

Attribution/License

- Original Materials developed by Mike Shah, Ph.D. (www.mshah.io)
- This slideset and associated source code may not be distributed without prior written notice

Please do not redistribute slides/source without prior written permission.



The Case for Graphics

-- in **DLang: Part 2 - Tech Demo**
with Mike Shah

Social: [@MichaelShah](#)

Web: [mshah.io](#)

Courses: [courses.mshah.io](#)

 **YouTube**

[www.youtube.com/c/MikeShah](#)

<http://tinyurl.com/mike-talks>

14:30 - 15:00 Fri, Sept 19, 2024

30 minutes | Introductory Audience

Abstract (which you already read :))

Talk Abstract: The D programming language and ecosystem provides many modern features which can help give developers a competitive advantage during the software development lifecycle. In this talk I will discuss how D provides a competitive advantage for graphics application development (e.g. games, rendering), where any reduction in iteration time can improve outcomes. As an example, when creating an art style, having systems allowing rapid iteration can help improve visual fidelity, both in aesthetics and correctness of the application. Throughout the talk I will showcase visual examples, developed in a summer, that show off a graphics engine built completely in the D ecosystem. Topics will include: a hot reload system, job system (using concurrency), and showcasing the architecture of a small graphics engine. This talk follows up a previous Dconf Online 2024 talk in which I make a case for using D for graphics programming – this time showing you the results (attendees need not have watched the previous talk).

Your Tour Guide for Today

Mike Shah

- **Current Role:** Teaching Faculty at **Yale University**
(Previously Teaching Faculty at Northeastern University)
 - **Teach/Research:** computer systems, graphics, geometry, game engine development, and software engineering.
- **Available for:**
 - **Contract work** in Gaming/Graphics Domains
 - e.g. tool building, plugins, code review
 - **Technical training** (virtual or onsite) in Modern C++, D, and topics in Performance or Graphics APIs
- **Fun:**
 - Guitar, running/weights, traveling, video games, and cooking are fun to talk to me about!



Web

www.mshah.io



YouTube

<https://www.youtube.com/c/MikeShah>

Non-Academic Courses

courses.mshah.io

Conference Talks

<http://tinyurl.com/mike-talks>

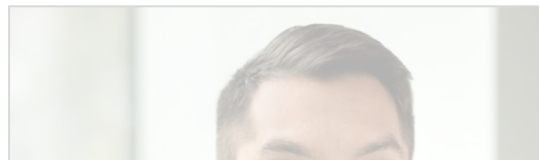
Your Tour Guide for Today

Mike Shah

- **Current Role:** Teaching Faculty at **Yale University**

(Previously Teaching Faculty at Northeastern University)

- **Teach/Research:** computer graphics, physics, geometry, game



D Community Announcement:

University students are learning and using D in my Building Game Engines course at Yale University! Hooray!

It's the right tool for the job!

[\[See last years DConf talk from myself and students why\]](#)



