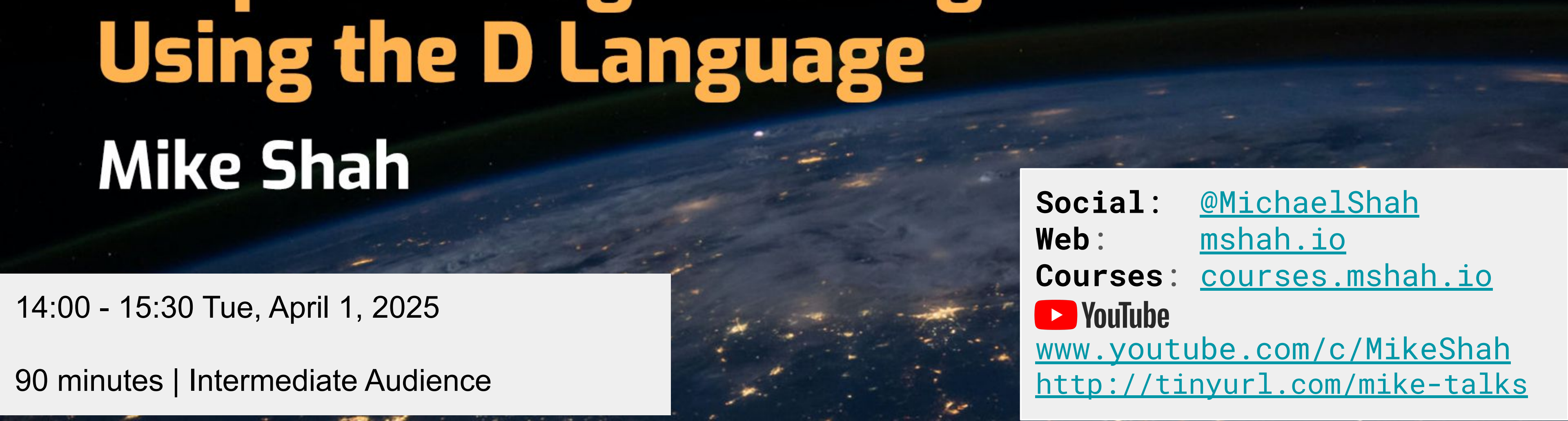
accu
conference
2025



The Case for Graphics Programming Using the D Language

Mike Shah

14:00 - 15:30 Tue, April 1, 2025

90 minutes | Intermediate Audience

Social: @MichaelShah
Web: mshah.io
Courses: courses.mshah.io
YouTube
www.youtube.com/c/MikeShah
http://tinyurl.com/mike-talks

Abstract (Which you already read :))

Talk Abstract: ‘write fast, read fast, and run fast’ is the mantra found on the D programming language homepage (https://dlang.org/). Did you notice a word game and graphics programmers love that is used 3 times? Fast! In this talk I will show examples of how I have used programming techniques that available in the D programming language to build graphics applications and games. Throughout this talk I will showcase graphics demos in the D language, and more generally programming language features that 'changed' my approach to graphics programming. The greater goal of this talk, is to show attendees why there can be a payoff of using non-mainstream programming languages in specific domains. After all -- why not have a competitive advantage?

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

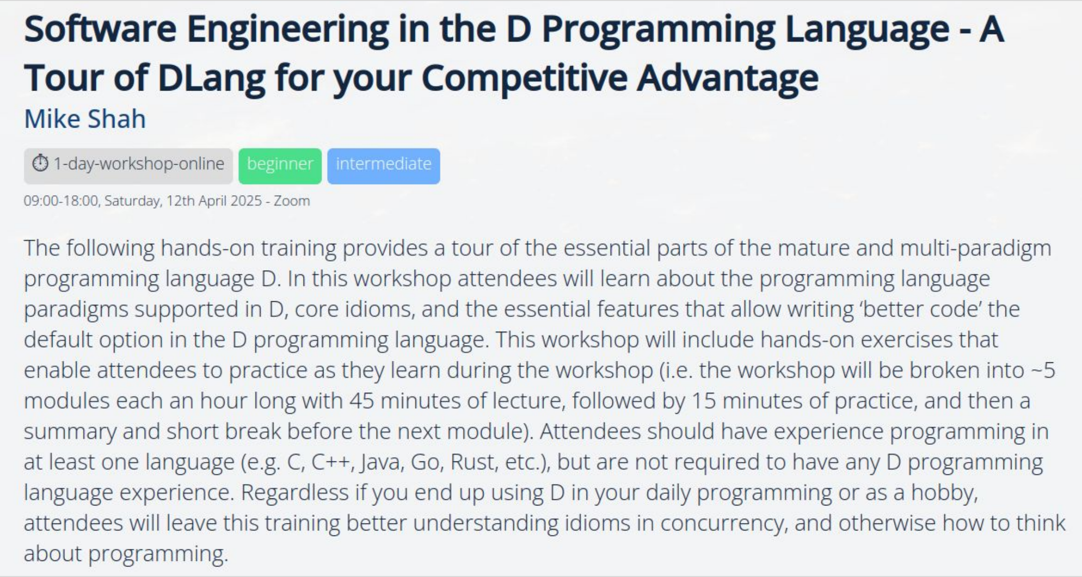
http://tinyurl.com/mike-talks

Your Tou

- **Current Role:** Teaching Fac

  (Previously Teaching Faculty at Northeastern Uni

  - **Teach/Research:** computer sys engine development, and softw
- **Available for:**
  - **Contract work** in Gami
    - e.g. tool building,
  - **Technical training** (C++, **D**, and topics in P
- **Fun:**
  - Guitar, running/weight games, and cooking are



Software Engineering in the D Programming Language - A Tour of DLang for your Competitive Advantage

Mike Shah

⏱ 1-day-workshop-online beginner intermediate

09:00-18:00, Saturday, 12th April 2025 - Zoom

The following hands-on training provides a tour of the essential parts of the mature and multi-paradigm programming language D. In this workshop attendees will learn about the programming language paradigms supported in D, core idioms, and the essential features that allow writing 'better code' the default option in the D programming language. This workshop will include hands-on exercises that enable attendees to practice as they learn during the workshop (i.e. the workshop will be broken into ~5 modules each an hour long with 45 minutes of lecture, followed by 15 minutes of practice, and then a summary and short break before the next module). Attendees should have experience programming in at least one language (e.g. C, C++, Java, Go, Rust, etc.), but are not required to have any D programming language experience. Regardless if you end up using D in your daily programming or as a hobby, attendees will leave this training better understanding idioms in concurrency, and otherwise how to think about programming.

Come join me April 12th online if you enjoy today's talk!

Non-Academic Courses
courses.mshah.io
Conference Talks
http://tinyurl.com/mike-talks

Quick Poll: Raise your hand if you have heard of the D programming language?

Quick Poll: Raise your hand if you used the D1 (2001 to ~2007) programming language?

Quick Poll: Raise your hand if you used the D2 (2007 to now) programming language?

Quick Poll: Raise your hand if you actively use D for some project (hobby / commercial / etc)?

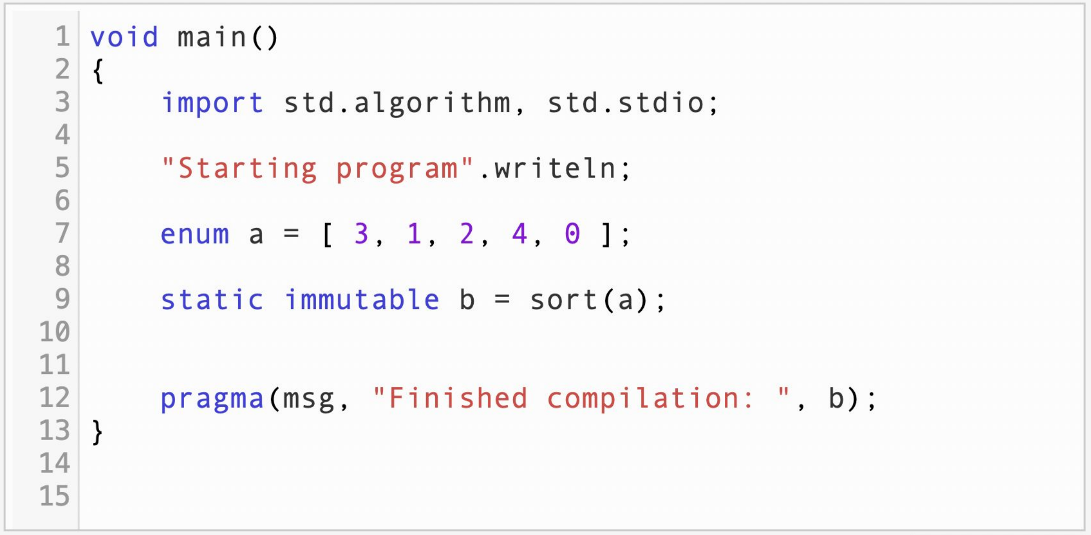
A First Impression

La premiere impression

첫인상

Pop Quiz: (l’examen surprise!) (1/8)

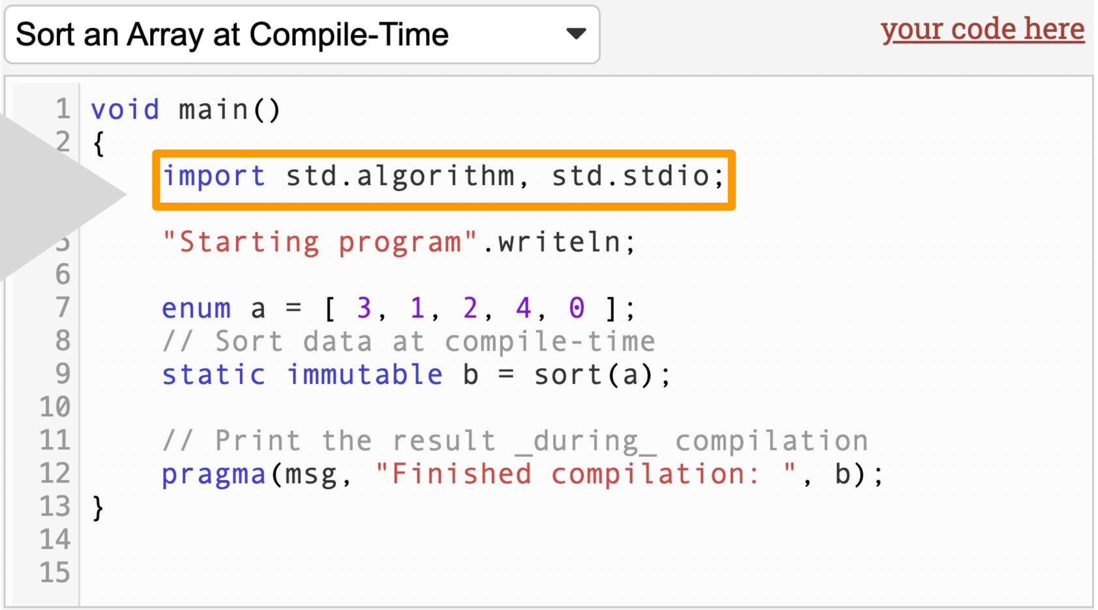
- Let’s take a look at an example of D code
  - I’ll give everyone a minute to think about or guess what this program does
- So… what does this program do?



```d
void main()
{
    import std.algorithm, std.stdio;

    "Starting program".writeln;

    enum a = [ 3, 1, 2, 4, 0 ];

    static immutable b = sort(a);


    pragma(msg, "Finished compilation: ", b);
}
```

Pop Quiz: (l’examen surprise!) (2/8)

- Line 3:
  - There’s a built-in **standard library** (named ‘Phobos’)
  - There’s a **module** system .
- Line 5:
  - Function call using **uniform function call syntax** (UFCS)
- Line 7:
  - enum constant, **evaluated at compile-time**
- Line 9:
  - **immutable static** data stored in b
- Line 12:
  - **pragma** outputs value after compilation (before runtime)

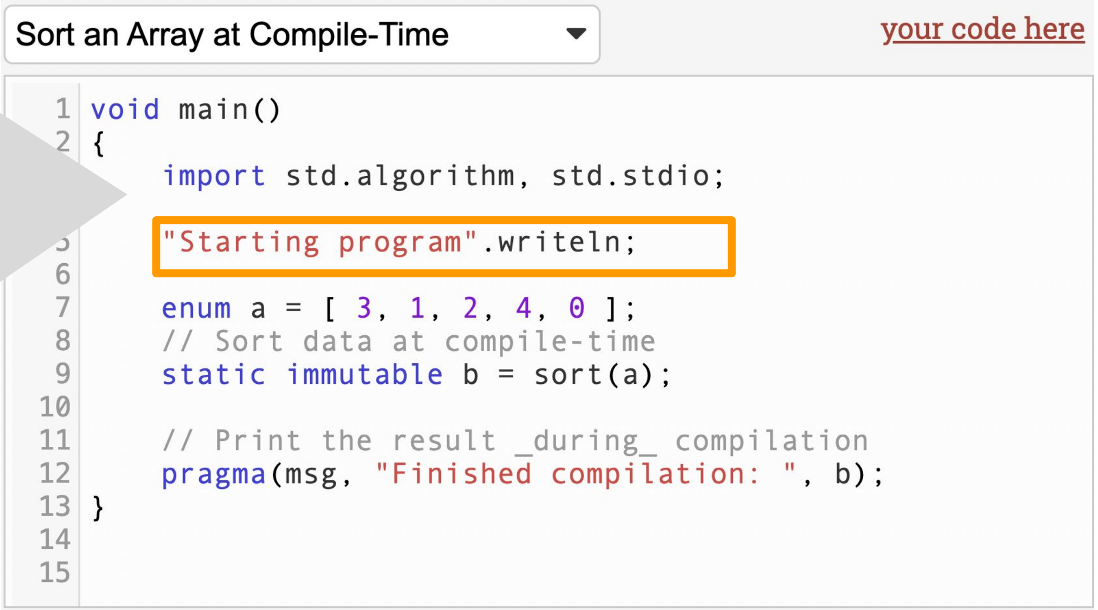


```
 1 void main()
 2 {
 3     import std.algorithm, std.stdio;
 4 
 5     "Starting program".writeln;
 6 
 7     enum a = [ 3, 1, 2, 4, 0 ];
 8     // Sort data at compile-time
 9     static immutable b = sort(a);
10 
11     // Print the result _during_ compilation
12     pragma(msg, "Finished compilation: ", b);
13 }
14 
15 
```

One of the first examples on the www.dlang.org webpage - sorting an array -- at compile-time!

Pop Quiz: (l'examen surprise!) (3/8)

- Line 3:
  - There's a built-in **standard library** (named 'Phobos')
  - There's a **module** system .
- Line 5:
  - Function call using **uniform function call syntax** (UFCS)
- Line 7:
  - enum constant, **evaluated at compile-time**
- Line 9:
  - **immutable static** data stored in b
- Line 12:
  - **pragma** outputs value after compilation (before runtime)

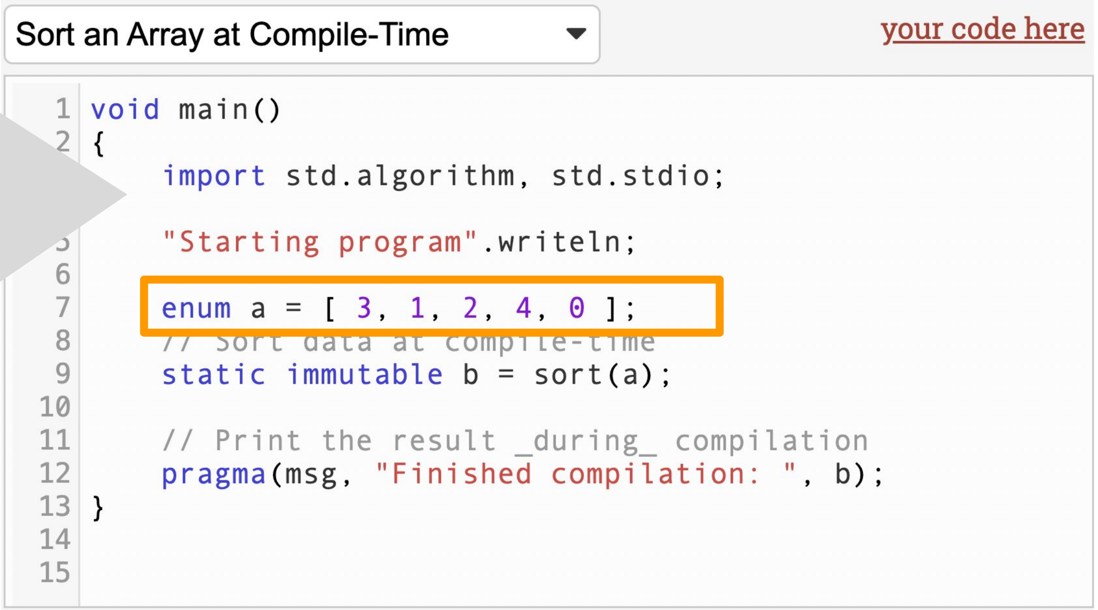


```
 1 void main()
 2 {
 3     import std.algorithm, std.stdio;
 4 
 5     "Starting program".writeln;
 6 
 7     enum a = [ 3, 1, 2, 4, 0 ];
 8     // Sort data at compile-time
 9     static immutable b = sort(a);
10 
11     // Print the result _during_ compilation
12     pragma(msg, "Finished compilation: ", b);
13 }
14 
15 
```

One of the first examples on the www.dlang.org webpage - sorting an array -- at compile-time!

Pop Quiz: (l'examen surprise!) (4/8)

- Line 3:
  - There's a built-in **standard library** (named 'Phobos')
  - There's a **module** system .
- Line 5:
  - Function call using **uniform function call syntax** (UFCS)
- Line 7:
  - enum constant, **evaluated at compile-time**
- Line 9:
  - **immutable static** data stored in b
- Line 12:
  - **pragma** outputs value after compilation (before runtime)

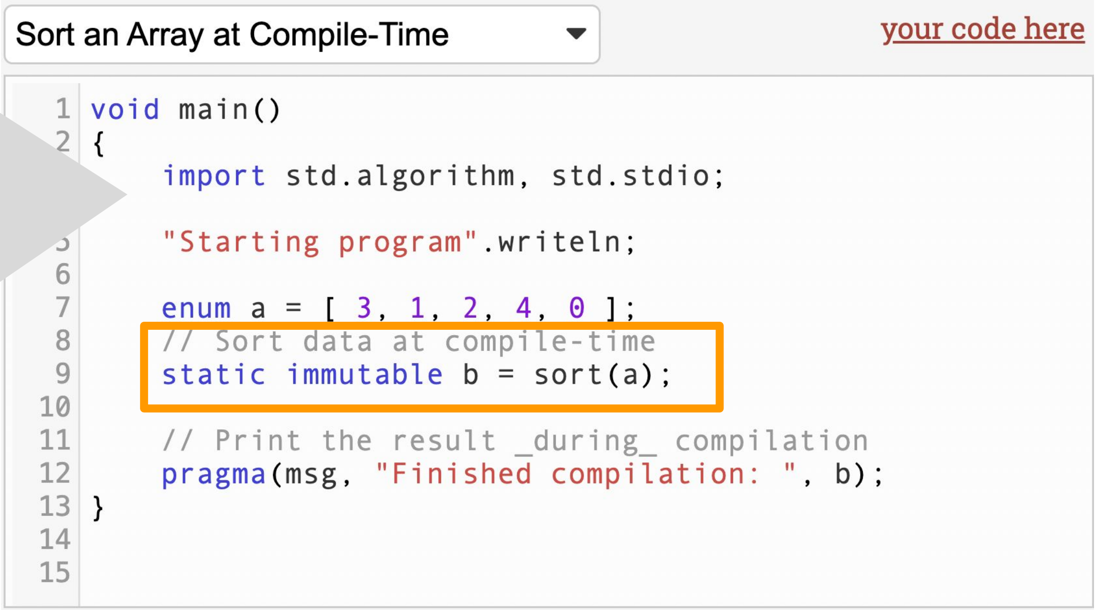


```
 1 void main()
 2 {
       import std.algorithm, std.stdio;

 5     "Starting program".writeln;
 6
 7     enum a = [ 3, 1, 2, 4, 0 ];
 8     // Sort data at compile-time
 9     static immutable b = sort(a);
10
11     // Print the result _during_ compilation
12     pragma(msg, "Finished compilation: ", b);
13 }
14
15
```

One of the first examples on the www.dlang.org webpage - sorting an array -- at compile-time!

