable structure?

Answer: a whole bunch of function pointers!

```
// OpenVR Function Pointer Tables
struct VR_IVRSystem_FnTable
{
void (OPENVR_FNTABLE_CALLTYPE *GetRecommendedRenderTargetSize)(uint32_t
struct HmdMatrix44_t (OPENVR_FNTABLE_CALLTYPE *GetProjectionMatrix)(EVR
void (OPENVR_FNTABLE_CALLTYPE *GetProjectionRaw)(EVREye eEye, float * p
bool (OPENVR_FNTABLE_CALLTYPE *ComputeDistortion)(EVREye eEye, float fU
struct HmdMatrix34_t (OPENVR_FNTABLE_CALLTYPE *GetEyeToHeadTransform)(E
bool (OPENVR_FNTABLE_CALLTYPE *GetTimeSinceLastVsync)(float * pfSeconds
int32_t (OPENVR_FNTABLE_CALLTYPE *GetD3D9AdapterIndex)();
void (OPENVR_FNTABLE_CALLTYPE *GetDXGIOutputInfo)(int32_t * pnAdapterIn
void (OPENVR_FNTABLE_CALLTYPE *GetOutputDevice)(uint64_t * pnDevice, ET
bool (OPENVR_FNTABLE_CALLTYPE *IsDisplayOnDesktop)();
bool (OPENVR_FNTABLE_CALLTYPE *SetDisplayVisibility)(bool bIsVisibleOnD
void (OPENVR_FNTABLE_CALLTYPE *GetDeviceToAbsoluteTrackingPose)(ETracki
void (OPENVR_FNTABLE_CALLTYPE *ResetSeatedZeroPose)();
struct HmdMatrix34_t (OPENVR_FNTABLE_CALLTYPE *GetSeatedZeroPoseToStand
/* ... ... ... */
```

If the API is a set of functions in a C struct, lets call them!

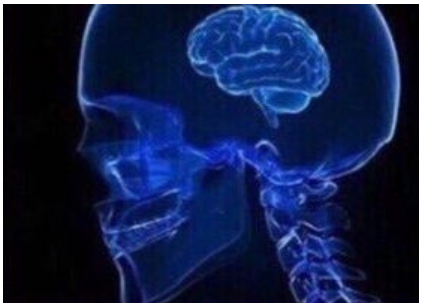
We could then set up a function to access that struct from C and then call the function like so:

```
func GetRecommendedRenderTargetSize() (uint32, uint32) {
    var w, h C.uint32_t
    systemInterface := C._iSystem
    systemInterface.GetRecommendedRenderTargetSize(&w, &h)
    return uint32(w), uint32(h)
}
```

Unfortunately this will generate an error:

```
cannot call non-function (*_Cvar__iSystem).GetRecommendedRenderTargetSi
```

It turns out that cgo doesn't support calling functions from pointers in a structure.



This happens to be my biggest source of frustration with making library wrappers

To get around this I create a series of C functions that take the structure with the function pointers as the first parameter.

```
struct HmdMatrix44_t system_GetProjectionMatrix(struct VR_IVRSystem_FnT
    return iSystem->GetProjectionMatrix(eEye, fNearZ, fFarZ);
}

struct HmdMatrix34_t system_GetEyeToHeadTransform(struct VR_IVRSystem_F
    return iSystem->GetEyeToHeadTransform(eEye);
}

bool system_ComputeDistortion(struct VR_IVRSystem_FnTable* iSystem, EVR
    return iSystem->ComputeDistortion(eEye, fU, fV, dest) != 0;
}

bool system_IsTrackedDeviceConnected(struct VR_IVRSystem_FnTable* iSyst
    return iSystem->IsTrackedDeviceConnected(unDeviceIndex);
}

/* ... */
```

Then on the Go side I create a Go struct to wrap the C struct and define methods

```
type System struct {
    ptr *C.struct_VR_IVRSystem_FnTable
}

func (sys *System) GetProjectionMatrix(eye int, near, far float32, dest
    m44 := C.system_GetProjectionMatrix(sys.ptr, C.EVREye(eye), C.float
    /* reorder the matrix for compatability with OpenGL */
}

func (sys *System) GetEyeToHeadTransform(eye int, dest *mgl.Mat3x4) {
    m34 := C.system_GetEyeToHeadTransform(sys.ptr, C.EVREye(eye))
    /* reorder the matrix for compatability with OpenGL */
}

/* ... */
```

And with that technique, we can wrap the rest of the OpenVR library!

Using this in Go client code looks like this:

```
var sys *System // initialized previously
```

```
var m mgl.Mat4
var m34 mgl.Mat3x4
```

```
near := float32(0.1)
far := float32(100.0)
```

```
sys.GetProjectionMatrix(EyeLeft, near, far, &m)
sys.GetEyeToHeadTransform(EyeLeft, &m34)
```

The last task then is to link the correct OpenVR library

To do this I add some cgo directives to the primary Go file of the package within the C import comment:

```
#cgo CFLAGS: -I${SRCDIR}/vendored/openvr/headers -std=c99
#cgo windows,386 LDFLAGS: -L${SRCDIR}/vendored/openvr/bin/win32 -lopenv
#cgo windows,amd64 LDFLAGS: -L${SRCDIR}/vendored/openvr/bin/win64 -lope
#cgo linux,amd64 LDFLAGS: -L${SRCDIR}/vendored/openvr/bin/linux64 -lope
#cgo linux,386 LDFLAGS: -L${SRCDIR}/vendored/openvr/bin/linux32 -lopenv
```

This makes sure that `openvr_capi.h` can be found by the C compiler and that the vendored library binaries can be found by the linker.

Keep looking up?

```
"It actually feels like all of the pieces, all of the ingredients that
are already really here, they're just not stirred, cooked, and seasoned
-- John Carmack, Oculus Connect 4 Oct 12, 2017
```

- we really do have access to great hardware
- the software exists the drive this hardware and is freely available
- now is the time to try new ideas and create totally new experiences in VR!

Thanks for listening!

- the main game developed for this talk is Infinigrid: Escape (https://github.com/tbogdala/infinigrid).
- my openvr-go (https://github.com/tbogdala/openvr-go) wrapper is open to pull requests.
- you can find me on twitch.tv/animalmachine (http://twitch.tv/animalmachine)
- or on twitter: @tbogdala (https://twitter.com/tbogdala)
- or the rest of my code on github.com/tbogdala (https://github.com/tbogdala)
- if all of that fails, drop an email: tdb@animal-machine.com

Thank you

Timothy Bogdala
tdb@animal-machine.com (mailto:tdb@animal-machine.com)
https://www.animal-machine.com (https://www.animal-machine.com)
@tbogdala (http://twitter.com/tbogdala)