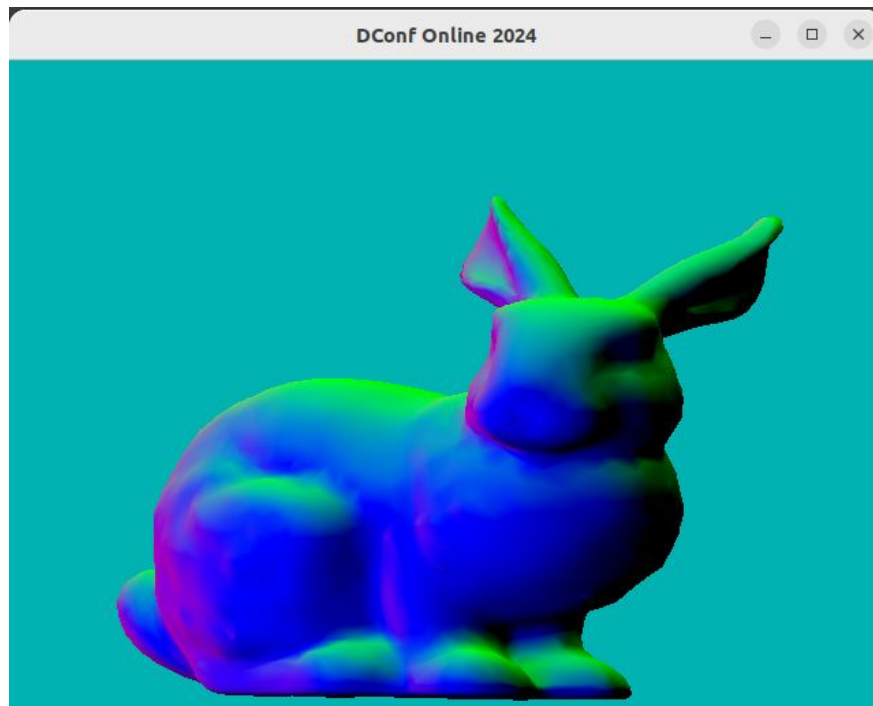
Non-Academic Courses
courses.mshah.io

Conference Talks

<http://tinyurl.com/mike-talk5>

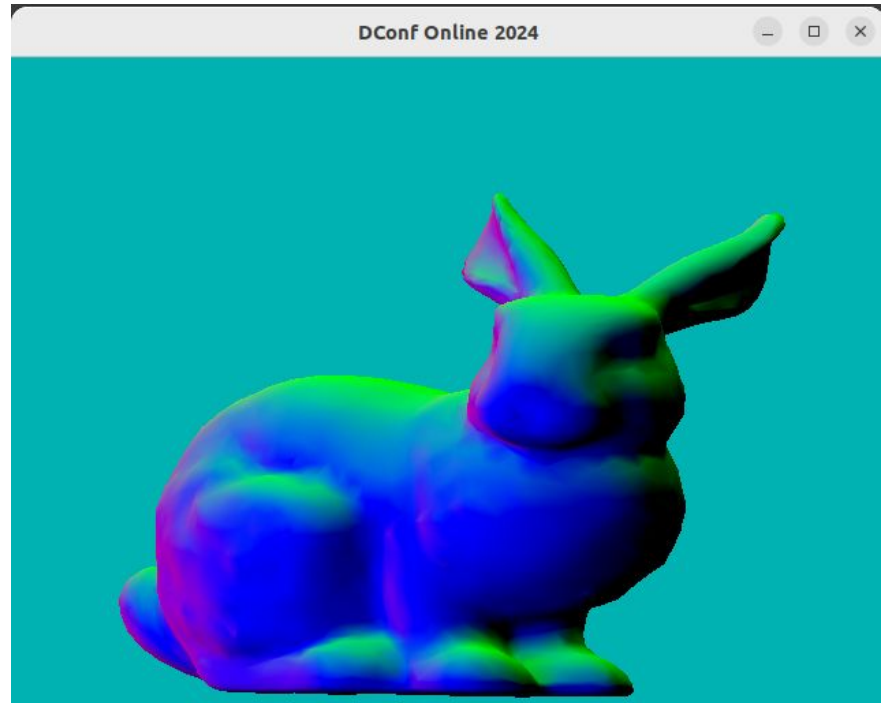
Last Time (Part 1)

We discussed glfw binding, a Bunny,
discussed Multiple Rendering Targets



DConf 2024 Online (1/2)

- This is a 'part 2' talk of my DConf 2024 Online talk
 - https://www.youtube.com/watch?v=8GV_TuYk3lk&t=11766s
- Don't worry though -- **part 1 is not a prerequisites for this talk.**
 - That said -- if you want to get started with graphics programming, that may be a good source of information.
 - I have some skeleton code for rendering objects with a framebuffer.



DConf 2024 Online (2/2)

- This is a 'part 2' talk of my DConf 2024 Online talk

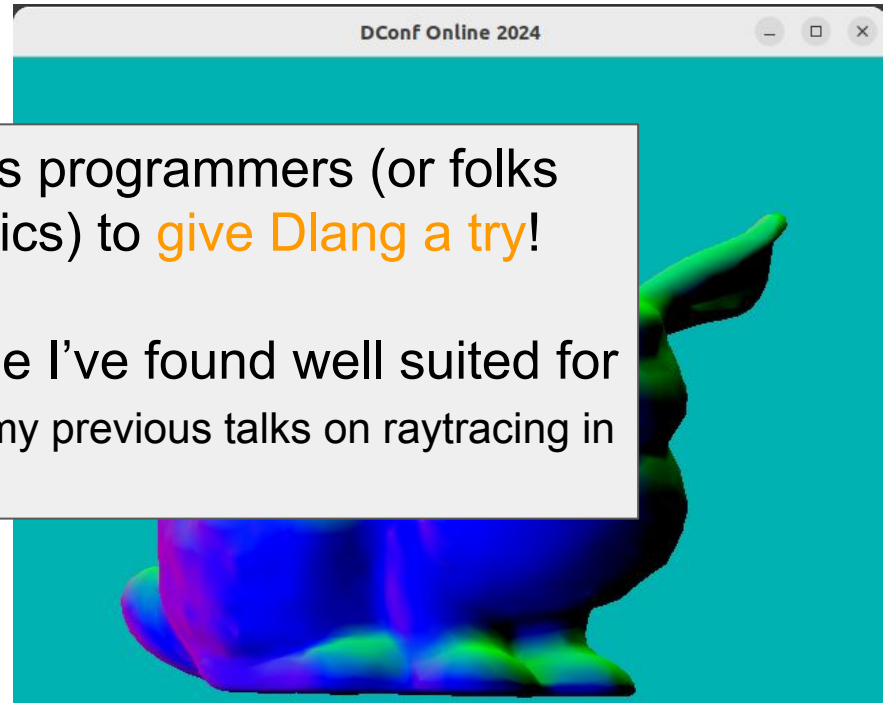
- **My goal today:** Is for graphics programmers (or folks considering getting into graphics) to **give Dlang a try!**

- Do is r tal

- The D Programming Language I've found well suited for real-time graphics work! (See my previous talks on raytracing in D for non-real time graphics work)

- programming, that may be a good source of information.