Pop Quiz: (l’examen surprise!) (5/8)

- Line 3:
  - There’s a built-in **standard library** (named ‘Phobos’)
  - There’s a **module** system .
- Line 5:
  - Function call using **uniform function call syntax** (UFCS)
- Line 7:
  - enum constant, **evaluated at compile-time**
- Line 9:
  - **immutable static** data stored in b
- Line 12:
  - **pragma** outputs value after compilation (before runtime)

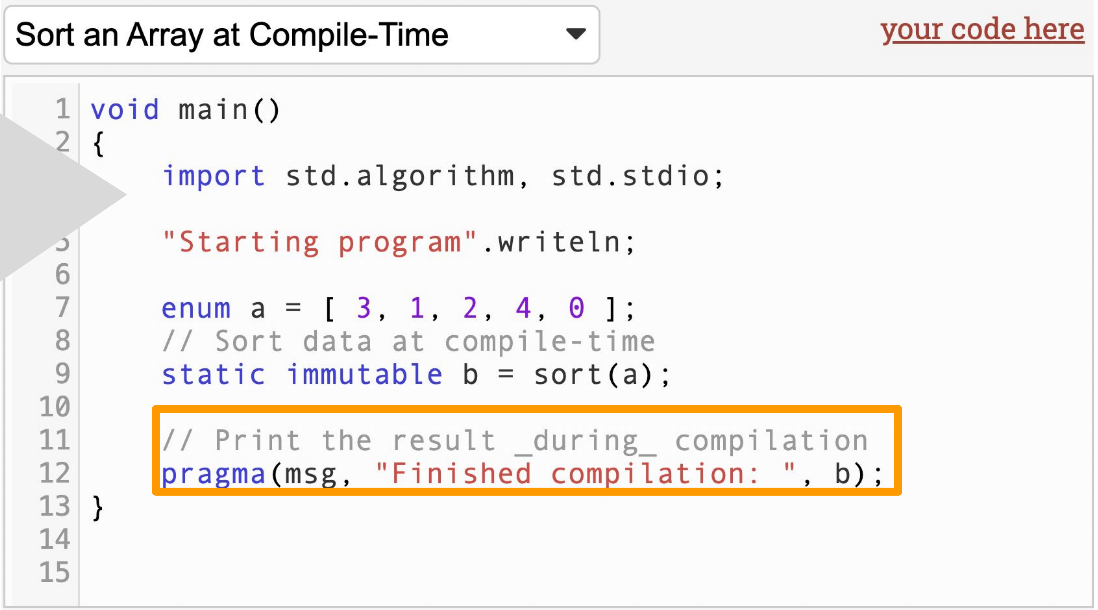


```
 1 void main()
 2 {
        import std.algorithm, std.stdio;

 5      "Starting program".writeln;
 6
 7      enum a = [ 3, 1, 2, 4, 0 ];
 8      // Sort data at compile-time
 9      static immutable b = sort(a);
10
11      // Print the result _during_ compilation
12      pragma(msg, "Finished compilation: ", b);
13 }
14
15
```

One of the first examples on the www.dlang.org webpage - sorting an array -- at compile-time!

Pop Quiz: (l’examen surprise!) (6/8)

- Line 3:
  - There’s a built-in **standard library** (named ‘Phobos’)
  - There’s a **module** system .
- Line 5:
  - Function call using **uniform function call syntax** (UFCS)
- Line 7:
  - enum constant, **evaluated at compile-time**
- Line 9:
  - **immutable static** data stored in b
- Line 12:
  - **pragma** outputs value after compilation (before runtime)

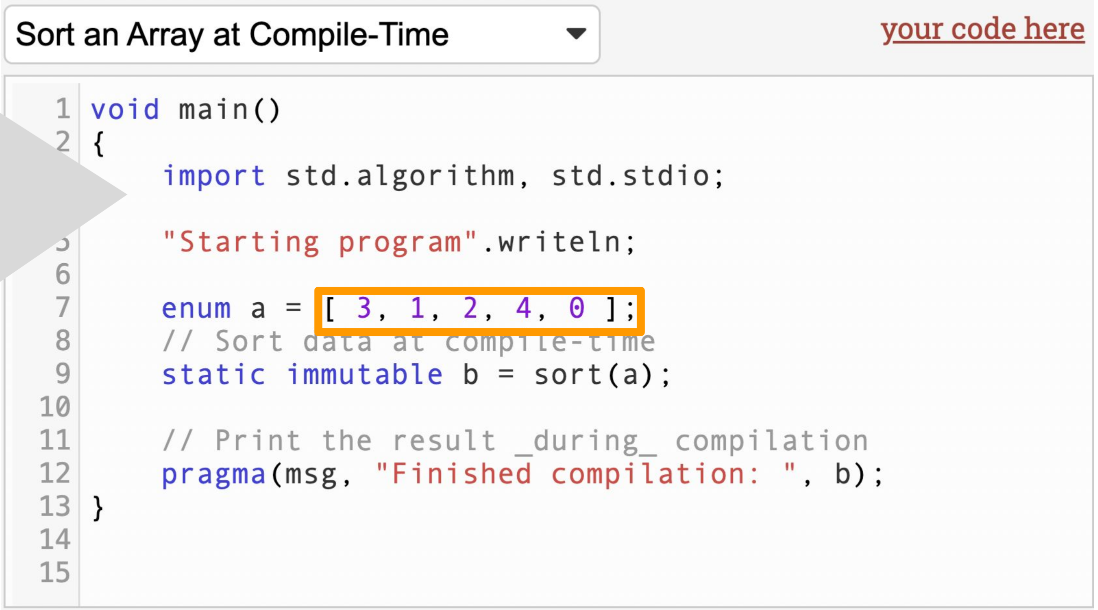


```d
 1 void main()
 2 {
 3     import std.algorithm, std.stdio;
 4
 5     "Starting program".writeln;
 6
 7     enum a = [ 3, 1, 2, 4, 0 ];
 8     // Sort data at compile-time
 9     static immutable b = sort(a);
10
11     // Print the result _during_ compilation
12     pragma(msg, "Finished compilation: ", b);
13 }
14
15
```

One of the first examples on the www.dlang.org webpage - sorting an array -- at compile-time!

Pop Quiz: (l’examen surprise!) (7/8)

- Line 7:
  - This is a **fixed-size array**.
  - We can slice into it
    - e.g.
    - a[0 .. 2] returns [3,1,2]
  - Arrays (whether dynamic or static) know their **‘length’** and store the **‘ptr’** together.



Sort an Array at Compile-Time

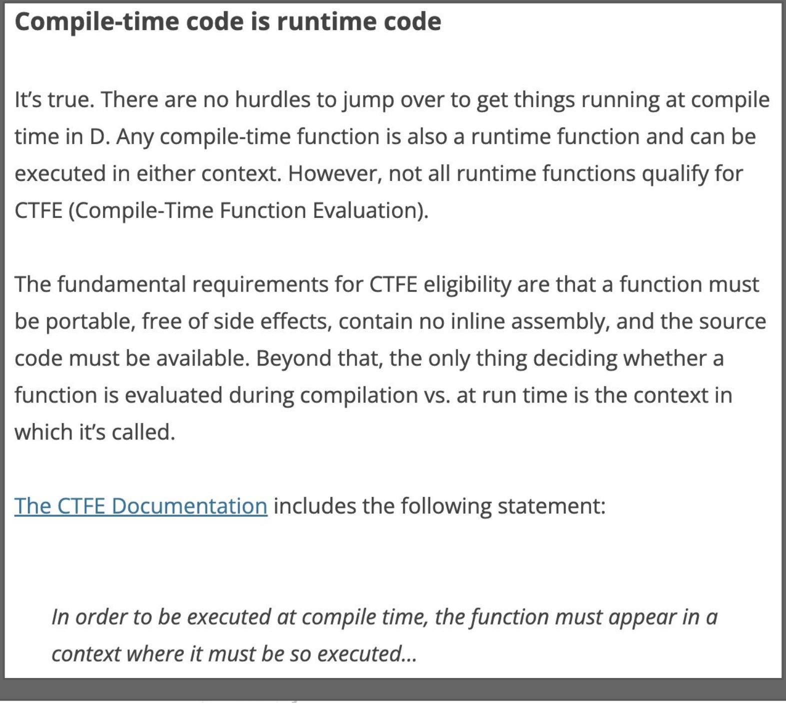
```
 1  void main()
 2  {
        import std.algorithm, std.stdio;

 5      "Starting program".writeln;
 6
 7      enum a = [ 3, 1, 2, 4, 0 ];
 8      // Sort data at compile-time
 9      static immutable b = sort(a);
10
11      // Print the result _during_ compilation
12      pragma(msg, "Finished compilation: ", b);
13  }
14
15
```

One of the first examples on the www.dlang.org webpage - sorting an array -- at compile-time!

Why you might care to look?

- D tries to **execute as much as possible at compile-time**
  - And the code...just looks like regular code!
- Compile-time execution saves the user time at run-time -- big win!

- https://dlang.org/blog/2017/06/05/compile-time-sort-in-d/
- https://tour.dlang.org/tour/en/gems/compile-time-function-evaluation-ctfe



Compile-time code is runtime code

It's true. There are no hurdles to jump over to get things running at compile time in D. Any compile-time function is also a runtime function and can be executed in either context. However, not all runtime functions qualify for CTFE (Compile-Time Function Evaluation).

The fundamental requirements for CTFE eligibility are that a function must be portable, free of side effects, contain no inline assembly, and the source code must be available. Beyond that, the only thing deciding whether a function is evaluated during compilation vs. at run time is the context in which it's called.

The CTFE Documentation includes the following statement:

> *In order to be executed at compile time, the function must appear in a context where it must be so executed...*

your code here

```
.stdio;
n;
ing_ompilation
npilation: ", b);
```

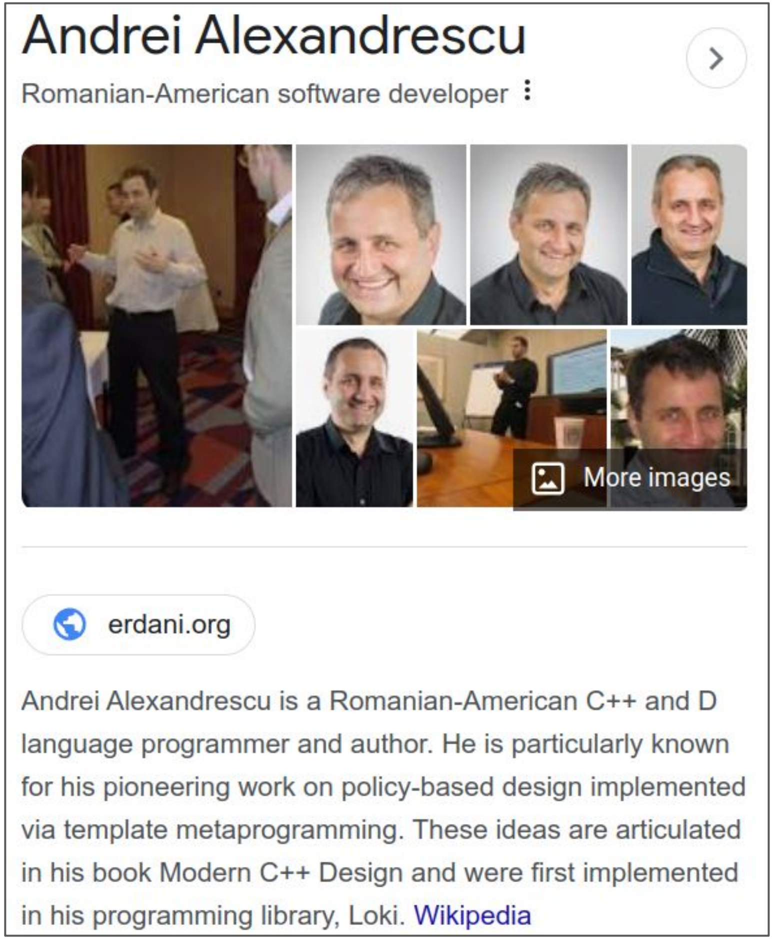
compilation

- This program does most of its work (the working) at compile-time!

One of the first examples on the www.dlang.org webpage - sorting an array -- at compile-time!

The Case for D

(By Andrei Alexandrescu)



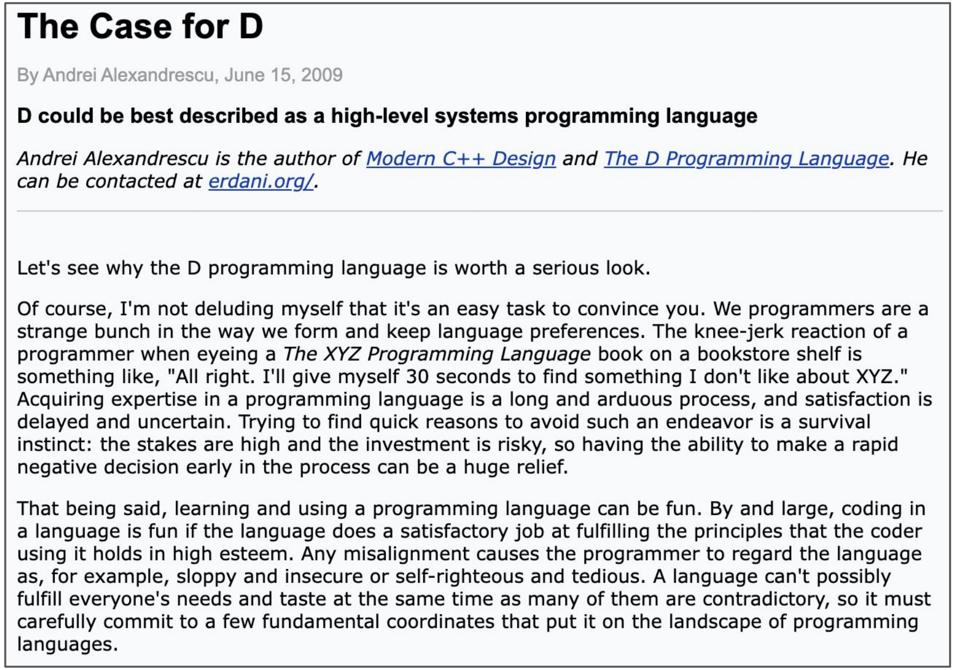
Andrei Alexandrescu

Romanian-American software developer

Andrei Alexandrescu is a Romanian-American C++ and D language programmer and author. He is particularly known for his pioneering work on policy-based design implemented via template metaprogramming. These ideas are articulated in his book Modern C++ Design and were first implemented in his programming library, Loki. Wikipedia

The Case for DLang (1/3)

- Nearly 16 years ago Andrei Alexandrescu wrote 'The Case for D' (posted on Dr. Dobb's journal and other sources)
  - The D language has continued to improve on its strong foundations since that time!
- Andrei summarizes DLang as:
  - *"D could be best described as a high-level systems programming language"*



The Case for D

By Andrei Alexandrescu, June 15, 2009

D could be best described as a high-level systems programming language

Andrei Alexandrescu is the author of Modern C++ Design and The D Programming Language. He can be contacted at erdani.org/.

Let's see why the D programming language is worth a serious look.

Of course, I'm not deluding myself that it's an easy task to convince you. We programmers are a strange bunch in the way we form and keep language preferences. The knee-jerk reaction of a programmer when eyeing a *The XYZ Programming Language* book on a bookstore shelf is something like, "All right. I'll give myself 30 seconds to find something I don't like about XYZ." Acquiring expertise in a programming language is a long and arduous process, and satisfaction is delayed and uncertain. Trying to find quick reasons to avoid such an endeavor is a survival instinct: the stakes are high and the investment is risky, so having the ability to make a rapid negative decision early in the process can be a huge relief.

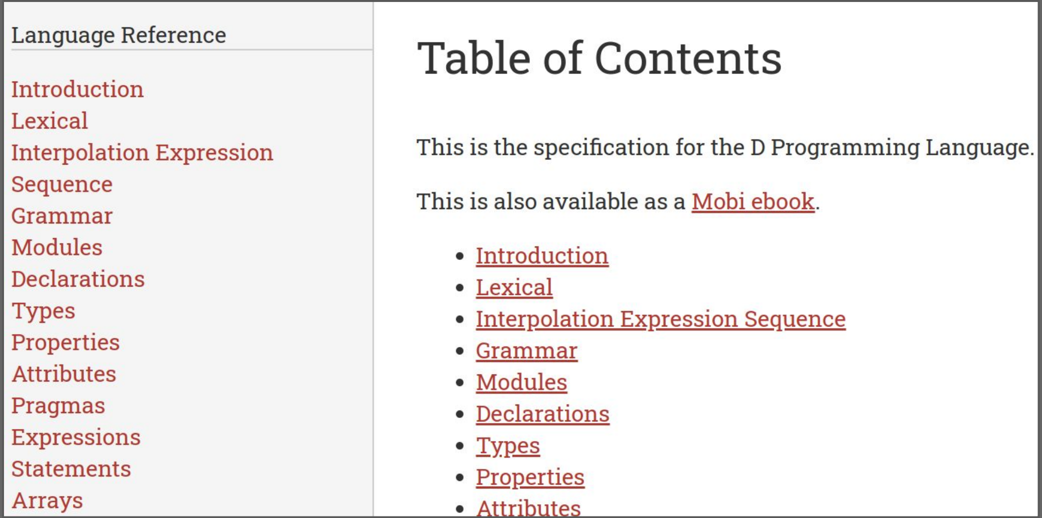
That being said, learning and using a programming language can be fun. By and large, coding in a language is fun if the language does a satisfactory job at fulfilling the principles that the coder using it holds in high esteem. Any misalignment causes the programmer to regard the language as, for example, sloppy and insecure or self-righteous and tedious. A language can't possibly fulfill everyone's needs and taste at the same time as many of them are contradictory, so it must carefully commit to a few fundamental coordinates that put it on the landscape of programming languages.

https://web.archive.org/web/20121020122307/https://www.drdobbs.com/parallel/the-case-for-d/217801225

The Cas

- Nearly
Alexa
Case f
Dobb'
sourc
  - 15
ha
its
- *Andrei summariz*
as:
  - *"D could be best described as a high-level systems programming language"*

At a glance D has many features: https://dlang.org/spec/spec.html



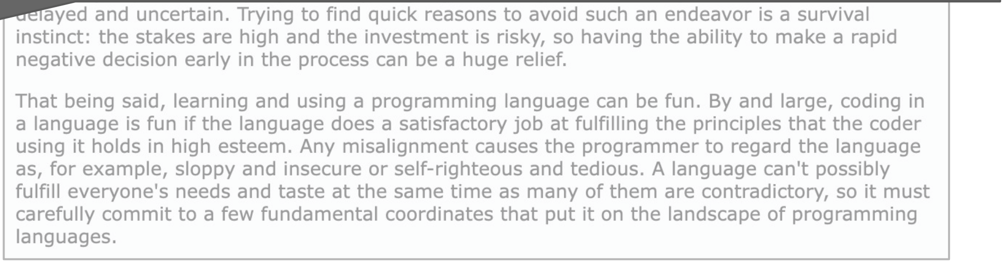
Language Reference

- Introduction
- Lexical
- Interpolation Expression
- Sequence
- Grammar
- Modules
- Declarations
- Types
- Properties
- Attributes
- Pragmas
- Expressions
- Statements
- Arrays

Table of Contents

This is the specification for the D Programming Language.

This is also available as a Mobi ebook.



delayed and uncertain. Trying to find quick reasons to avoid such an endeavor is a survival instinct: the stakes are high and the investment is risky, so having the ability to make a rapid negative decision early in the process can be a huge relief.