- I have some skeleton code for rendering objects with a framebuffer.



Quick Demo (1/2)

- So here's a little capture of a scene I'm working on
 - There's A little bit of lighting, and a few models loaded, and about 260,000 triangles to draw the scene.
 - It's a purposefully 'unoptimized set of art assets' to stress the system
 - This is the classic 'Sponza' scene used in graphics with the classic 'Stanford Bunny' usually as benchmarks
- This was just a small hackathon in a few days work!



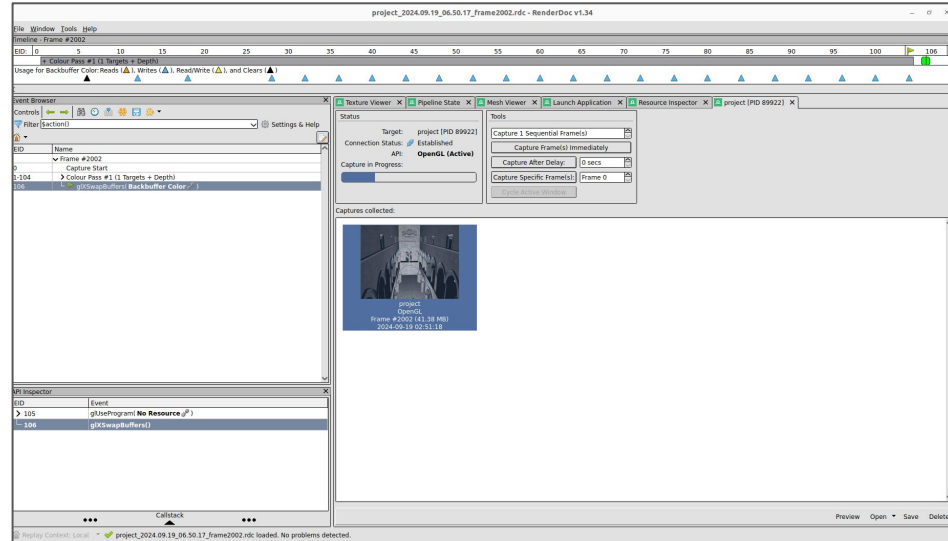
Quick Demo (2/2)

- Let's dive in just a tiny bit deeper to see what was interesting here
 - **Next Slide**



RenderDoc (a GPU Profiler) (1/2)

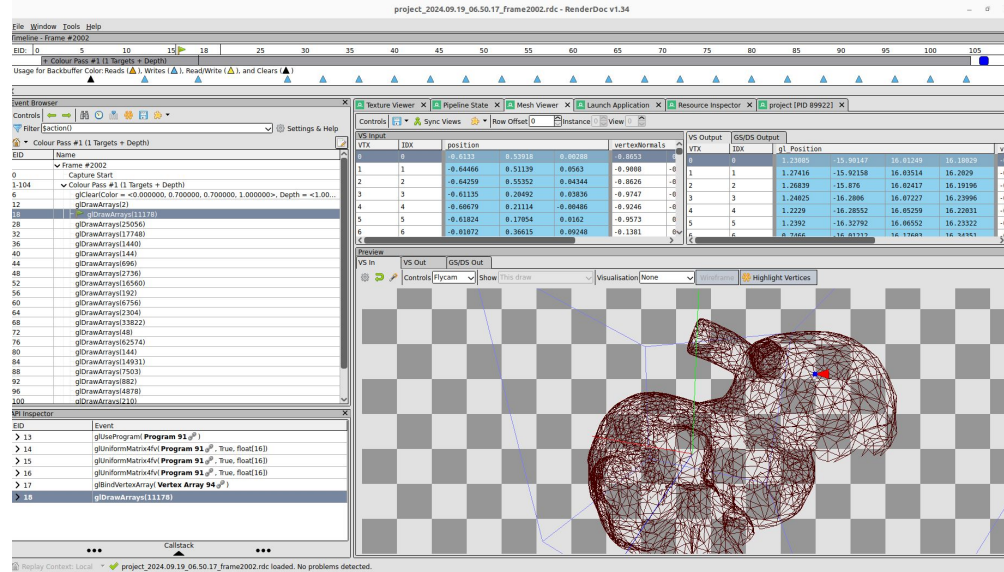
- For game and graphics programming, the same GPU tools (e.g. Renderdoc) have worked just fine for me in D as other toolstacks (e.g. C++)
 - These GPU Profilers are very valuable for capturing a ‘memory snapshot’ of what’s been allocated on the GPU
- Other tools like ‘perf’ for CPU profiling also work well with D.
 - *I also like reminding folks of the builtin profiler in D which is handy.)



<https://renderdoc.org/builds>

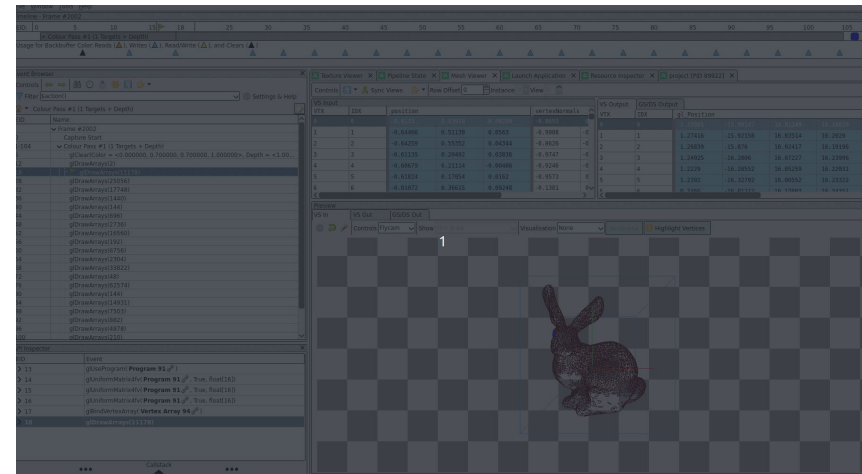
RenderDoc (a GPU Profiler) (2/2)

- Within a tool like Renderdoc, you can inspect the geometry just as you normally would --
 - Again it's the same OpenGL, Vulkan, etc. function calls.
 - If you have prior programming experience in these APIs, the experience transfers directly over.



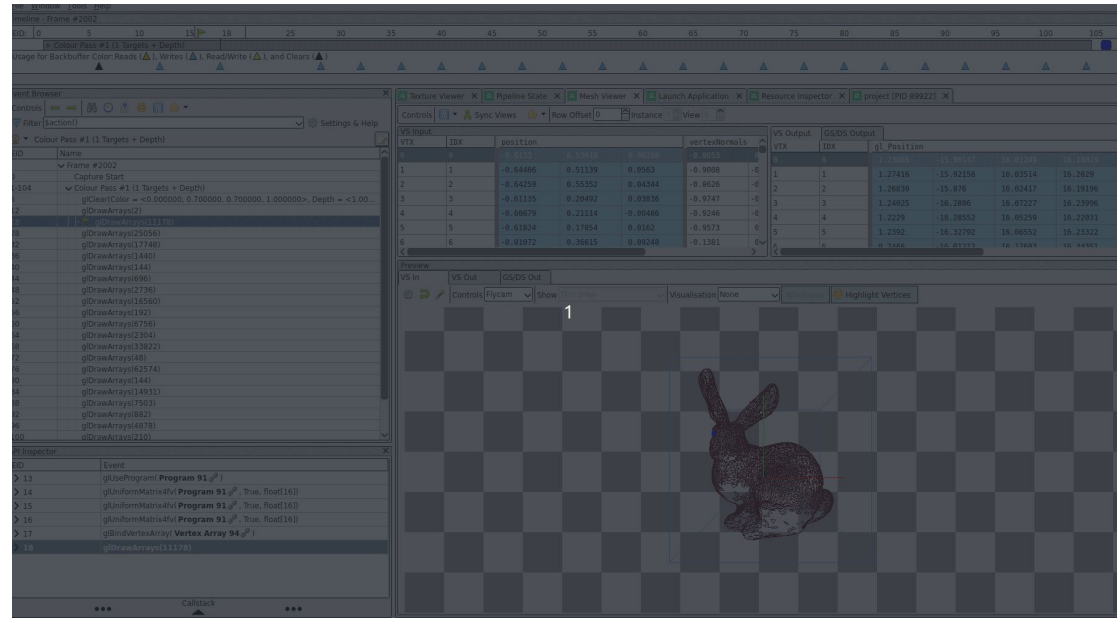
Working with Geometry Challenge

- What is perhaps interesting in this scene is that it may appear to the user that there are only ‘two’ pieces of geometry.
 - the ‘bunny’ (glowing fun colors)
 - The building -- itself is just one file, but made up of many ‘chunks’ of other 3D data
 - **next slide to see closer**



OBJ File Format [[wavefront obj file format](#)]

- Same screenshot as before, just slightly larger
- Again showing that in order to sift through the many 'chunks' of data in one file .obj I had to parse it and separate out the data.



Parsing OBJ Files (1/2)

- A .obj (3D Object File Format) file looks something like on the right
- We have geometry data at the top
- We then have potentially 1 or more materials and/or objects group on the bottom

```
# Vertices: 8
# Points: 0
# Lines: 0
# Faces: 6
# Materials: 1
```

```
o 1
```

```
# Vertex list
```

```
v -0.5 -0.5 0.5
v -0.5 -0.5 -0.5
v -0.5 0.5 -0.5
v -0.5 0.5 0.5
v 0.5 -0.5 0.5
v 0.5 -0.5 -0.5
v 0.5 0.5 -0.5
v 0.5 0.5 0.5
```

```
# Point/Line/Face list
```

```
usemtl Default
```

```
f 4 3 2 1
f 2 6 5 1
f 3 7 6 2
f 8 7 3 4
f 5 8 4 1
f 6 7 8 5
```

```
# End of file
```

See DConf 2024 Online talk for how short the parsing code can be!