That being said, learning and using a programming language can be fun. By and large, coding in a language is fun if the language does a satisfactory job at fulfilling the principles that the coder using it holds in high esteem. Any misalignment causes the programmer to regard the language as, for example, sloppy and insecure or self-righteous and tedious. A language can't possibly fulfill everyone's needs and taste at the same time as many of them are contradictory, so it must carefully commit to a few fundamental coordinates that put it on the landscape of programming languages.

https://web.archive.org/web/20121020122307/https://www.drdobbs.com/parallel/the-case-for-d/217801225

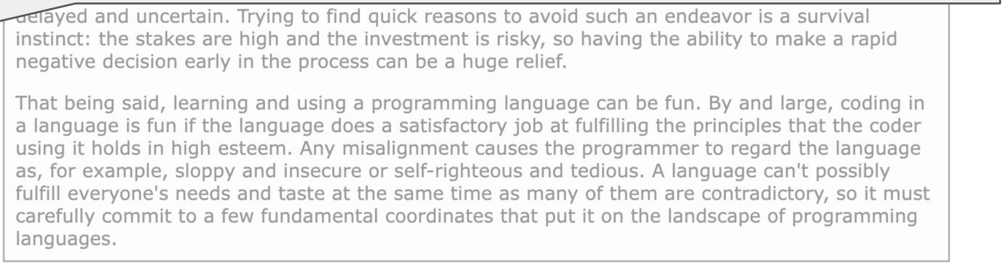
[1] and more here: https://dlang.org/comparison.html

The Cas

At a glance -- **Dlang is** :

- A **compiled** language (3 freely available compilers)
  - Extremely **fast compilation** with - DMD Compiler
  - Two additional compilers with LLVM (LDC) and GCC (GDC) backends
- **statically typed** language
- **Plays well** with C, C++, Obj-C
  - Embedded C compiler - ImportC
  - e.g. of interoperation with C++ (Interfacing with C++)
- **Many modern language features:**
  - Ranges (and foreach), Compile-Time Function Execution (CTFE), Array slicing, lambda's, mixins, contracts, unit testing, template constraints, multiple memory allocation strategies, and more[1].

- Nearly
  Alexa
  Case f
  Dobb'
  sourc
  - 15
    ha
    its
- *Andrei summariz*
  as:
  - *"D could be best described as a high-level systems programming language"*



delayed and uncertain. Trying to find quick reasons to avoid such an endeavor is a survival instinct: the stakes are high and the investment is risky, so having the ability to make a rapid negative decision early in the process can be a huge relief.

That being said, learning and using a programming language can be fun. By and large, coding in a language is fun if the language does a satisfactory job at fulfilling the principles that the coder using it holds in high esteem. Any misalignment causes the programmer to regard the language as, for example, sloppy and insecure or self-righteous and tedious. A language can't possibly fulfill everyone's needs and taste at the same time as many of them are contradictory, so it must carefully commit to a few fundamental coordinates that put it on the landscape of programming languages.

https://web.archive.org/web/20121020122307/https://www.drdobbs.com/parallel/the-case-for-d/217801225

[1] and more here: https://dlang.org/comparison.html

My Goal Today

- Is to convince you **D is the best programming language!**😉
- (next slide)

Real Goal for you Today

- ~~Is to convince you **D is the best programming language**!😉~~
- **...**okay I know it is April 1st -- *so that's not quite what I feel I need to do.* 😉
- My **goal** is for you to expand your horizon, and decide if D will give you a **competitive advantage** for your project.
- Specifically today, I'll be looking at the **graphics programming** domain, where I think D has personally given me an advantage in iteration speed and performance
- Note:
  - This talk is not meant to teach you graphics from scratch, but rather focus on language features that made my life easier in the graphics domain.

The Case for D as a Graphics Programmer

(By Mike Shah)

The Case for D for graphics programming (1/2)

1. Most of the right defaults
   a. e.g. variables are initialized (or use =void to avoid .init values), const is transitive, casts must be explicit, arrays carry ‘length’ and ‘ptr’, thread local storage, etc.
2. Faster prototyping as a result of module system and excellent DMD compiler
   a. One can leverage the DMD frontend with LLVM and GCC backend for optimizations and targeting more platforms
3. Can generate fast code!
   a. SIMD vector extensions available https://dlang.org/spec/simd.html
   b. Multitasking support available [introduction here]:
      i. Threads, fibers, etc.
4. It’s fun to write code in DLang (my personal bias)

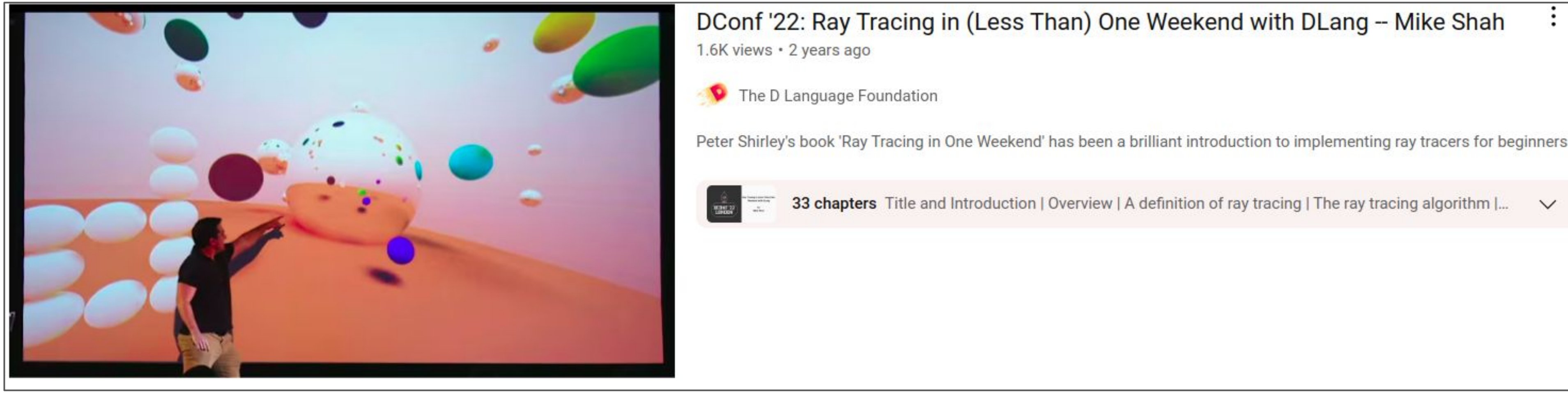
The Case for D for graphics programming (2/2)

1. Most of the right defaults
   a. e.g. variables are initialized (o e,
      casts must be explicit, arrays c
2. Faster prototyping as a resu D
   compiler
   a. (Can then leverage D frontend
      and target platforms)
3. Can generate fast code
   a. SIMD vector extensions availa
   b. Multitasking support available [introduction here]:
      i. Threads, fibers, etc.
4. **It’s fun to write code in DLang (my personal bias)**

I will show you! :)

My Case for D: Ray Tracing (non-real time graphics) (1/2)

- My case for D starts in 2022 when I built a ray tracer in D in a weekend (based on Peter Shirley's book)
  - The productivity of the language was encouraging as someone with a good background in C++



DConf '22: Ray Tracing in (Less Than) One Weekend with DLang -- Mike Shah

1.6K views • 2 years ago

The D Language Foundation

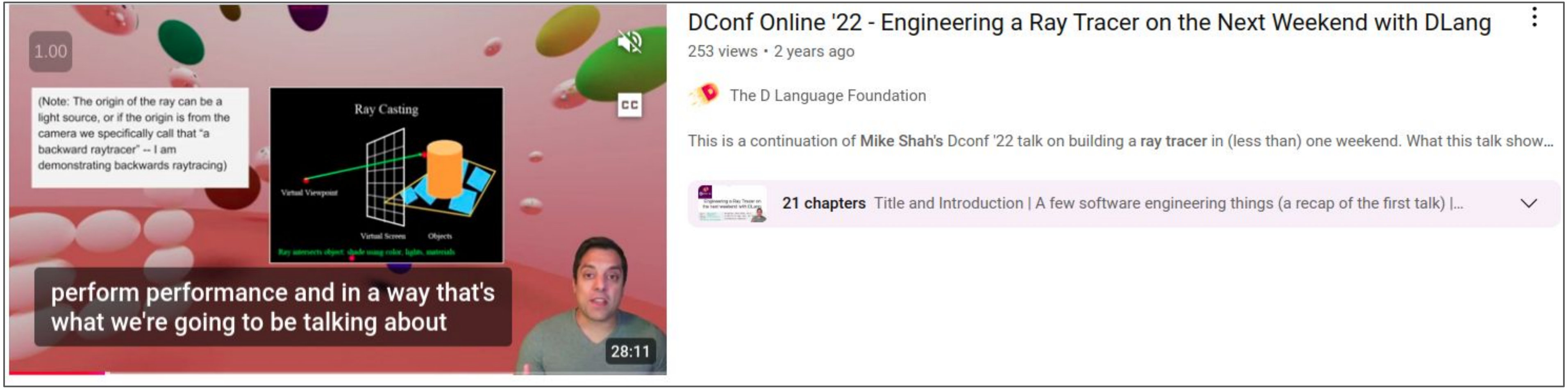
Peter Shirley's book 'Ray Tracing in One Weekend' has been a brilliant introduction to implementing ray tracers for beginners

33 chapters Title and Introduction | Overview | A definition of ray tracing | The ray tracing algorithm |...

https://www.youtube.com/watch?v=nCIB8df7q2g

My Case for D: Ray Tracing (non-real time graphics) (2/2)

- I was encouraged enough to then give a second talk a few months later in 2023, for which I really started to learn to use the D language more fully
  - So let me give you a highlight of some of my early insights to give you more of a taste of the D language.



(Note: The origin of the ray can be a light source, or if the origin is from the camera we specifically call that "a backward raytracer" -- I am demonstrating backwards raytracing)

perform performance and in a way that's what we're going to be talking about

DConf Online '22 - Engineering a Ray Tracer on the Next Weekend with DLang

253 views • 2 years ago

The D Language Foundation

This is a continuation of **Mike Shah's** Dconf '22 talk on building a **ray tracer** in (less than) one weekend. What this talk show...

21 chapters Title and Introduction | A few software engineering things (a recap of the first talk) |...

https://www.youtube.com/watch?v=MFhTRiobWfU

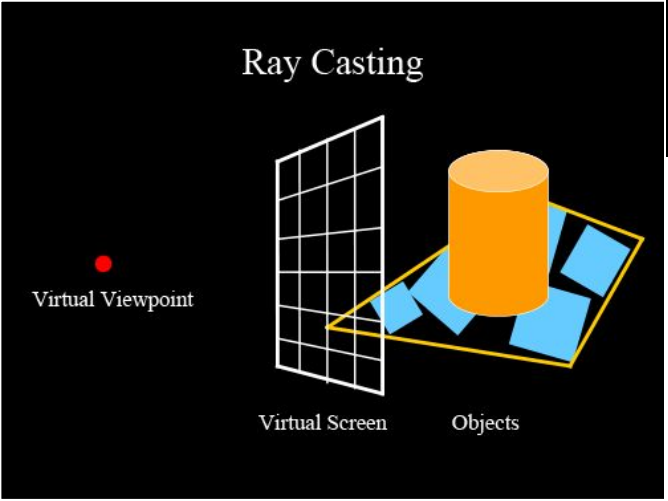
Raytraced Graphics

(Non-interactive, the stuff they generally use in the movies)

Note: Ray Tracing and path tracing are terms often used interchangeably, but they are technically different based on the where the origin of the ray is coming from.

What is a raytracer?

- As a quick introduction, a raytracer is where we ‘cast’ a ray from some location and see if it intersects with another object.
- Typically we do this (at least) once per-pixel
  - Rays may also bounce multiple time (to create reflections and shadows)
- You can otherwise see an example of a raytracer progressively building the scene for each scanline on the bottom-right

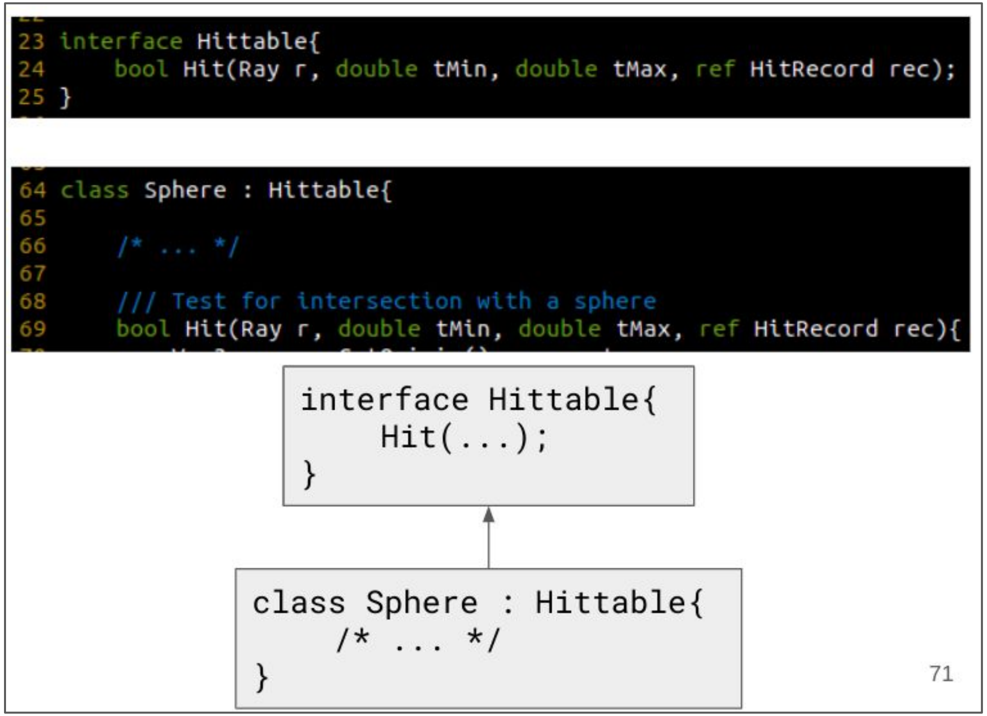


Ray Tracing - Analogy

- The analogy is exactly like pointing a laser pointer
  - Our laser pointer hits the closest surface that it hits against

Interfaces in D for ‘ray intersection’

- Dlang supports interfaces, which allow us to derive a class from common interface, where we must implement the member functions of the interface.
    - abstract classes and regular classes also exist.
        - abstract classes are similar to interfaces, but allow member variables.
- Interfaces provides a nice ‘contract’ when implementing some hittable surface in a raytracer



```
interface Hittable{
    bool Hit(Ray r, double tMin, double tMax, ref HitRecord rec);
}
```

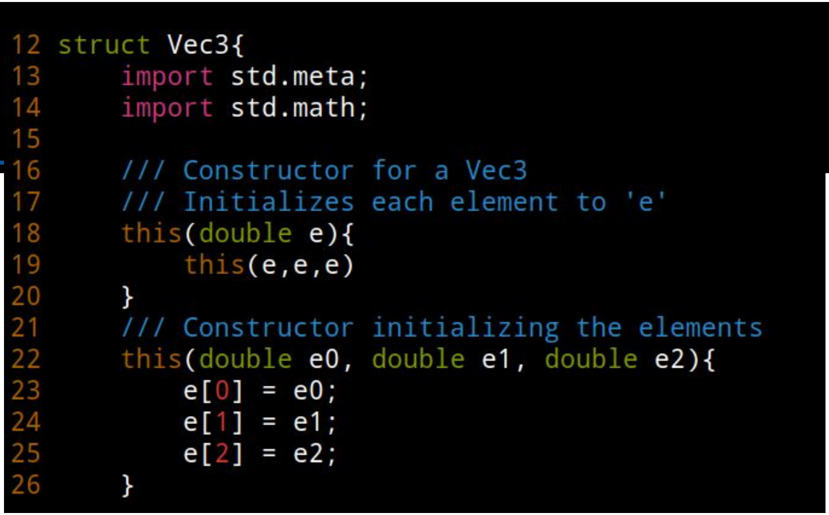
```
class Sphere : Hittable{

    /* ... */

    /// Test for intersection with a sphere
    bool Hit(Ray r, double tMin, double tMax, ref HitRecord rec){
```

```
interface Hittable{
    Hit(...);
}
```

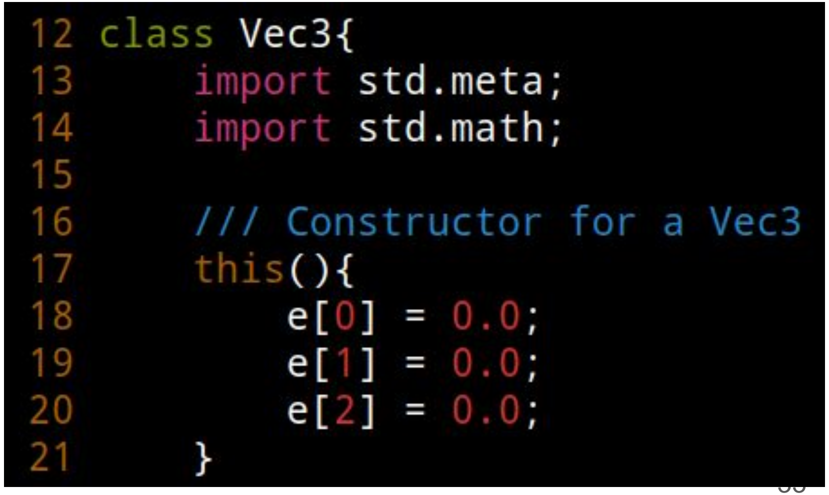
```
class Sphere : Hittable{
    /* ... */
}
```

D class versus struct

- In D **class** and **struct** represent reference and value types
  - Classes are (by default) heap allocated
  - Classes allow for polymorphism
  - Structs are (by default) stack allocated
  - No default constructor for ‘struct’
- I *like* that these keywords have different meaning in the design of my programs.