Parsing OBJ Files (2/2)

- What's neat is you can actually parallelize this process (where it makes sense on large enough files!)
- So if your artists are throwing lots of geometry and textures at you, you can parse the top half first -- then
 - Every time you hit 'usemtl' you can kickstart the process of creating a 'chunk' of a 3D object, or otherwise parsing the material file or loading the image files
 - It's become a little bit of a hobby project to see how fast I can parse these .obj files -- stay tuned!
 - i.e. [Caldera Data Set from Call of Duty](#) will begin investigation soon.

```
# Vertices: 8
# Points: 0
# Lines: 0
# Faces: 6
# Materials: 1
```

```
o 1
```

```
# Vertex list
```

```
v -0.5 -0.5 0.5
v -0.5 -0.5 -0.5
v -0.5 0.5 -0.5
v -0.5 0.5 0.5
v 0.5 -0.5 0.5
v 0.5 -0.5 -0.5
v 0.5 0.5 -0.5
v 0.5 0.5 0.5
```

```
# Point/Line/Face list
```

```
usemtl Default
```

```
f 4 3 2 1
f 2 6 5 1
f 3 7 6 2
f 8 7 3 4
f 5 8 4 1
f 6 7 8 5
```

```
# End of file
```


Graphics Engine Design

Working Backwards a Bit

Structure of a Game (1/2)

- So moving away from the graphics stuff for a moment, the infrastructure for these projects is pretty neat at the ‘core game loop’
 - Basically it’s just an input/update/render function
 - I like to separate that out to another function (AdvanceFrame()) for more control

```
void Run(){
    while(mGameRunning){
        AdvanceFrame();
    }
}

void AdvanceFrame(){
    if(mGameRunning.paused){
        /* Show pause screen or suspend process */
    }
    // Execute all of our callbacks that users have added
    foreach(callback ; mGameFrameCallbacks){
        callback.frameStarted();
    }

    InputFunc();
    UpdateFunc();
    RenderFunc();

    // Execute all of our callbacks that users have added
    foreach(callback ; mGameFrameCallbacks){
        callback.frameEnded();
    }
}
```

Structure of a Game (2)

- Of course in a game/graphics application you may want more power and make the system more dynamic
- It becomes relatively easy to have some 'interface' that you can write to
 - This is where 'callbacks' come in, and I can hook into the system to do whatever is needed.
 - Note: Writing your own events to some FIFO queue is another strategy

```
11 // Provide a common interface from which we can derive new
12 // classes from.
13 interface GameFrameCallback{
14     void frameStarted();
15     void frameEnded();
16 }
17
18 // Example of a new 'callback' to add functionality to our game loop
19 // Note: Must use a 'class' in dlang, as 'structs' cannot use interface
20 class PrintFrameCallback : GameFrameCallback{
21     import std.stdio;
22     this(){ }
23     ~this(){ }
24     void frameStarted(){
25         writeln("New frame starting");
26     }
27     void frameEnded(){
28         writeln("Frame is ending");
29     }
30 }
31 ///////////////////////////////////////////////////
```

```
callback.frameStar
```

```
InputFunc();
UpdateFunc();
RenderFunc();
```



```
// Execute all of our callbacks that users have added
foreach(callback ; mGameFrameCallbacks){
    callback.frameEnded();
}
}
```

Runtime Polymorphism without classes

- With a little bit of cleverness, I am doing something for my callback system similar to the tardy project.
 - (Thanks Atila!)
- Then I can basically use only structs for everything :)
 - Atila has a nice project here I got some ideas from!
- My interest is wanting to keep flexibility of polymorphism, but within the betterC subset.
 - betterC is a really neat part of the D ecosystem -- top notch for portability and/or embedded systems
 - <https://dlang.org/spec/betterc.html>
 - https://wiki.dlang.org/Generating_WebAssembly_with_LDC

Info Reference ↗

tardy - runtime polymorphism without inheritance

build unknown  build unknown  codecov 60%

What?

```
import tardy;

interface ITransformer {
    int transform(int) @safe pure const;
}
alias Transformer = Polymorphic!ITransformer;

int xform(Transformer t) {
    return t.transform(3);
}

struct Adder {
    int i;
    int transform(int j) @safe pure const { return i + j; }
}

struct Plus1 {
    int transform(int i) @safe pure const { return i + 1; }
}

unittest {
    assert(xform(Transformer(Adder(2))) == 5);
    assert(xform(Transformer(Adder(3))) == 6);
}
```

<https://code.dlang.org/packages/tardy>

Productivity in D for Graphics Programming

Back to graphics

Folks here already know (or are learning !)

- D is a productive language
- So I have a few small highlights from this project of what I found useful

Rapid Iteration Time Matters A lot (Bloopers Reel)

- In graphics you encounter all sorts of strange errors
 - So fast build times matter!
- Compiling and building primarily on DMD
 - LDC2 and GDC also build quite fast!