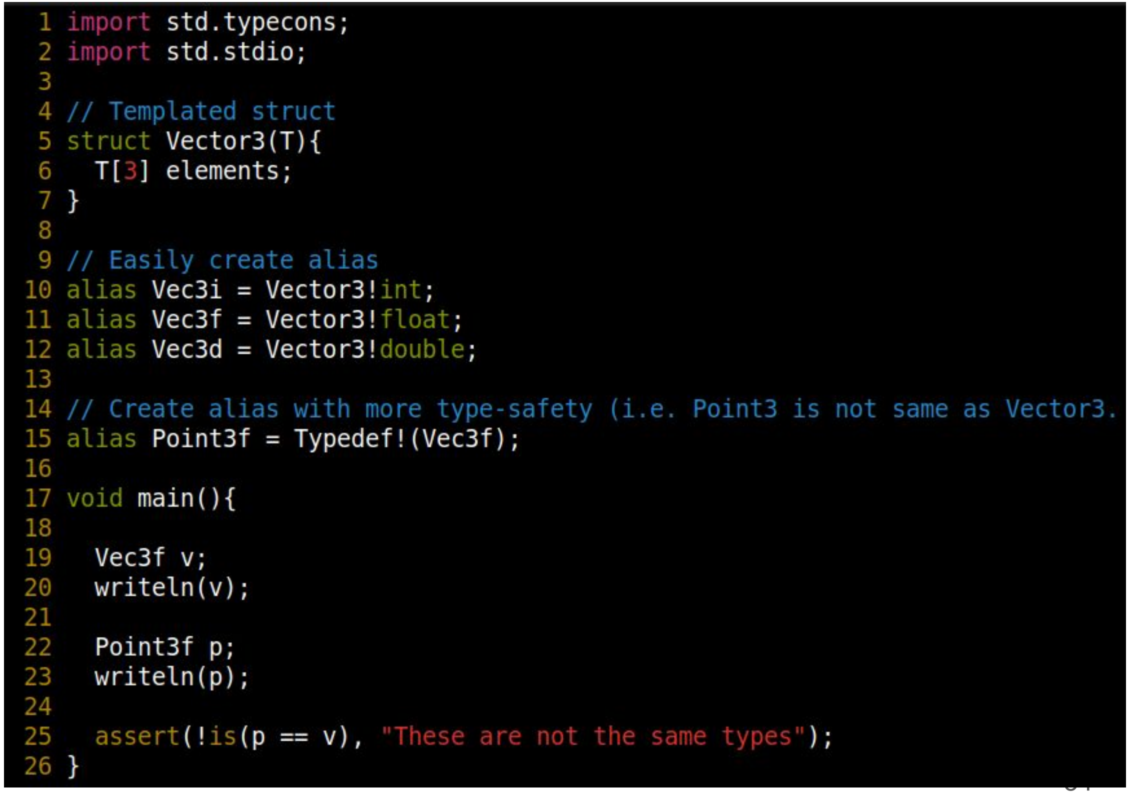
```d
struct Vec3{
      import std.meta;
      import std.math;

      /// Constructor for a Vec3
      /// Initializes each element to 'e'
      this(double e){
            this(e,e,e)
      }
      /// Constructor initializing the elements
      this(double e0, double e1, double e2){
            e[0] = e0;
            e[1] = e1;
            e[2] = e2;
      }
```



```d
class Vec3{
      import std.meta;
      import std.math;

      /// Constructor for a Vec3
      this(){
            e[0] = 0.0;
            e[1] = 0.0;
            e[2] = 0.0;
      }
```

Clean Template Syntax (my opinion)

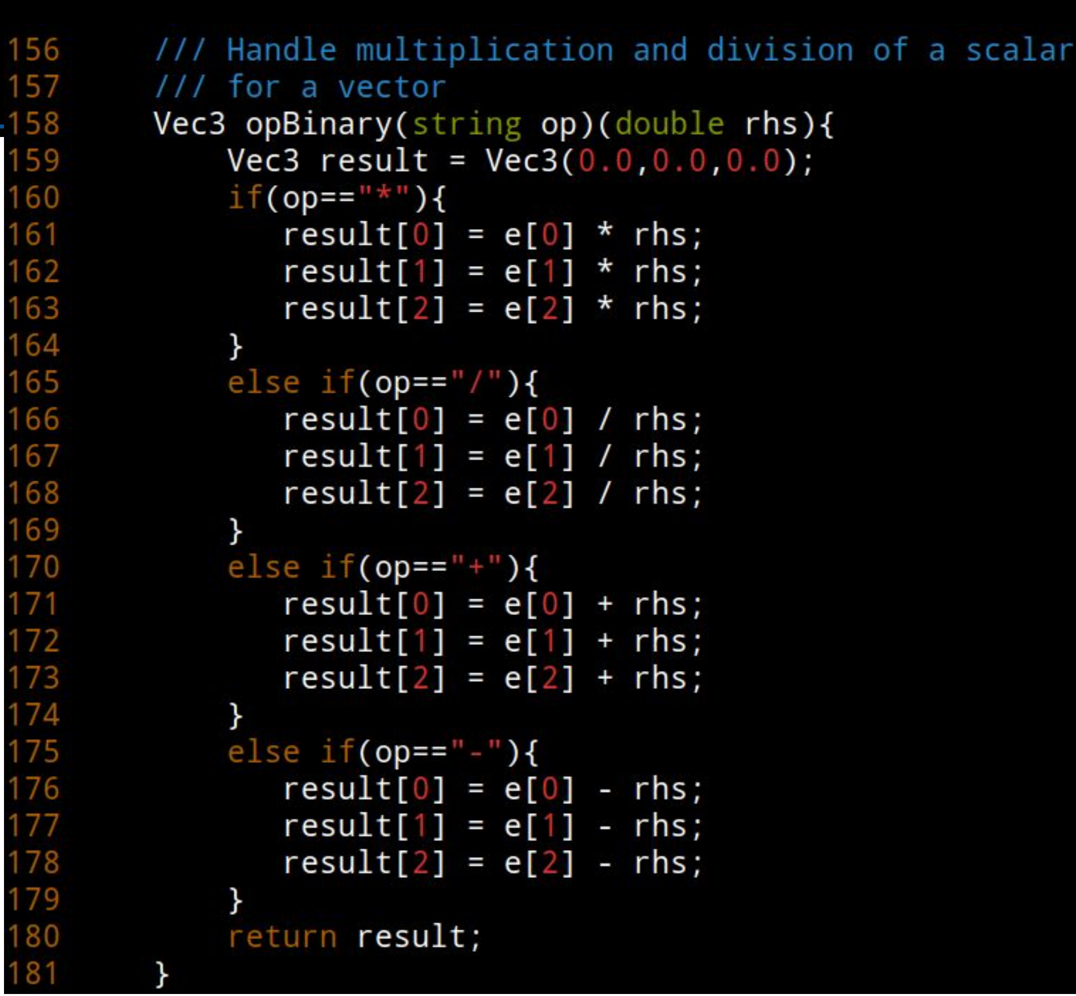
- Creating templated types are a breeze in D for any struct, class, function, or type we create
- Simply use parentheses (line 5) where the type is
    - Lines 10-12 demonstrate with '!' the type
    - Using the 'alias' keyword at global or local scopes gives us another name.
    - Line 15 adds further power if we want some 'semantic' meaning of a Point being different than a Vector
        - (Even though the data is the same)



```
 1 import std.typecons;
 2 import std.stdio;
 3
 4 // Templated struct
 5 struct Vector3(T){
 6   T[3] elements;
 7 }
 8
 9 // Easily create alias
10 alias Vec3i = Vector3!int;
11 alias Vec3f = Vector3!float;
12 alias Vec3d = Vector3!double;
13
14 // Create alias with more type-safety (i.e. Point3 is not same as Vector3.
15 alias Point3f = Typedef!(Vec3f);
16
17 void main(){
18
19   Vec3f v;
20   writeln(v);
21
22   Point3f p;
23   writeln(p);
24
25   assert(!is(p == v), "These are not the same types");
26 }
```

Operator Overload

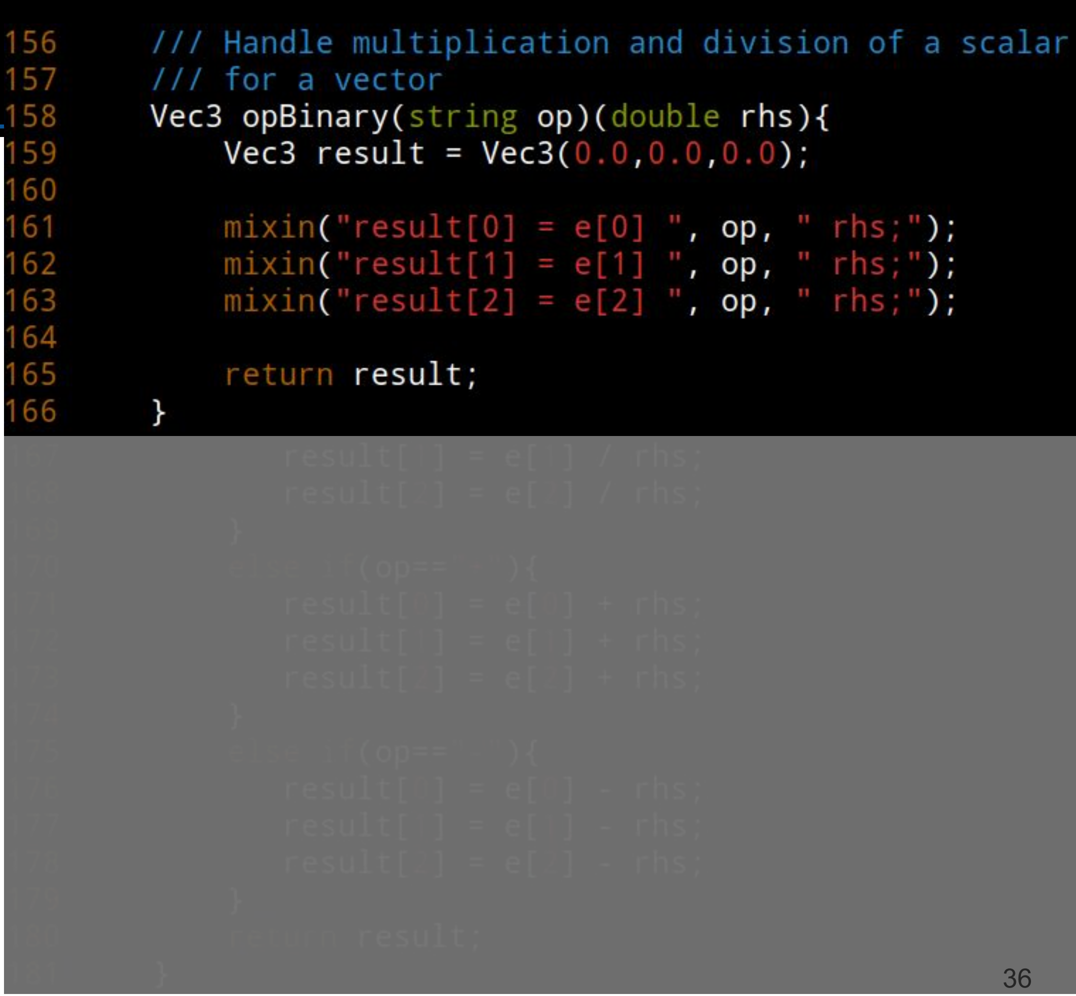
- D allows **operator overloading** for member functions
    - e.g. ‘opBinary’ for binary operations involving operations with the type on the left, and type on the right



```
/// Handle multiplication and division of a scalar
/// for a vector
Vec3 opBinary(string op)(double rhs){
    Vec3 result = Vec3(0.0,0.0,0.0);
    if(op=="*"){
       result[0] = e[0] * rhs;
       result[1] = e[1] * rhs;
       result[2] = e[2] * rhs;
    }
    else if(op=="/"){
       result[0] = e[0] / rhs;
       result[1] = e[1] / rhs;
       result[2] = e[2] / rhs;
    }
    else if(op=="+"){
       result[0] = e[0] + rhs;
       result[1] = e[1] + rhs;
       result[2] = e[2] + rhs;
    }
    else if(op=="-"){
       result[0] = e[0] - rhs;
       result[1] = e[1] - rhs;
       result[2] = e[2] - rhs;
    }
    return result;
}
```

A better Overload

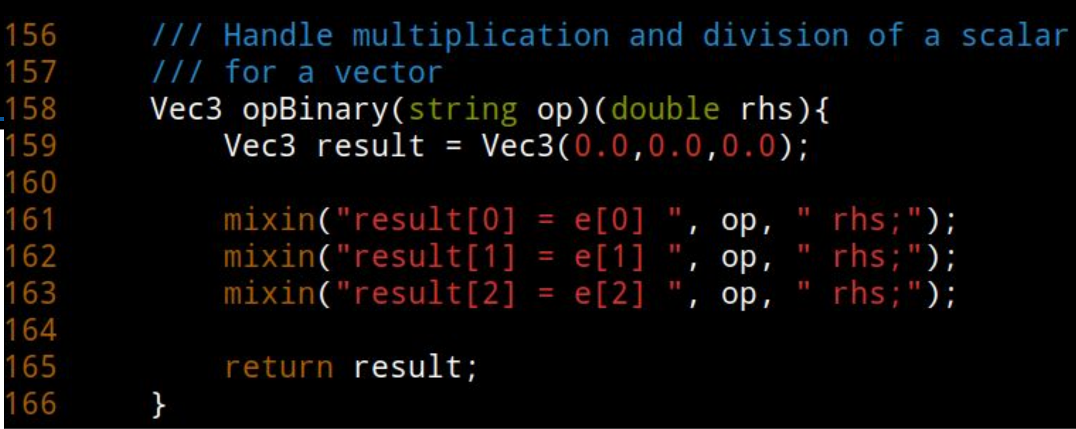
- Using D’s **mixin** feature, the equivalent code can be generated at compile-time for each template.
    - The ‘string op’ is already the template parameter for the operating being used.
    - So instead of having to compare, simply use the mixin.
    - No comparisons, no branches used, only generate code needed (e.g. + or -), and otherwise future-proof your code if you add other operators.



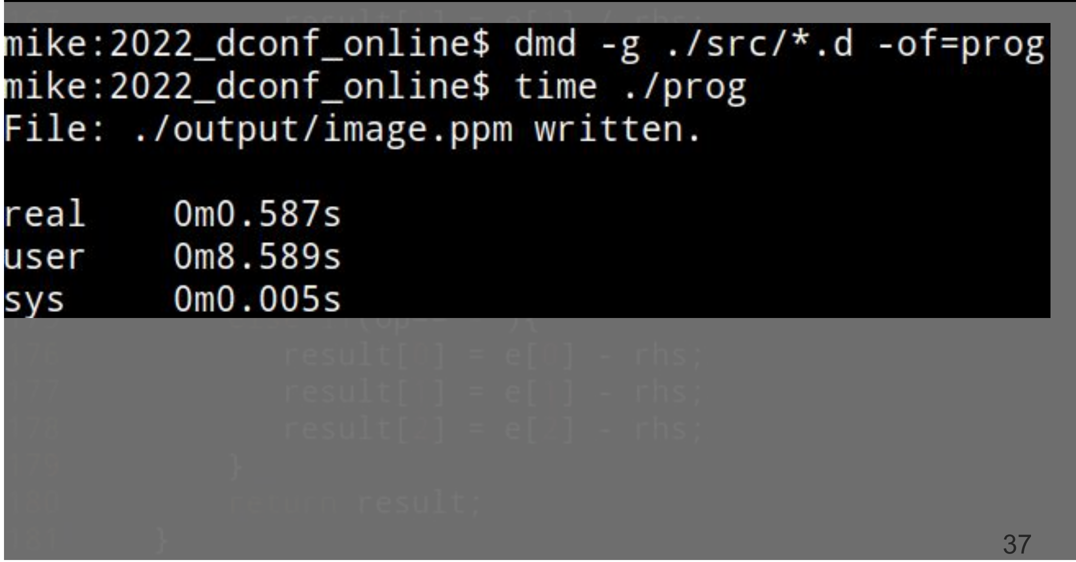
```
/// Handle multiplication and division of a scalar
/// for a vector
Vec3 opBinary(string op)(double rhs){
    Vec3 result = Vec3(0.0,0.0,0.0);

    mixin("result[0] = e[0] ", op, " rhs;");
    mixin("result[1] = e[1] ", op, " rhs;");
    mixin("result[2] = e[2] ", op, " rhs;");

    return result;
}
```

Templates for the win!

- Avoiding branches in this particular case sped up my raytracer
    - (From 0.769 seconds to 0.587seconds)
- The code features fewer branches, is easier to understand, supports more operators, and is arguably easier to read.



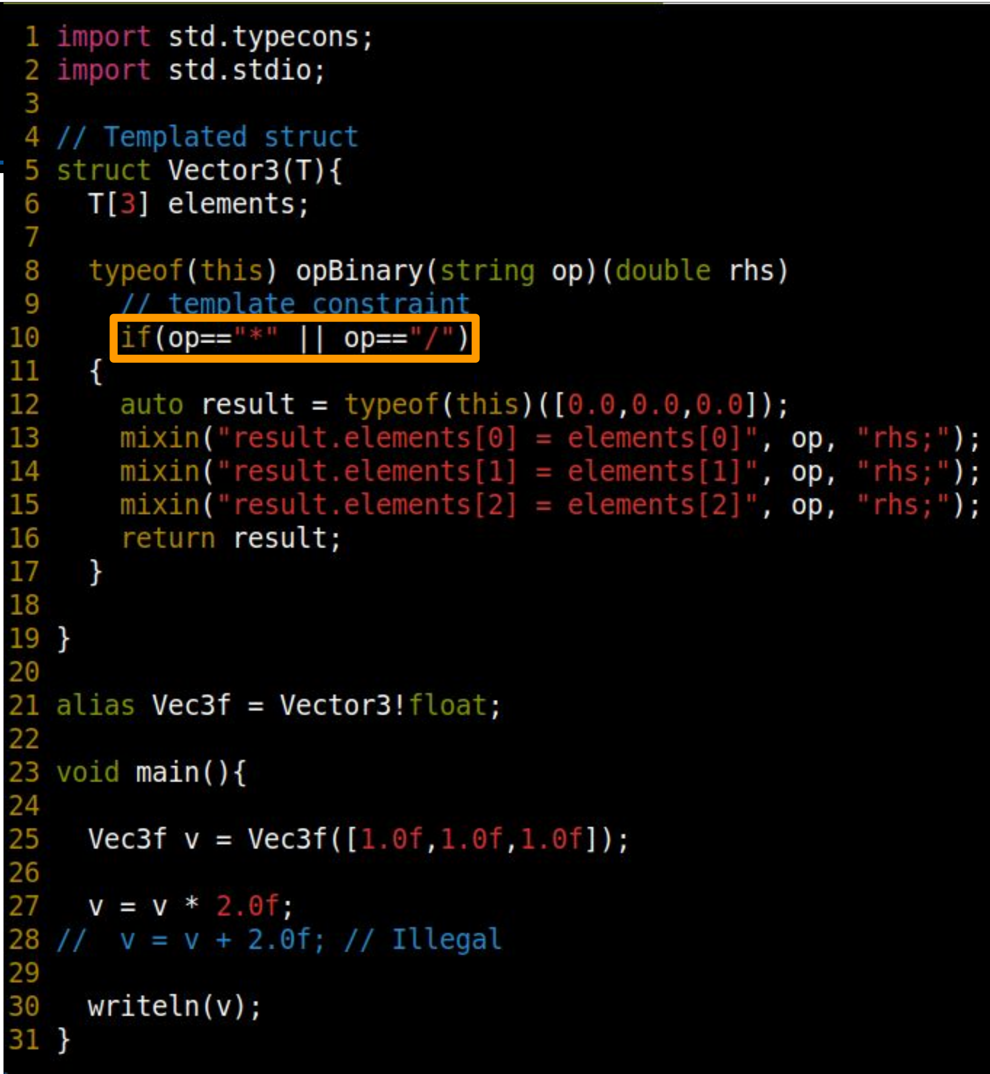
```
    /// Handle multiplication and division of a scalar
    /// for a vector
    Vec3 opBinary(string op)(double rhs){
        Vec3 result = Vec3(0.0,0.0,0.0);

        mixin("result[0] = e[0] ", op, " rhs;");
        mixin("result[1] = e[1] ", op, " rhs;");
        mixin("result[2] = e[2] ", op, " rhs;");

        return result;
    }
```



```
mike:2022_dconf_online$ dmd -g ./src/*.d -of=prog
mike:2022_dconf_online$ time ./prog
File: ./output/image.ppm written.

real	0m0.587s
user	0m8.589s
sys	0m0.005s
```

Template Constraints

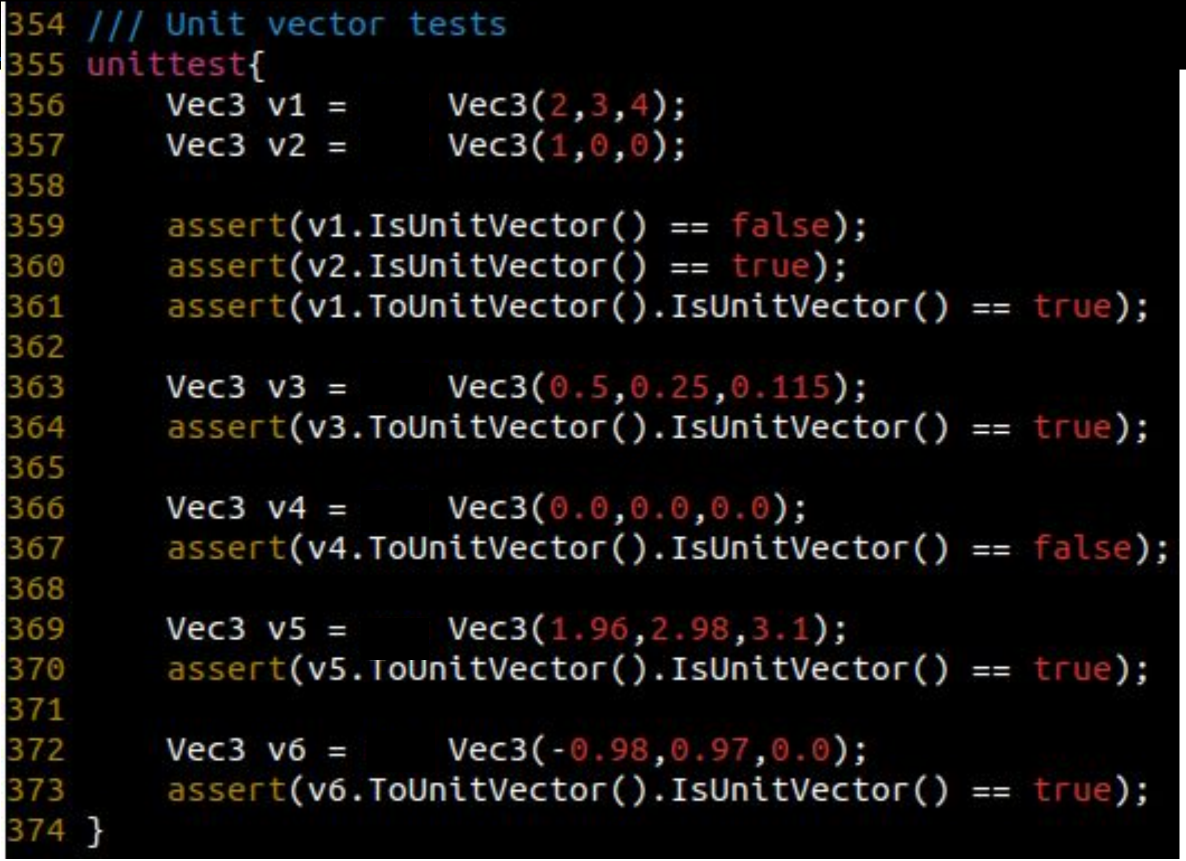
- Note:
  - Maybe we don't want to allow 'any' operator with opBinary
  - Observe line '10' we can add template constraints to otherwise check which symbols are allowed.
- Other features shown:
  - line 12:
    - 'auto' for deducing the type and generic programming
    - 'typeof(this)' for placing the templated type instance.
  - Note:
    - For structs, constructors are automatically created for us if one is not defined.



```
 1 import std.typecons;
 2 import std.stdio;
 3
 4 // Templated struct
 5 struct Vector3(T){
 6   T[3] elements;
 7
 8   typeof(this) opBinary(string op)(double rhs)
 9     // template constraint
10     if(op=="*" || op=="/")
11   {
12     auto result = typeof(this)([0.0,0.0,0.0]);
13     mixin("result.elements[0] = elements[0]", op, "rhs;");
14     mixin("result.elements[1] = elements[1]", op, "rhs;");
15     mixin("result.elements[2] = elements[2]", op, "rhs;");
16     return result;
17   }
18
19 }
20
21 alias Vec3f = Vector3!float;
22
23 void main(){
24
25   Vec3f v = Vec3f([1.0f,1.0f,1.0f]);
26
27   v = v * 2.0f;
28 //  v = v + 2.0f; // Illegal
29
30   writeln(v);
31 }
```

Vec3 and Unit Test

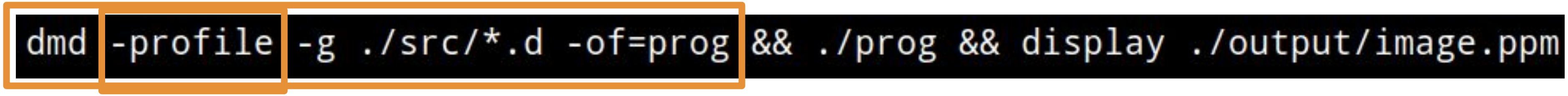
- D has built-in ‘unittest’ blocks to otherwise increase my confidence in the correctness of my code.
- Here is another example of a Vec3 type



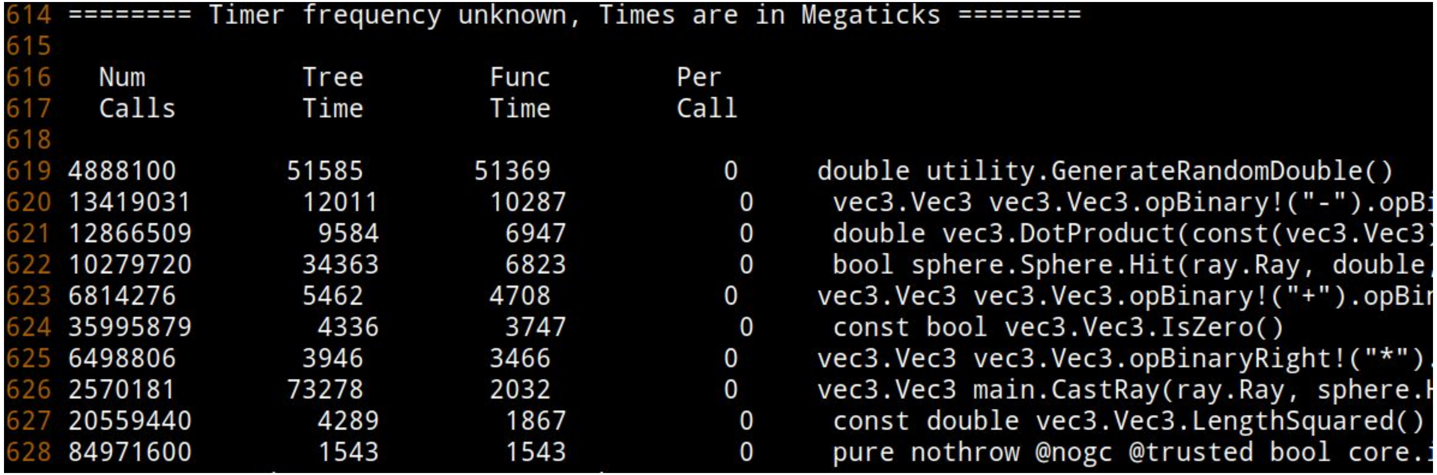
```
/// Unit vector tests
unittest{
    Vec3 v1 =     Vec3(2,3,4);
    Vec3 v2 =     Vec3(1,0,0);

    assert(v1.IsUnitVector() == false);
    assert(v2.IsUnitVector() == true);
    assert(v1.ToUnitVector().IsUnitVector() == true);

    Vec3 v3 =     Vec3(0.5,0.25,0.115);
    assert(v3.ToUnitVector().IsUnitVector() == true);

    Vec3 v4 =     Vec3(0.0,0.0,0.0);
    assert(v4.ToUnitVector().IsUnitVector() == false);

    Vec3 v5 =     Vec3(1.96,2.98,3.1);
    assert(v5.ToUnitVector().IsUnitVector() == true);

    Vec3 v6 =     Vec3(-0.98,0.97,0.0);
    assert(v6.ToUnitVector().IsUnitVector() == true);
}
```

-profile [switches see -profile]

- Note: D has a built in profiler, garbage collection profiler, and code coverage tools that just make it feel complete!
  - These are great instrumentation tools to help you understand your performance!

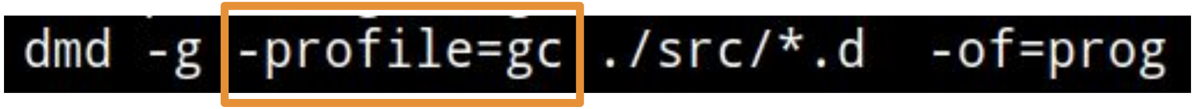


```
dmd -profile -g ./src/*.d -of=prog && ./prog && display ./output/image.ppm
```



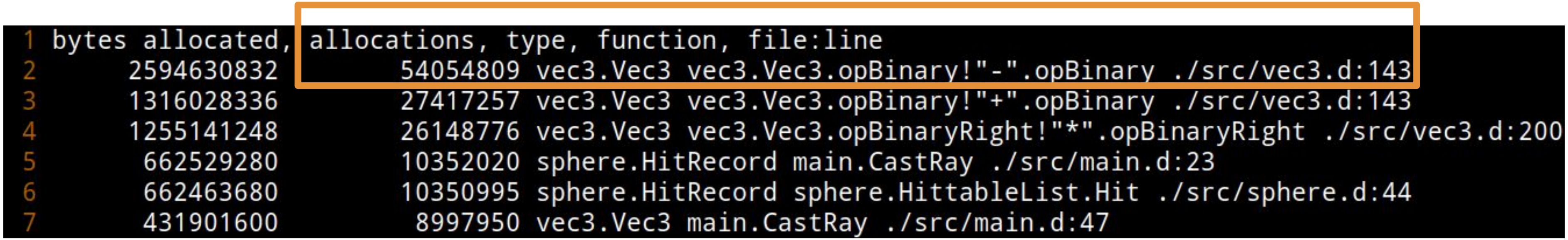
```
======== Timer frequency unknown, Times are in Megaticks ========

  Num          Tree        Func        Per
  Calls        Time        Time        Call

4888100       51585       51369          0     double utility.GenerateRandomDouble()
13419031       12011       10287          0     vec3.Vec3 vec3.Vec3.opBinary!("-").opBi
12866509        9584        6947          0     double vec3.DotProduct(const(vec3.Vec3)
10279720       34363        6823          0     bool sphere.Sphere.Hit(ray.Ray, double,
6814276        5462        4708          0     vec3.Vec3 vec3.Vec3.opBinary!("+").opBir
35995879        4336        3747          0     const bool vec3.Vec3.IsZero()
6498806        3946        3466          0     vec3.Vec3 vec3.Vec3.opBinaryRight!("*").
2570181       73278        2032          0     vec3.Vec3 main.CastRay(ray.Ray, sphere.H
20559440        4289        1867          0     const double vec3.Vec3.LengthSquared()
84971600        1543        1543          0     pure nothrow @nogc @trusted bool core.i
```

-profile=gc



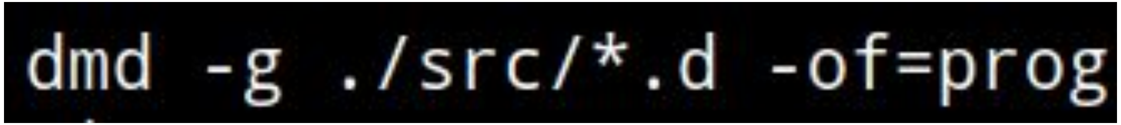
```
dmd -g -profile=gc ./src/*.d  -of=prog
```

- Using D’s profiler we can see how many heap allocations took place, and it turns out at some point I was doing many with Vec3!



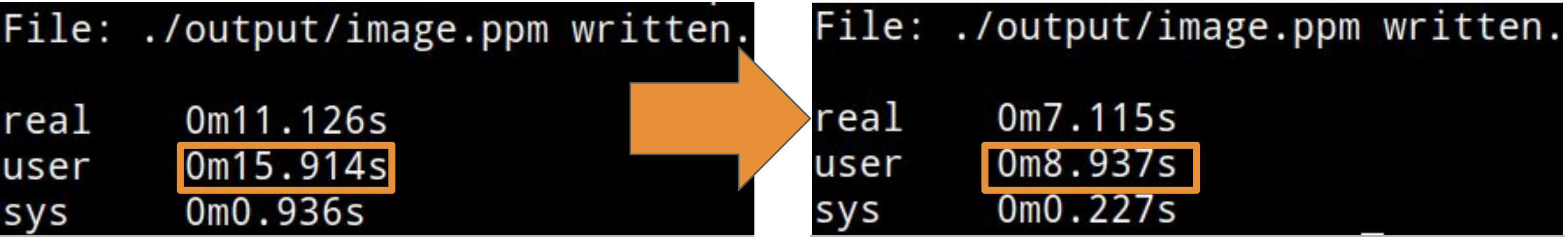
```
bytes allocated, allocations, type, function, file:line
     2594630832        54054809 vec3.Vec3 vec3.Vec3.opBinary!"-".opBinary ./src/vec3.d:143
     1316028336        27417257 vec3.Vec3 vec3.Vec3.opBinary!"+".opBinary ./src/vec3.d:143
     1255141248        26148776 vec3.Vec3 vec3.Vec3.opBinaryRight!"*".opBinaryRight ./src/vec3.d:200
      662529280        10352020 sphere.HitRecord main.CastRay ./src/main.d:23
      662463680        10350995 sphere.HitRecord sphere.HittableList.Hit ./src/sphere.d:44
      431901600         8997950 vec3.Vec3 main.CastRay ./src/main.d:47
```

-profile=gc (After making a Vec3 a struct)



```
dmd -g ./src/*.d -of=prog
```

- Rerunning again (this time, no profile collected)
- We’re again, about twice as fast again!



```
File: ./output/image.ppm written.

real	0m11.126s
user	0m15.914s
sys	0m0.936s
```

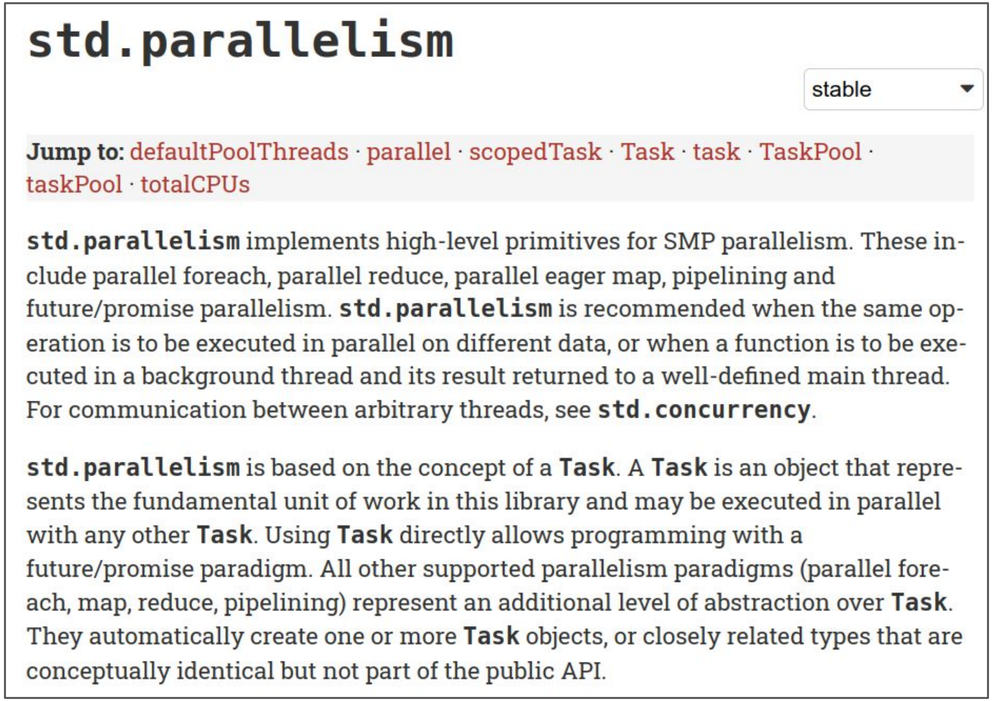
Before

```
File: ./output/image.ppm written.

real	0m7.115s
user	0m8.937s
sys	0m0.227s
```

After

std.parallelism [docs]

- D offers several forms of concurrency as well as parallelism.
- For our ray tracer, we truly want parallelism, as we are able to cast rays in an order independent task of casting rays
    - (i.e. We cast ~1 ray per pixel in our screen, and we write to one location in memory at a time.)



std.parallelism

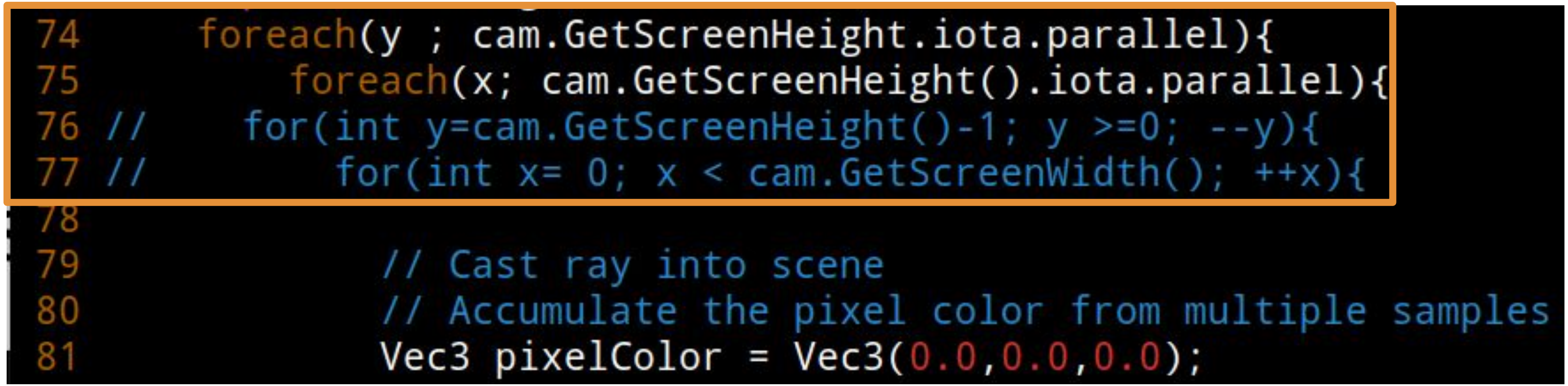
Jump to: defaultPoolThreads · parallel · scopedTask · Task · task · TaskPool · taskPool · totalCPUs

`std.parallelism` implements high-level primitives for SMP parallelism. These include parallel foreach, parallel reduce, parallel eager map, pipelining and future/promise parallelism. **`std.parallelism`** is recommended when the same operation is to be executed in parallel on different data, or when a function is to be executed in a background thread and its result returned to a well-defined main thread. For communication between arbitrary threads, see **`std.concurrency`**.

`std.parallelism` is based on the concept of a **`Task`**. A **`Task`** is an object that represents the fundamental unit of work in this library and may be executed in parallel with any other **`Task`**. Using **`Task`** directly allows programming with a future/promise paradigm. All other supported parallelism paradigms (parallel foreach, map, reduce, pipelining) represent an additional level of abstraction over **`Task`**. They automatically create one or more **`Task`** objects, or closely related types that are conceptually identical but not part of the public API.