Hot Reloading Shaders (GPU)

- For graphic shaders (separate compiled programs that execute on the GPU) -- hot reloading is fairly standard to help improve iteration time
- **Hot reload:** Ability to recompile a portion of the program while the program is still running
 - https://antongerdelan.net/opengl/shader_hot_reload.html

Hot Reload (CPU Side) (1/2)

- What you can do on the GPU, you can of course do on the CPU
- I prototyped a little system to recompile and rebuild individual modules on the fly
 - Effectively allows me to use D as my scripting language for compiled code and maximum performance
- In D we have a great option, because I can recompile very fast using DMD -- also have the option to use the GDC or LDC compilers otherwise to generate optimized code to reload.

```
11 /// Compile a shared library:
12 /// dmd module2.d -of=libmodule2.so -shared -fPIC -defaultlib=libphobos2.so -L-rpath=.
13 void* RebuildAndReload(string modulename){
14     string dfilename = modulename~".d";
15     string sharedlibfilenamepath = "./lib"~modulename~".so";
16
17     writeln("dfilename:\t\t",dfilename);
18     writeln("sharedlibfilename:\t",sharedlibfilenamepath);
19
20     // Compile the individual module
21     // Note: 'execute' does block until finished.
22     auto dmd = execute(["dmd", dfilename, "-of=~sharedlibfilenamepath", "-shared", "-fPIC", "-defaultlib=libphobos2.so", "-L-rpath=."]);
23
24     // Check for any errors
25     // Note: Because D can execute code at compile-time,
26     then
27     // we need to consider writing out the output d
```

Hot reload with shared libraries

Note: Some care needed if you allocate in shared libraries (Work in progress to do so safely)

Hot Reload (CPU Side) (2/2)

- So where this became handy was in the little ‘callback’ system I had
 - I could trigger a RebuildAndReload and add (or remove) callbacks to my system to change behavior without having to stop.
 - D’s compile times are more than fast enough for this small project -- but I like speed!
- I know there have also been previous efforts with LDC2 with @dynamicCompile traits
 - These features are certainly appreciated, and perhaps worth taking a further look at.

```
11 // Compile a shared library:
12 // dmd module2.d -of=libmodule2.so -shared -fPIC -defaultlib=libphobos2.so -L-rpath=.
13 void* RebuildAndReload(string modulename){
14     string dfilename = modulename~".d";
15     string sharedlibfilepath = "./lib"~modulename~".so";
16
17     writeln("dfilename:\t\t",dfilename);
18     writeln("sharedlibfilepath:\t",sharedlibfilepath);
19
20     // Compile the individual module
21     // Note: 'execute' does block until finished.
22     auto dmd = execute(["dmd", dfilename, "-of="~sharedlibfilepath, "-shared", "-fPIC", "-defaultlib=libphobos2.so", "-L-rpath=."]);
23
24     // Check for any errors
25     // Note: Because D can execute code at compile-time,
26     then
27     // we need to consider writing out the output d
```

A Few Other Things Handy Things (1/3)

- Post condition and ‘invariant’ have been useful constructs in my code for early exit
 - e.g.
 - Checking for NaN and ensure we are always in a good state after vector operations.
 - Anytime I am creating unit Vectors (and I do so frequently) -- it’s good to not divide by 0!
 - <https://dlang.org/spec/contracts.html>

A Few Other Things Handy Things (2/3)

- Easy to template code
 - Able to make function templates to eliminate branches in code (i.e. which shader type to compile at run-time
 - Instead make a templated function
 - Also can apply a 'template constraint' to avoid illegal types from being created
 - Enforced at compile-time, again so you don't have to pay the cost if you compile your shaders at run-time.

A Few Other Things Handy Things (3/3)

- In many places where I have enums in OpenGL, I can I can template them away -- often making my codebase more robust.
 - Code can be self-documenting for what 'enum' types are legal
 - (i.e. Use a function as a template constraint to check valid enums)

```
*/  
GLuint CompileShader(GLuint type)(char[] source)  
    if(type == GL_VERTEX_SHADER || type == GL_FRAGMENT_SHADER)  
    {
```