For-loop to parallel task

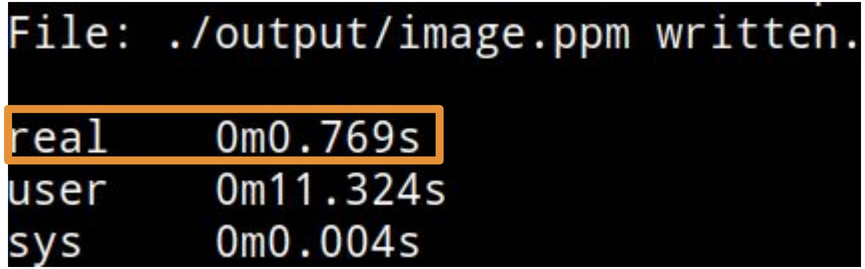
- Highlighted below is the conversion from a serial O(n²) loop, to a parallel computation using Tasks built in Dlang.
    - Note: iota gives us the range of values that we are going to iterate on in parallel.
    - Note: See Ali's Dconf 22 talk for a guide to iota: https://www.youtube.com/watch?v=gwUcngTmKhg



```
    foreach(y ; cam.GetScreenHeight.iota.parallel){
        foreach(x; cam.GetScreenHeight().iota.parallel){
//    for(int y=cam.GetScreenHeight()-1; y >=0; --y){
//        for(int x= 0; x < cam.GetScreenWidth(); ++x){

            // Cast ray into scene
            // Accumulate the pixel color from multiple samples
            Vec3 pixelColor = Vec3(0.0,0.0,0.0);
```

real time (versus user time)

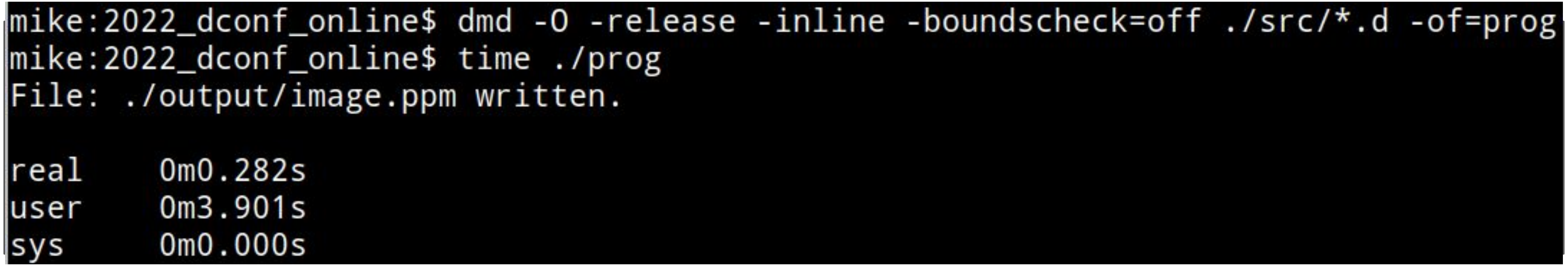
- Measuring the time now, we need to somewhat rely on the ‘real’ time when running parallel threads.
  - ‘user’ time represents the total cpu time -- and that’s a sum of all of the cpus running in parallel.
  - Before converting to parallel, we have now gone from 5.9 seconds to less than a second by adding ‘.parallel’ in our loops to spawn threads automatically



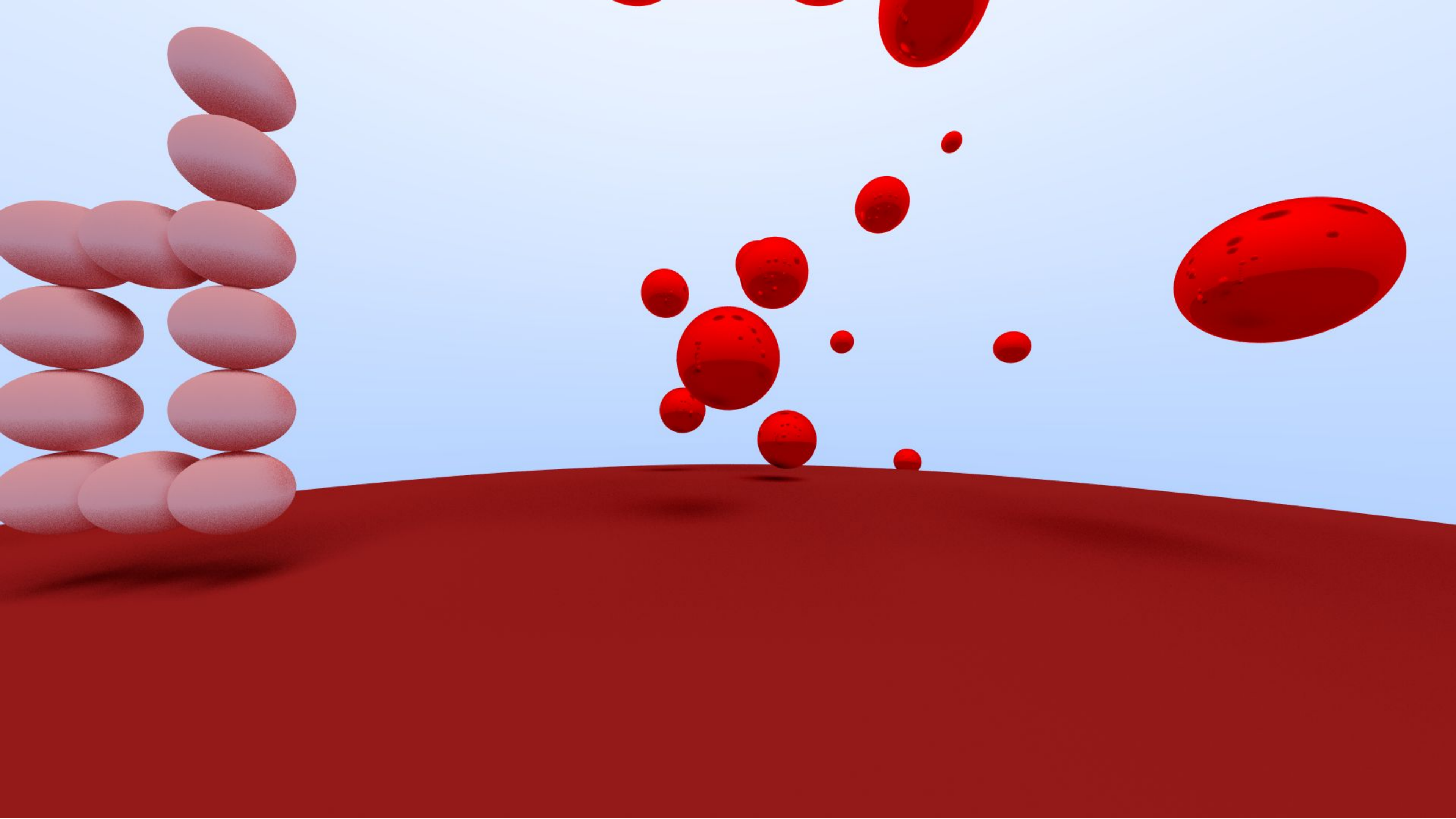
```
File: ./output/image.ppm written.

real	0m0.769s
user	0m11.324s
sys	0m0.004s
```

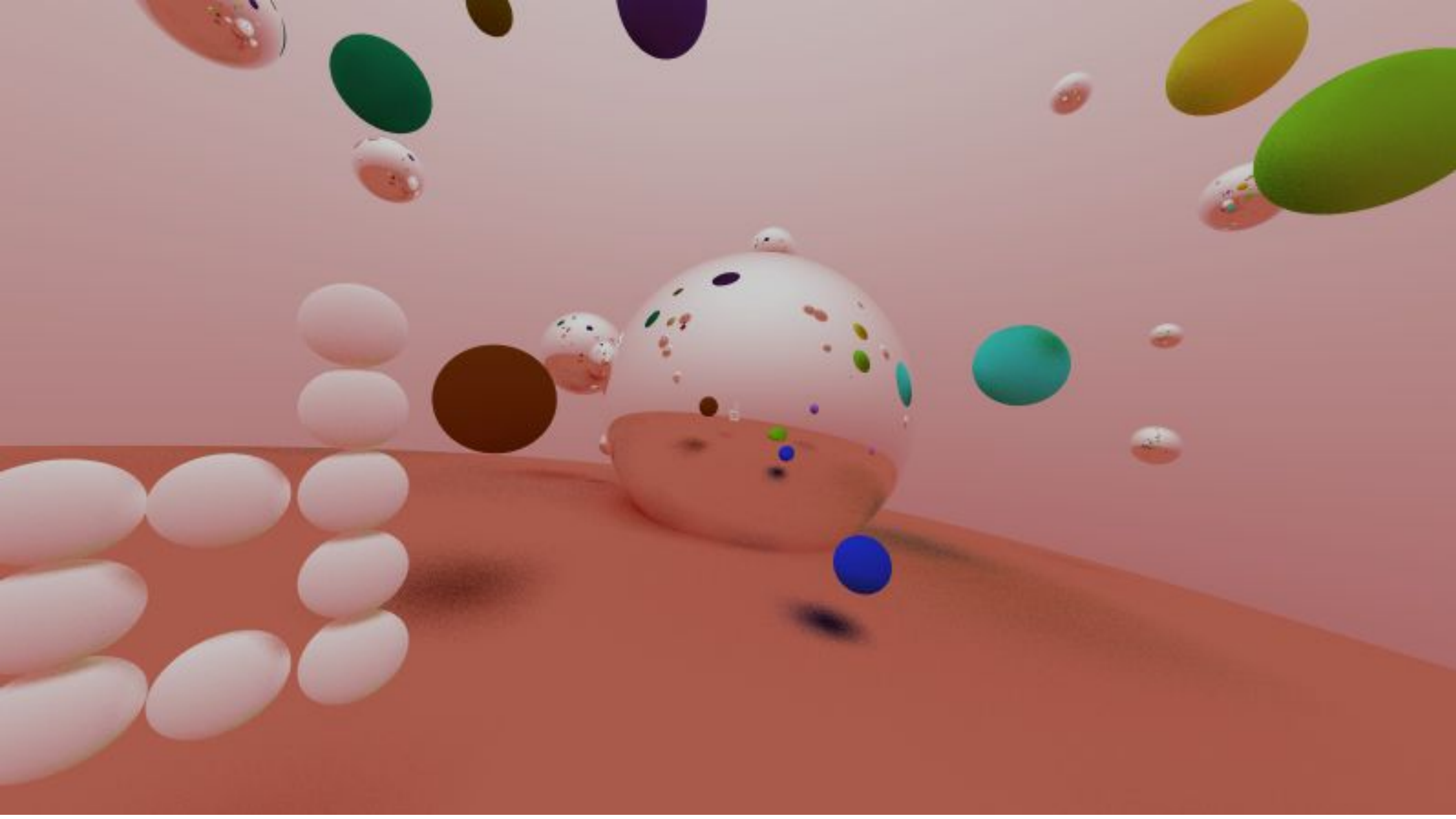
Release Build

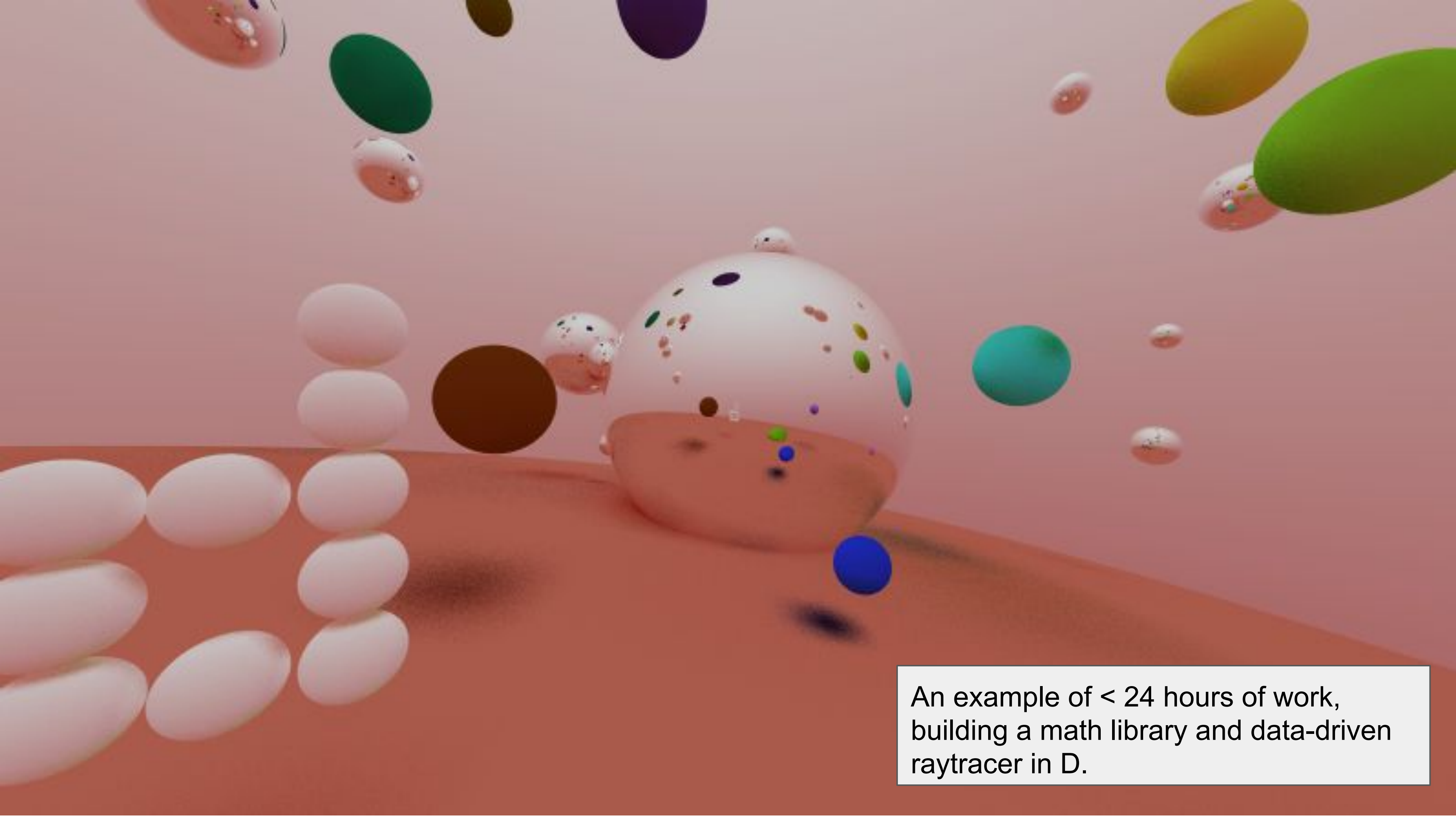
- D by default offers safety (e.g. default initialized values, bounds checking on arrays, thread local variables, and more!), but we can toggle some of those options on and off as needed
    - Note: There are additional memory safety annotations (@safe, @trusted, @system) that I will not cover during this talk.
- Toggling the compiler flags from https://dlang.org/dmd-linux.html we can do a release build for more performance with DMD
    - (And using GDC or LDC compiler backends provides even faster executables.)



```
mike:2022_dconf_online$ dmd -O -release -inline -boundscheck=off ./src/*.d -of=prog
mike:2022_dconf_online$ time ./prog
File: ./output/image.ppm written.

real    0m0.282s
user    0m3.901s
sys     0m0.000s
```



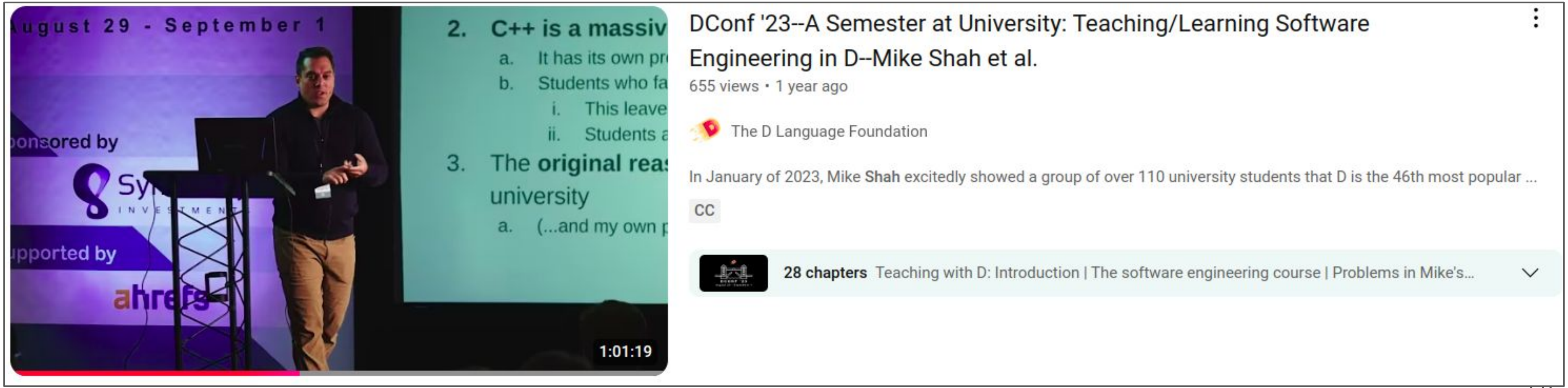




An example of < 24 hours of work, building a math library and data-driven raytracer in D.

My Case for D

- So at this point, I was pretty encouraged by D, enough that I decided I would start teaching D (at Northeastern University and now Yale University) in Spring of 2023 in Software Engineering to start
  - (If you watch the second half of the talk -- you also hear directly from the students their unfiltered thoughts on using and learning D in the course)



DConf '23--A Semester at University: Teaching/Learning Software Engineering in D--Mike Shah et al.

655 views • 1 year ago

The D Language Foundation

In January of 2023, Mike Shah excitedly showed a group of over 110 university students that D is the 46th most popular ...

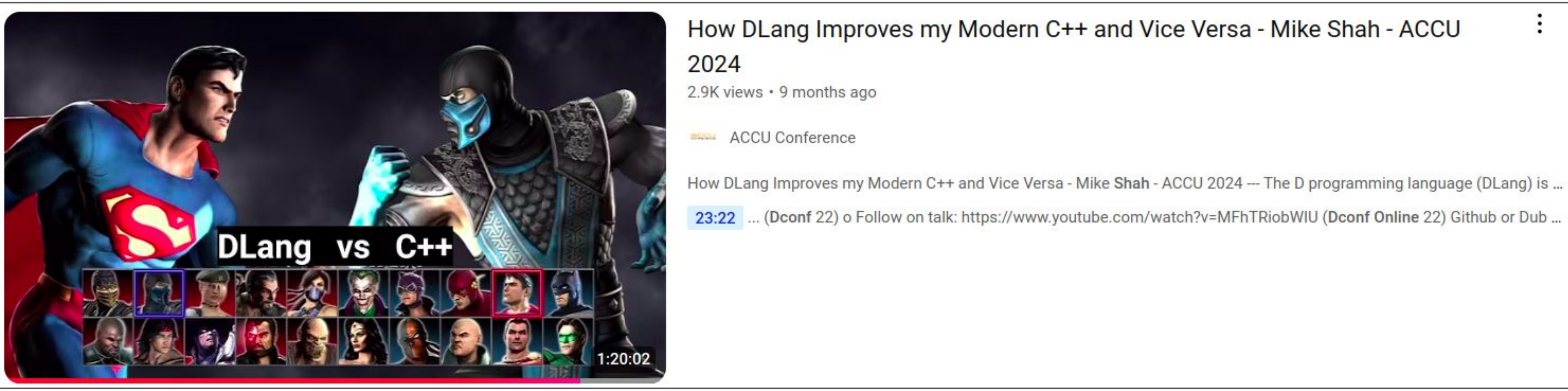
CC

28 chapters Teaching with D: Introduction | The software engineering course | Problems in Mike's...

https://www.youtube.com/watch?v=V2YwTIIMEeU

C++ and DLang as complementary languages

- At ACCU last year, I found that writing D code improved my C++ knowledge quite a bit as well.
- One key thing was that my ‘D’ programming had a much faster iteration time -- so I wanted to take on the challenge of real-time graphics programming in D next



How DLang Improves my Modern C++ and Vice Versa - Mike Shah - ACCU 2024

2.9K views • 9 months ago

ACCU Conference

How DLang Improves my Modern C++ and Vice Versa - Mike **Shah** - ACCU 2024 — The D programming language (DLang) is ...

23:22 ... (**Dconf** 22) o Follow on talk: https://www.youtube.com/watch?v=MFhTRiobWIU (**Dconf Online** 22) Github or Dub ...

https://www.youtube.com/watch?v=CnKsOak0DHU (Note: The talk is **not** about Dlang vs C++ as the thumbnail suggests!)

Real-Time Graphics Programming in D

(For things like games and simulation)

What is needed for real-time graphics programming? (1/2)

Generally speaking:

1. A systems programming language (is most commonly used) for graphics programming
   a. Many graphics APIs (OpenGL, Vulkan, etc.) are C-based APIs
   b. D talks with C very easily (See the interfacing guide), and it is often merely a matter of using a binding to expose the C library functions to a programmer.
      i. D also provides a way to transition C code (https://dlang.org/spec/importc.html) to D code (C++ and Obj-C are also works in progress)
      ii. See some of the example guides here: https://dlang.org/articles/ctod.html
2. We need a math library, or otherwise the ability to make a good math library
   a. D itself provides operating overloading as we have previously seen to make this convenient.

What is needed for real-time graphics programming? (2/2)

But first -- as some inspiration, I wondered if anyone using D for serious real-time graphics work (the answer was of course yes).

It’s sort of a confidence booster to see someone else has used a tool to build something -- so here are some examples

Generally speaking:

1. A s … or gra …
   a.
   b. … y a … er. … lso … .html
2. We need a math library, or otherwise the ability to make a good math library
   a. D itself provides operating overloading as we have previously seen to make this convenient.

D Graphics Projects

(More projects found at my FOSDEM 2024 talk here: https://www.youtube.com/watch?v=yLaUsmLr9so)

First Look at:
Dlang
"Starting program".writeln;
enum a = [3, 1, 2, 4, 0];
// Sort data at compile-time
static immutable b = sort(a);
FOSDEM'24
pragma(msg, "Finished com
53:58

[Programming Languages] Episode 19 - First Impression - dlang (FOSDEM 2024 Talk)

673 views • 3 weeks ago

Mike Shah

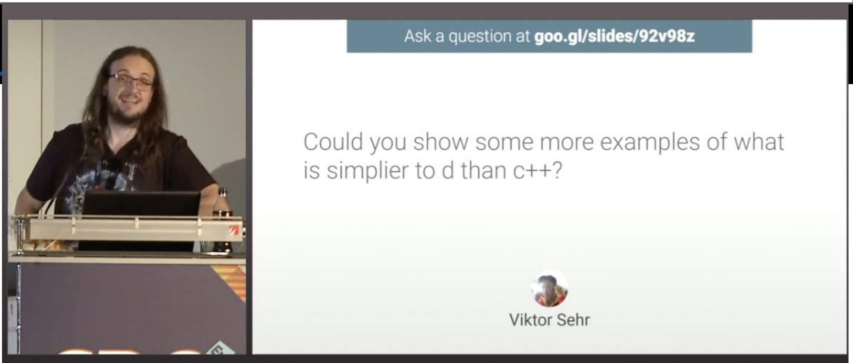
⊳Lesson Description: In this lesson I present one of my favorite languages -- in fact I'm breaking the rules a bit -- dlang! As many ...

Utilized the D Programming Language

Quantum Break -- Game

AAA Game Projects in D

- It’s also worth noting that D has been used in AAA Commercial Games
  - Ethan Watson has a wonderful presentation describing that experience
  - Link to talk: https://www.gdcvault.com/play/1023843/D-Using-an-Emerging-Language
- Talk Abstract: *Can you use D to make games? Yes. Has it been used in a major release? It has now. But what benefits does it have over C++? Is it ready for mass use? Does treating code as data with a traditional C++ engine work? This talk will cover Remedy's usage of the D programming language in Quantum Break and also provide some details on where we want to take usage of it in the future.*



Ask a question at goo.gl/slides/92v98z

Could you show some more examples of what is simplier to d than c++?

Viktor Sehr

-20:20 1x

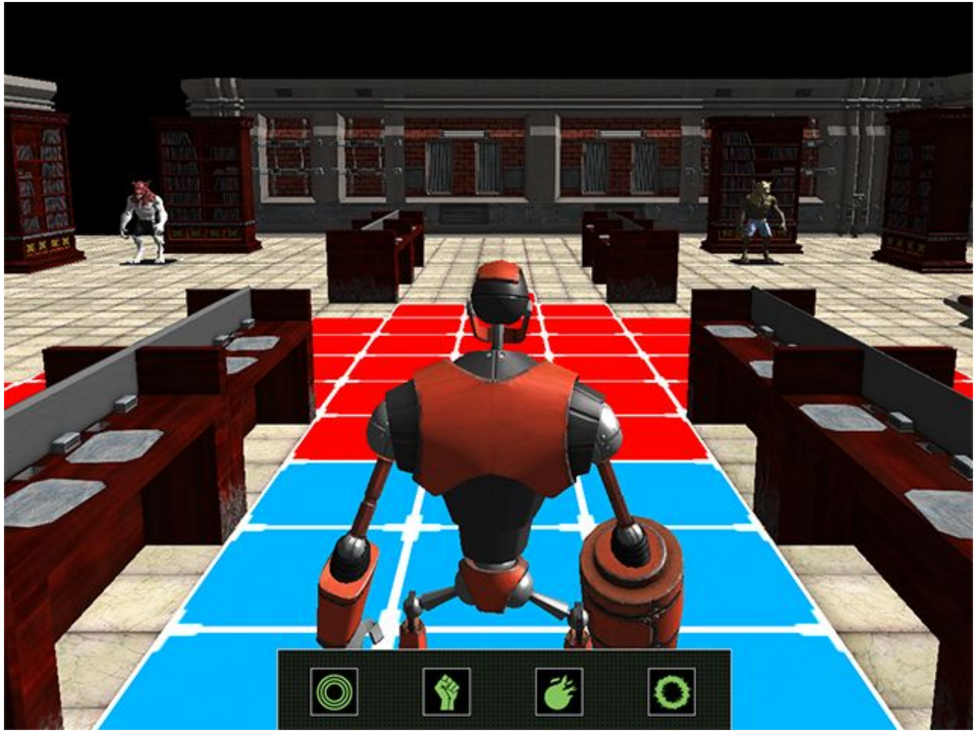
https://m.media-amazon.com/images/M/MV5BOThjOWRhN2QtYmIxMy00MGE3LTk5ZWMtY2ZkMzI0MGY1ZTM1XkEyXkFqcGdeQXVyMTYxMzY1ODg@._V1_.jpg

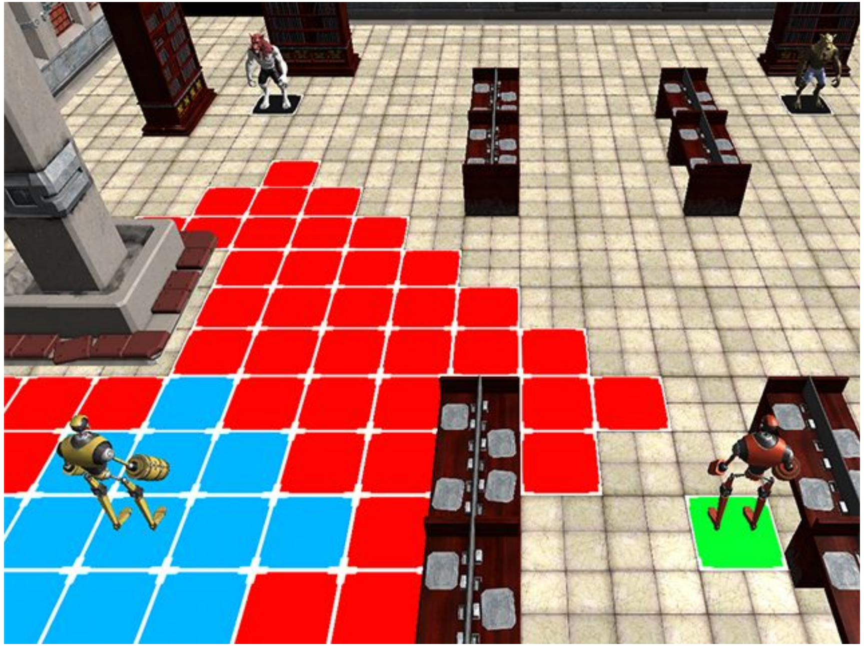
Dagon -- Game Engine

- Website with games and tutorials: https://gecko0307.github.io/dagon/
- Github or Dub Repository: https://github.com/gecko0307/dagon | https://code.dlang.org/packages/dagon

Built in the D Programming Language

Dash -- Game Engine

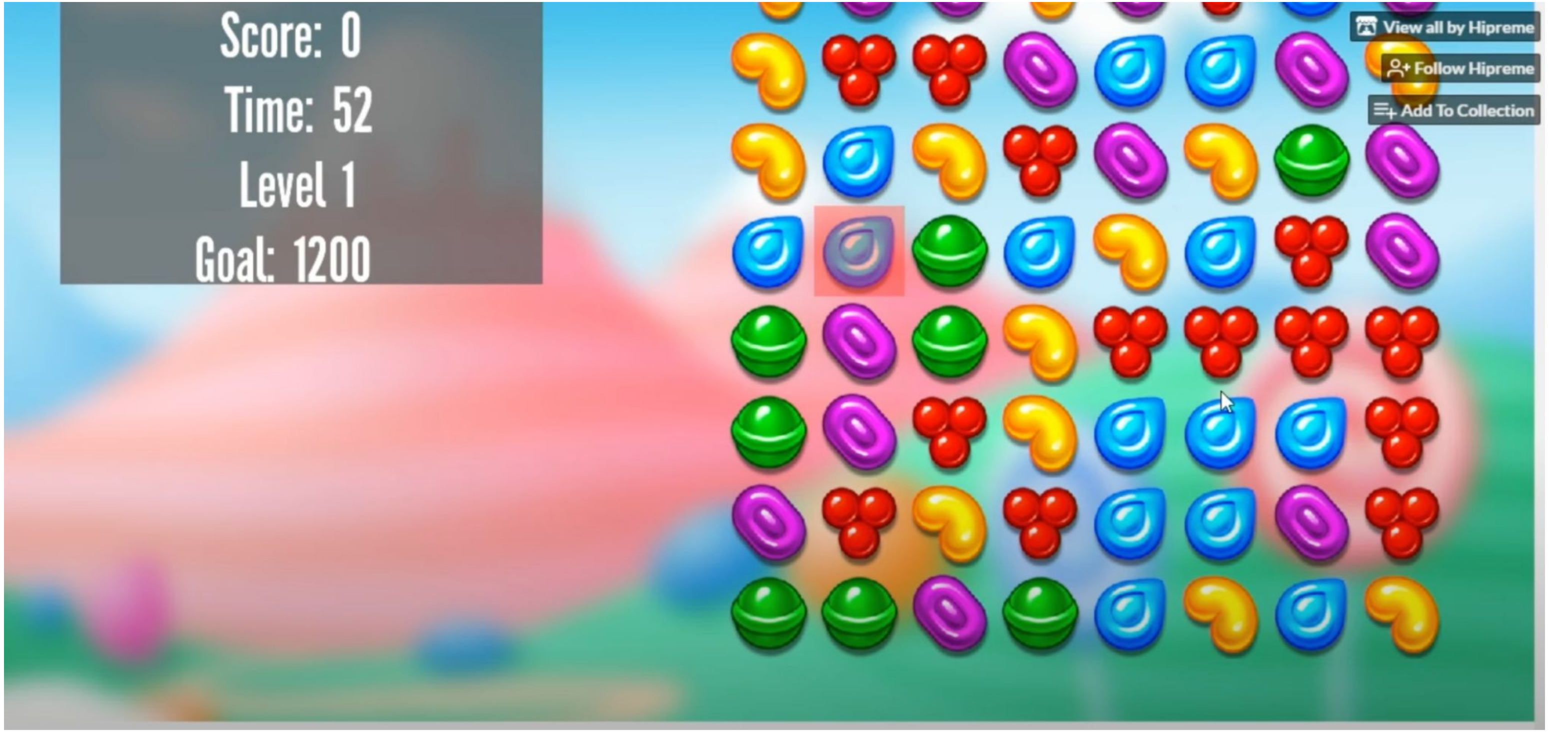




- Website with games: https://circularstudios.com/
- Github or Dub Repository: https://github.com/Circular-Studios/Dash
- Forum Post: https://forum.dlang.org/thread/qnaqymkehjvopwxwvwig@forum.dlang.org

Built in the D Programming Language

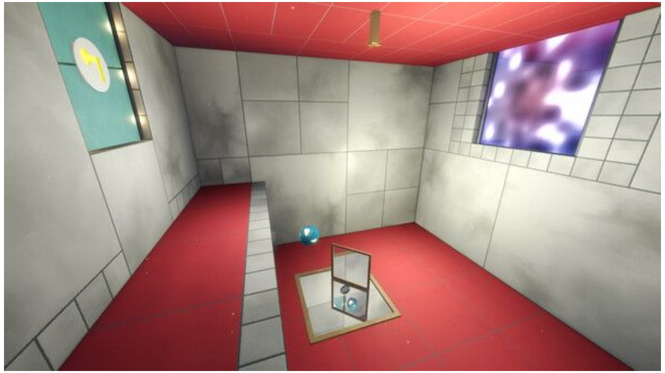
Hipreme Engine -- Game Engine



- Github or Dub Repository: https://github.com/MrcSnm/HipremeEngine
- DConf 2023 Talk: DConf '23 -- Hipreme Engine: Bringing D Everywhere -- Marcelo Mancini

Built in the D Programming Language

The Art of Reflections -- Game



- Steam Page: https://store.steampowered.com/app/2290770/The_Art_of_Reflection/
- D Forums 2025: https://forum.dlang.org/post/bwlxpoolebphvgrbbzcr@forum.dlang.org
  - Utilizing Direct3D 11 and PhysX

Built in the D Programming Language

Eilmer(/ɛlmə/) Compressible Flow Simulator

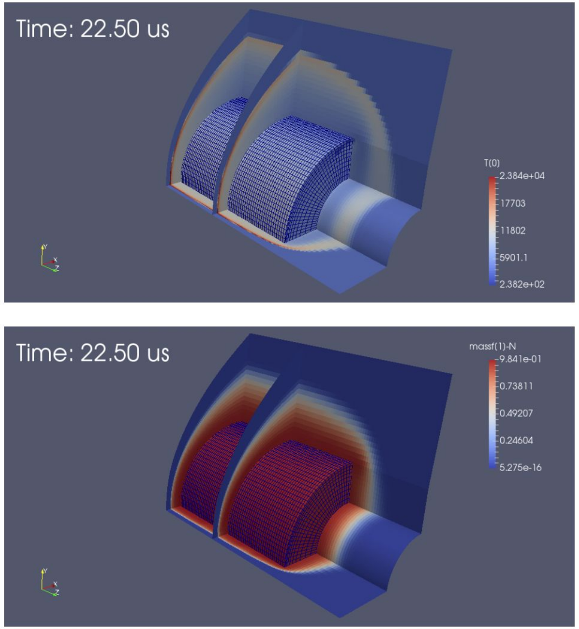
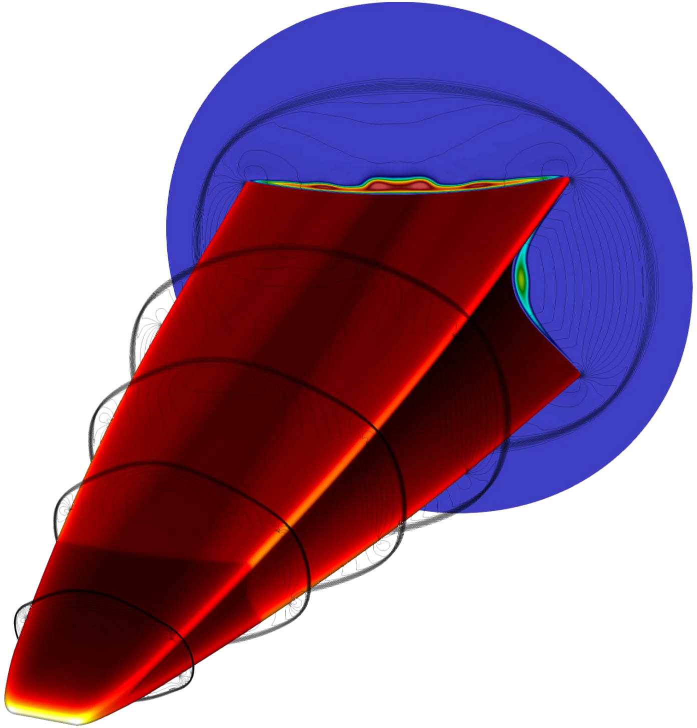
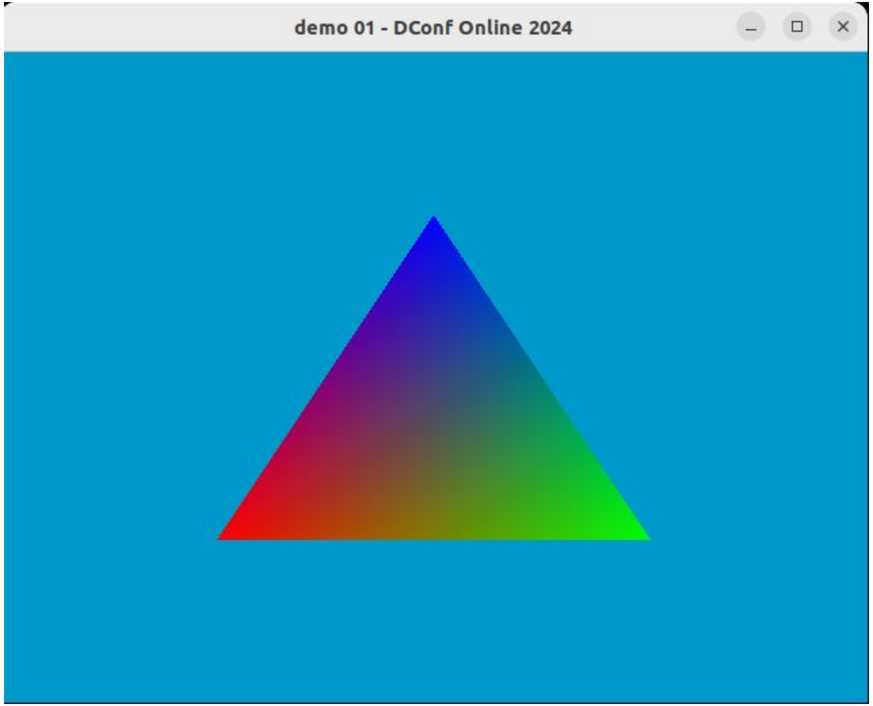


Figure 5.7: Static temperature and mass fraction of nitrogen atoms in the flow field from the chemical nonequilibrium simulation.



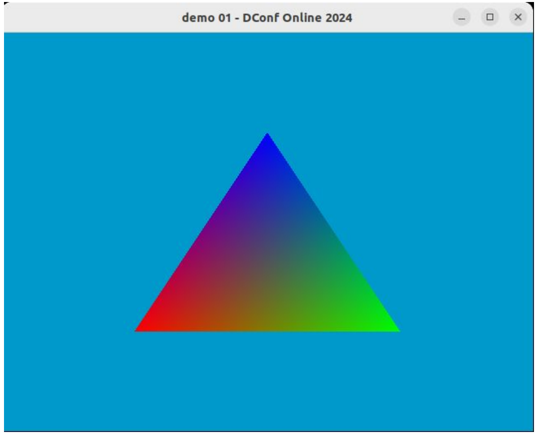
- Website: https://gdtk.uqcloud.net/ and https://gdtk.uqcloud.net/pdfs/eilmer-user-guide.pdf
- Github or Dub Repository: https://github.com/gdtk-uq/gdtk

Demo 1
First Triangle



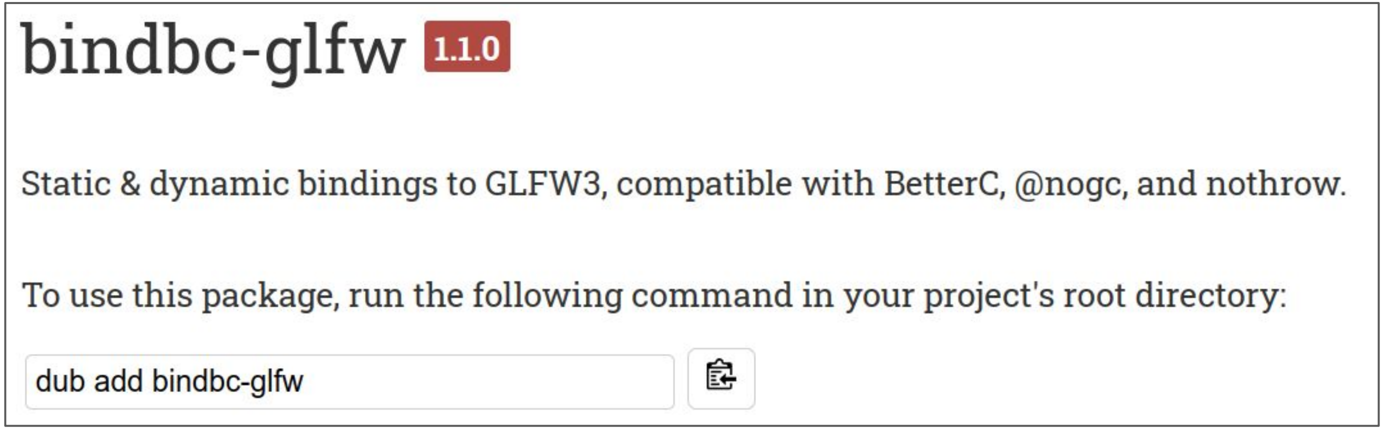
Graphics Programming Crash Course

- In order to get a triangle drawing using our a GPU we need a few things:
  - 1. A window
  - 2. To setup OpenGL (or your preferred graphics API)
  - 3. Upload data from the CPU to GPU (i.e. the graphics pipeline



Graphics Programming Crash Course - Window Setup (1/2)

- The easiest way to setup a window is to use a cross-platform windowing library like glfw or SDL
  - Mike Parker's bindbc-glfw or bindbc-sdl are great packages to get started
  - https://code.dlang.org/packages/bindbc-glfw
  - These packages are 'bindings' that otherwise expose the C functions calls from windowing libraries to D code



bindbc-glfw 1.1.0

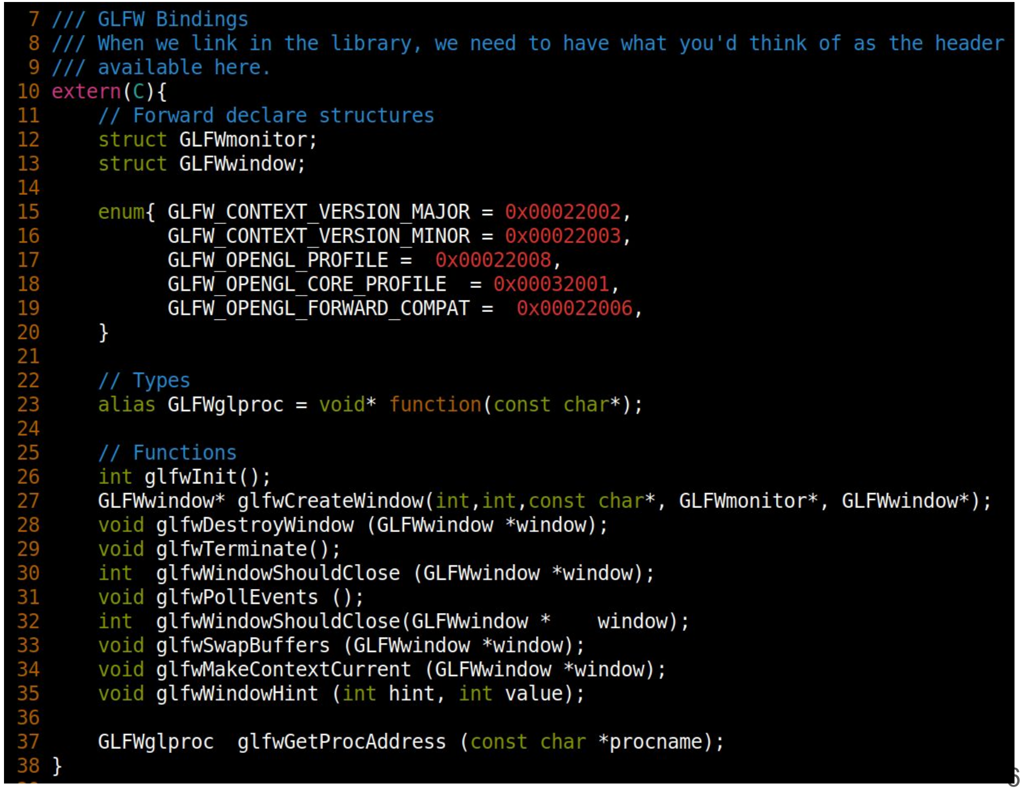
Static & dynamic bindings to GLFW3, compatible with BetterC, @nogc, and nothrow.