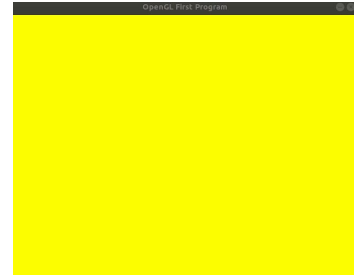
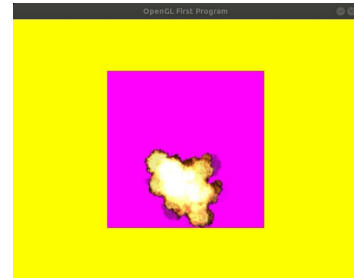
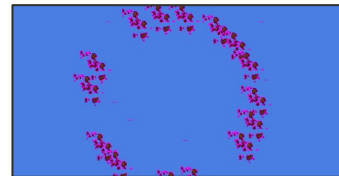
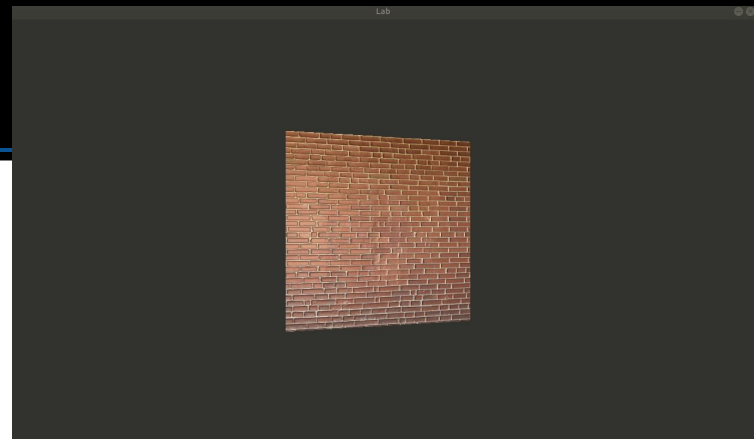
```
92         if(type == GL_VERTEX_SHADER){  
93             writeln("ERROR: GL_VERTEX_SHADER compilation failed!\n", errorMessages, "\n");  
94             writeln("====failed:", source);  
95         }else if(type == GL_FRAGMENT_SHADER){  
96             writeln("ERROR: GL_FRAGMENT_SHADER compilation failed!\n", errorMessages, "\n");  
97             writeln("====failed:", source);  
98         }  
99     }
```

```
92     writeln("ERROR: ~to!string(type)~" compilation failed!\n", errorMessages, "\n");  
93     writeln("====failed:", source);
```

A Few More Pictures and Thoughts Before We Leave

More to explore for me

- I did some more experiments and some porting of some of my C++ code to the right to Dlang
 - (top-right normal mapping)
 - (bunny - diffuse shading)
 - (Particles -- demonstrating instanced draw calls)
- **The other goal here** is to just generate lots of sample code for other folks to draw from.
 - I'm otherwise implementing a column-ordered math library and exploring the vector extensions for math (and other use cases)
 - See: <https://dlang.org/spec/simd.html>
 - I've been also learning Vulkan and WebGLGPU otherwise see the experience with DLang.



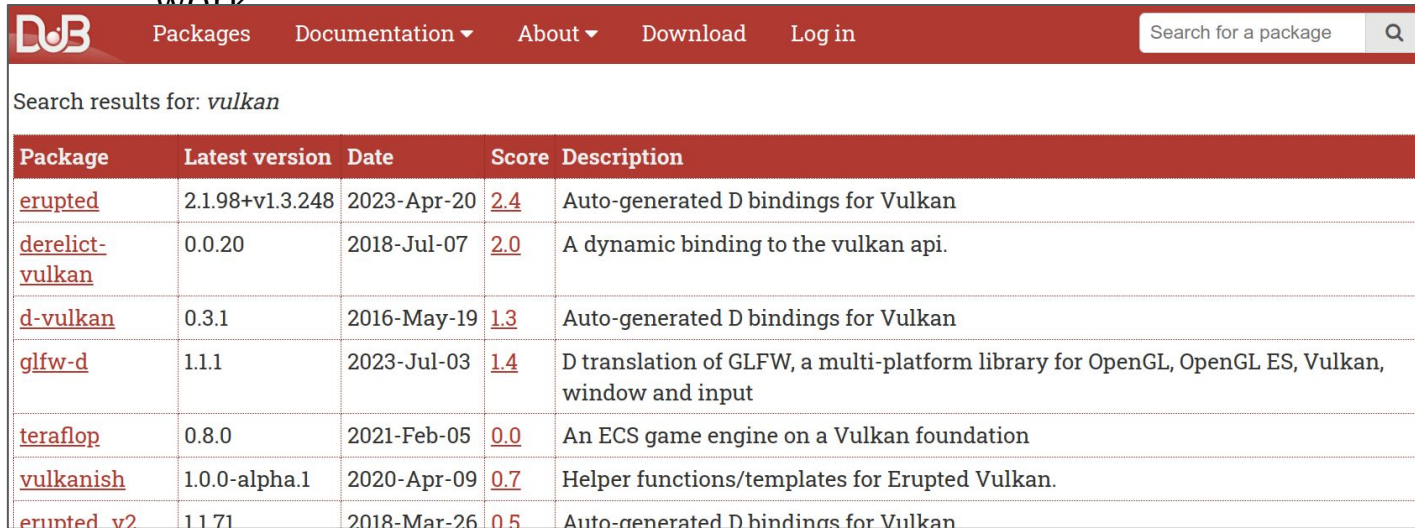
Toolstack for today's work

- Project uses:
 - bincbc-sdl
 - bindbc-opengl
 - bindbc-loader
 - A combination of my own math library and some portions of inmath/gl3n libraries

Wrap Up and More Resources

OpenGL, Vulkan, etc. Bindings

- Many graphics API bindings otherwise exist
 - If you've previously worked in these APIs, they will be relatively straightforward to get back into
 - Folks should consider these libraries
 - Raylib-d, glfw-d, etc. are other excellent libraries for starting to do graphics work



The screenshot shows the DUB website interface. At the top, there is a navigation bar with links for 'Packages', 'Documentation', 'About', 'Download', and 'Log in'. A search bar on the right contains the text 'Search for a package' and a magnifying glass icon. Below the navigation bar, the search results for 'vulkan' are displayed. The results are presented in a table with the following columns: Package, Latest version, Date, Score, and Description. The table lists several packages, including 'erupted', 'derelict-vulkan', 'd-vulkan', 'glfw-d', 'teraflop', 'vulkanish', and 'erupted v2'.

Package	Latest version	Date	Score	Description
erupted	2.1.98+v1.3.248	2023-Apr-20	2.4	Auto-generated D bindings for Vulkan
derelict-vulkan	0.0.20	2018-Jul-07	2.0	A dynamic binding to the vulkan api.
d-vulkan	0.3.1	2016-May-19	1.3	Auto-generated D bindings for Vulkan
glfw-d	1.1.1	2023-Jul-03	1.4	D translation of GLFW, a multi-platform library for OpenGL, OpenGL ES, Vulkan, window and input
teraflop	0.8.0	2021-Feb-05	0.0	An ECS game engine on a Vulkan foundation
vulkanish	1.0.0-alpha.1	2020-Apr-09	0.7	Helper functions/templates for Erupted Vulkan.
erupted v2	11.71	2018-Mar-26	0.5	Auto-generated D bindings for Vulkan

Some High Level Takeaways

- D for real-time graphics work I have found excellent for improving my iteration time
 - I get to think more about solving interesting problems and building useful systems.
- The toolset is effectively the same if you're already familiar with graphics programming (i.e. C-APIs like OpenGL, Vulkan, etc.)
- I hope this little talk encourages other folks to give the D language a try otherwise for graphics and game work!



YouTube

- I am actively adding more lessons about the D programming language
 - <https://www.youtube.com/c/MikeShah>
- **Eventually** I will start **adding graphics** to this playlist (or a new playlist) on my channel.

[Episode 0] Series Teaser

matrix.py
matrix.d
DLang

D Language (DLang) Programming

Mike Shah

Public

85 videos 19,883 views Last updated on Dec 22, 2023

▶ Play all ⌘ Shuffle

A full playlist on learning the D Programming language. A great starting place for beginners to start, as we'll start from the very beginning. This playlist will also move towards more advanced features of the language as well - find it all here!

Sort All Videos Shorts

- [Episode 0] Series Teaser [Dlang Series Teaser] Dlang versus Python speed comparison (Matrix Multiply) Mike Shah · 4.7K views · 1 year ago
- [Episode 0] Series Teaser Dlang versus Python (Matrix Multiply) #shorts series intro Mike Shah · 2.2K views · 1 year ago
- [Episode 1] What Is DLang? [Dlang Episode 1] The D Programming Language - dlang Mike Shah · 5.5K views · 1 year ago
- [Episode 2] DLang Install on Linux [Dlang Episode 2] D Language - setup on Linux (dmd, gdc, and ldc2 shown!) Mike Shah · 1.8K views · 1 year ago
- [Episode 3] DLang Install on Mac (M1 Shown) [Dlang Episode 3] D Language - setup on Mac (Shown on Mac M1, DMD and LDC2) Mike Shah · 1.1K views · 1 year ago
- [Episode 4] DLang Install on Windows [Dlang Episode 4] D Language - DMD command line and Visual D for Visual Studio (DMD and LDC2) Mike Shah · 1.5K views · 1 year ago
- [Episode 5] Hello World (Explained) [Dlang Episode 5] The Anatomy of a Hello World Application Mike Shah · 1.4K views · 1 year ago

<https://www.youtube.com/playlist?list=PLvv0ScY6vfd9Fso-3cB4CGnSIW0E4btJV>

Further resources and training materials

- Tons of talks (Games, graphics, servers, etc.)
 - <https://wiki.dlang.org/Videos#Tutorials>
 - Talks more recently from Ethan Watson, Manu Evans, Hipreme Engine for folks who want to see game stuff.
- My 'Graphics Related' talks on Ray Tracers
 - DConf '22: Ray Tracing in (Less Than) One Weekend with DLang -- Mike Shah
 - <https://www.youtube.com/watch?v=nCIB8df7q2g>
 - DConf Online '22 - Engineering a Ray Tracer on the Next Weekend with DLang
 - <https://www.youtube.com/watch?v=MFhTRiobWfU>



The Case for Graphics

-- in **DLang: Part 2 - Tech Demo**
with Mike Shah

Social: [@MichaelShah](#)

Web: [mshah.io](#)

Courses: [courses.mshah.io](#)

 **YouTube**

[www.youtube.com/c/MikeShah](#)

<http://tinyurl.com/mike-talks>

14:30 - 15:00 Fri, Sept 19, 2024

30 minutes | Introductory Audience

Extra