To use this package, run the following command in your project's root directory:

```
dub add bindbc-glfw
```

Graphics Programming Crash Course - Window Setup (2/2)

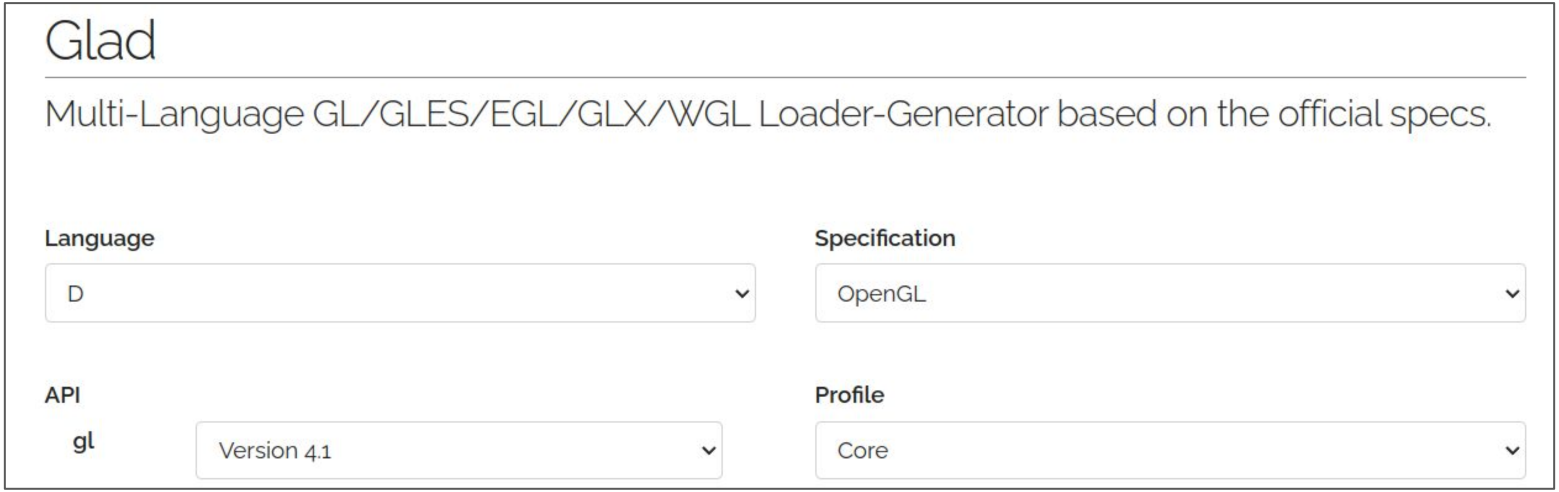
- You can avoid any ‘language bindings’ if you like as I show here
- In general, you should use the bindbc or other bindings however, as that way you’ll get a complete set of functions.
- But as you can see, talking to C code is as simple as either including the binding, or providing a function or type declaration, and then simply linking in the library
    - e.g. `-L-lglfw3`
        - -L -- passes a flag to the linker
        - -lglfw3 -- brings in the library
        - Additionally, you may specify the path to where to find the library file
            - e.g. `-L-L/usr/local/lib`



```d
/// GLFW Bindings
/// When we link in the library, we need to have what you'd think of as the header
/// available here.
extern(C){
    // Forward declare structures
    struct GLFWmonitor;
    struct GLFWwindow;

    enum{ GLFW_CONTEXT_VERSION_MAJOR = 0x00022002,
          GLFW_CONTEXT_VERSION_MINOR = 0x00022003,
          GLFW_OPENGL_PROFILE =  0x00022008,
          GLFW_OPENGL_CORE_PROFILE   = 0x00032001,
          GLFW_OPENGL_FORWARD_COMPAT =  0x00022006,
    }

    // Types
    alias GLFWglproc = void* function(const char*);

    // Functions
    int glfwInit();
    GLFWwindow* glfwCreateWindow(int,int,const char*, GLFWmonitor*, GLFWwindow*);
    void glfwDestroyWindow (GLFWwindow *window);
    void glfwTerminate();
    int  glfwWindowShouldClose (GLFWwindow *window);
    void glfwPollEvents ();
    int  glfwWindowShouldClose(GLFWwindow *    window);
    void glfwSwapBuffers (GLFWwindow *window);
    void glfwMakeContextCurrent (GLFWwindow *window);
    void glfwWindowHint (int hint, int value);

    GLFWglproc  glfwGetProcAddress (const char *procname);
}
```

Graphics Programming Crash Course - API Setup (1/4)

- For graphics APIs, then you need to typically ‘load’ the functions or extensions.
  - For OpenGL, you can use a tool like ‘glad’ to generate the C-function declarations for each function that your hardware supports.
    - https://glad.dav1d.de/



Glad

Multi-Language GL/GLES/EGL/GLX/WGL Loader-Generator based on the official specs.

Language: D

Specification: OpenGL

API: gl Version 4.1

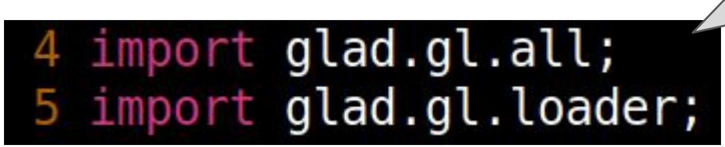
Profile: Core

Graphics Programming C

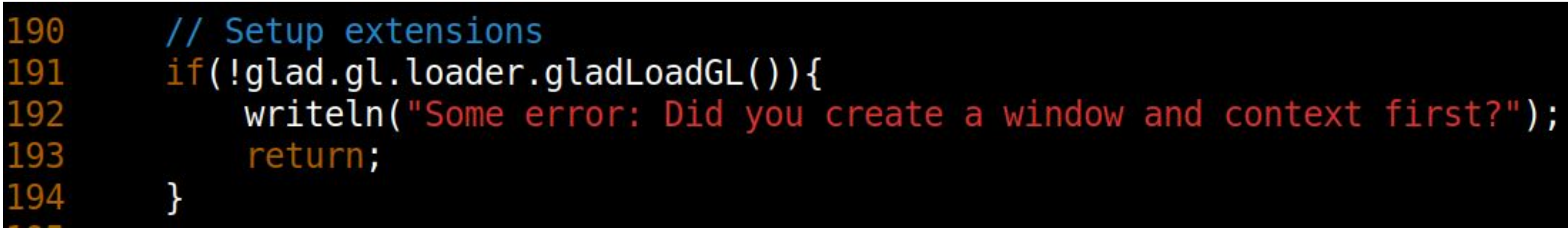
- For graphics APIs, then you ne
  extensions.
  - For OpenGL, you can use a tool lil
    for each function that your hardv

Now as we’re seeing our first D code -- let me mention the ‘D language’ advantage.

- D has a module system -- no need to mess with .h or .hpp files (in fact, there’s no preprocessor)
- Compiling with individual modules allows the DMD compiler to work super fast!



```d
import glad.gl.all;
import glad.gl.loader;
```

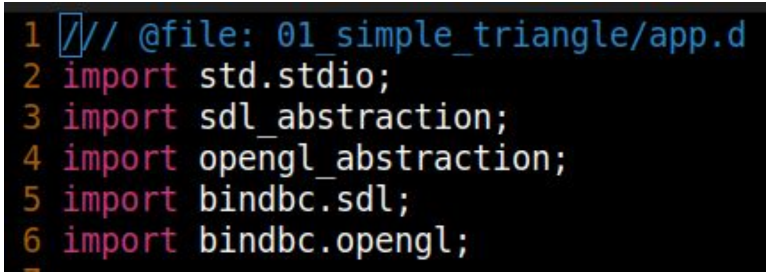


```d
    // Setup extensions
    if(!glad.gl.loader.gladLoadGL()){
        writeln("Some error: Did you create a window and context first?");
        return;
    }
```

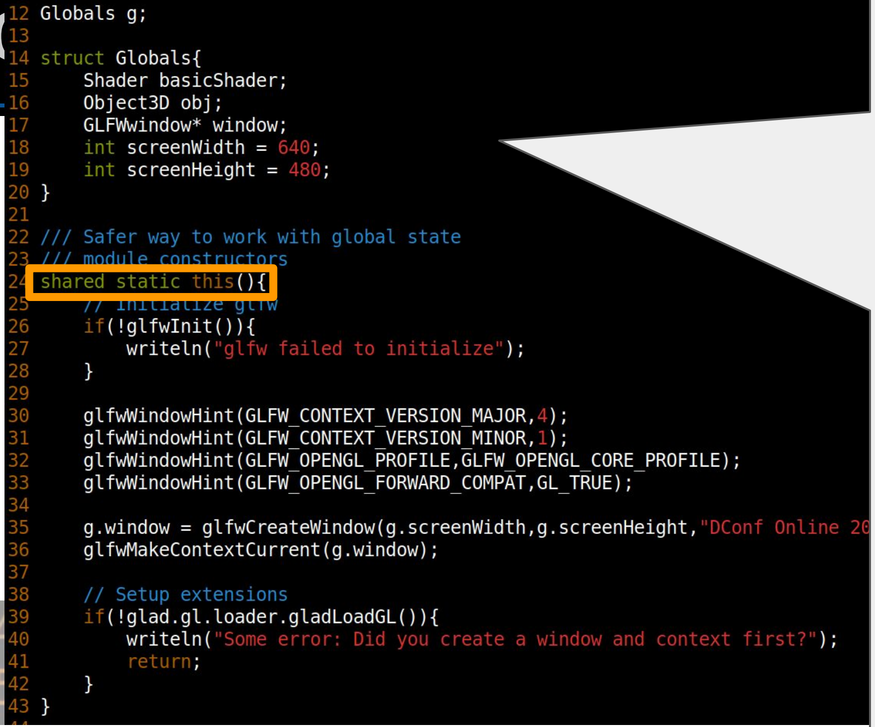
Graphics Programming C

- For graphics APIs, then you ne extensions.
  - For OpenGL, you can use a tool lil for each function that your hardy

- Note: If you don’t want to bother with the ‘C-style way’ (which I don’t), then you can use the ‘bindbc’ loaders to simply load OpenGL and your windowing library of choice



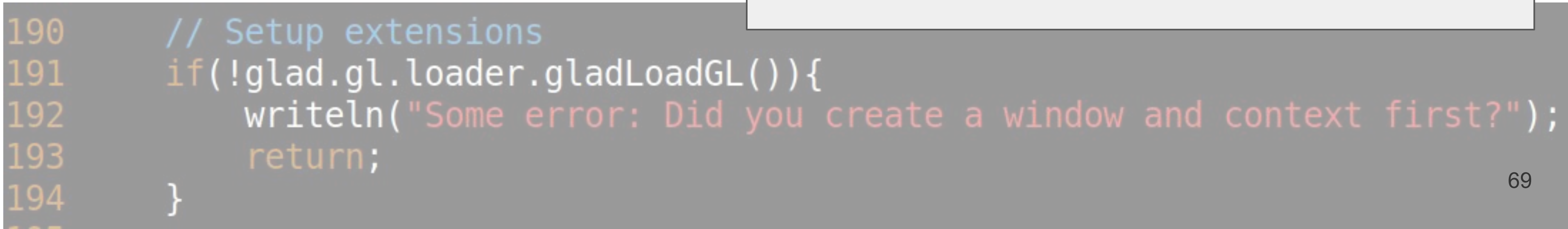
```
/// @file: 01_simple_triangle/app.d
import std.stdio;
import sdl_abstraction;
import opengl_abstraction;
import bindbc.sdl;
import bindbc.opengl;
```



```
Globals g;

struct Globals{
    Shader basicShader;
    Object3D obj;
    GLFWwindow* window;
    int screenWidth = 640;
    int screenHeight = 480;
}

/// Safer way to work with global state
/// module constructors
shared static this(){
    // initialize glfw
    if(!glfwInit()){
        writeln("glfw failed to initialize");
    }

    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR,4);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR,1);
    glfwWindowHint(GLFW_OPENGL_PROFILE,GLFW_OPENGL_CORE_PROFILE);
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT,GL_TRUE);

    g.window = glfwCreateWindow(g.screenWidth,g.screenHeight,"DConf Online 20
    glfwMakeContextCurrent(g.window);

    // Setup extensions
    if(!glad.gl.loader.gladLoadGL()){
        writeln("Some error: Did you create a window and context first?");
        return;
    }
}
```

Quality of life improvements

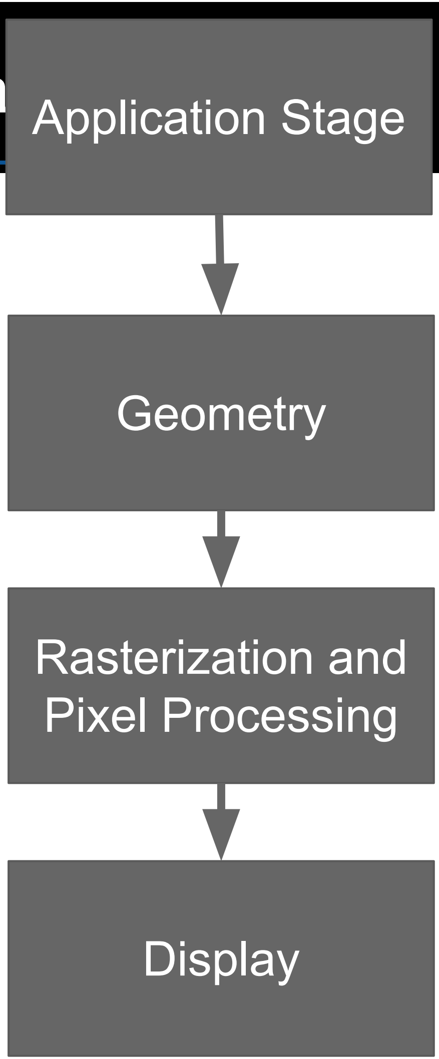
- Modules generally allow you to avoid worrying about the order you declare functions.
- There’s also ‘module level constructors’ that are called before main.
  - This can be clearly utilized if you have some initialization code -- like setting up a graphics API prior to its use
    - ‘**shared static this**’ means that block of code is called once ever (even amongst many threads) -- and this again is called before main() in lexicographical order



```
    // Setup extensions
    if(!glad.gl.loader.gladLoadGL()){
        writeln("Some error: Did you create a window and context first?");
        return;
    }
```

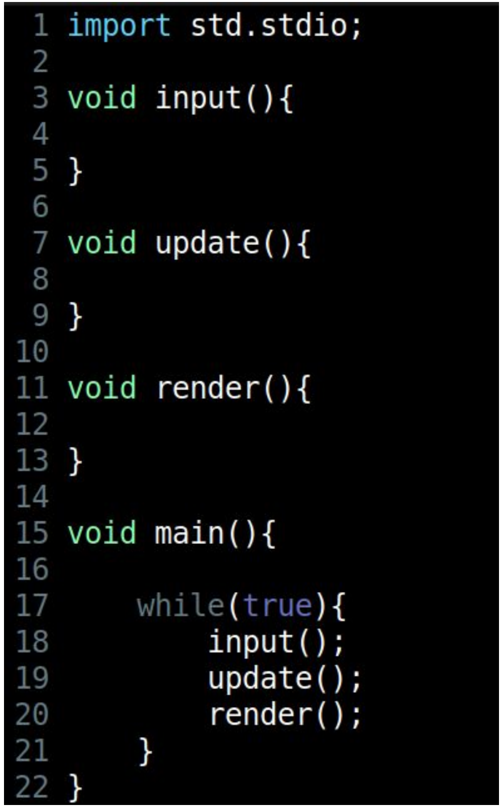
Graphics Pipelines - High Level Abstraction

- We now have OpenGL functions loaded (using glad), and a window setup (using glfw with our C binding)
- We are now ready to start doing some graphics programming using the OpenGL API

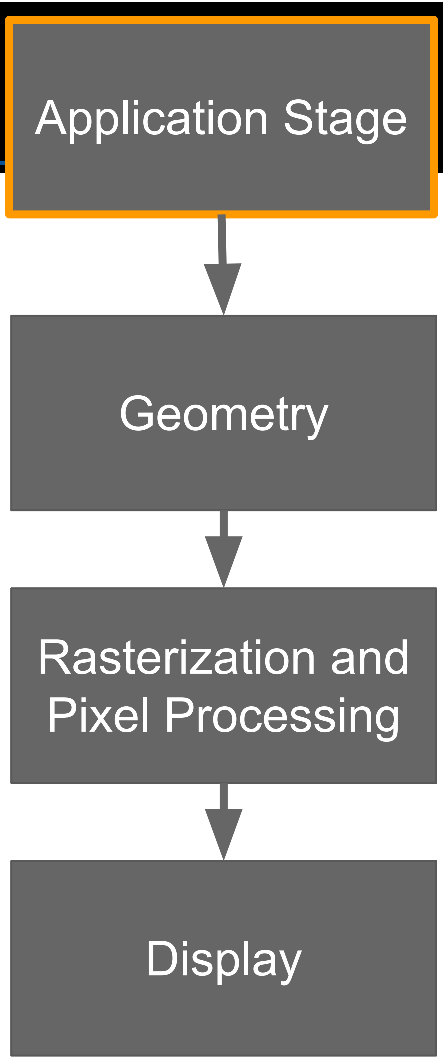


Graphics Pipelines - Application Stage

- At the application stage, this is our main loop
    - We also will ‘send’ geometric data at this stage from CPU to the GPU
    - The application stage otherwise is where all the ‘cpu’ work is completed:
        - File I/O
        - cpu memory allocation
        - Handling input



```
import std.stdio;

void input(){

}

void update(){

}

void render(){

}

void main(){

    while(true){
        input();
        update();
        render();
    }
}
```



Application Stage

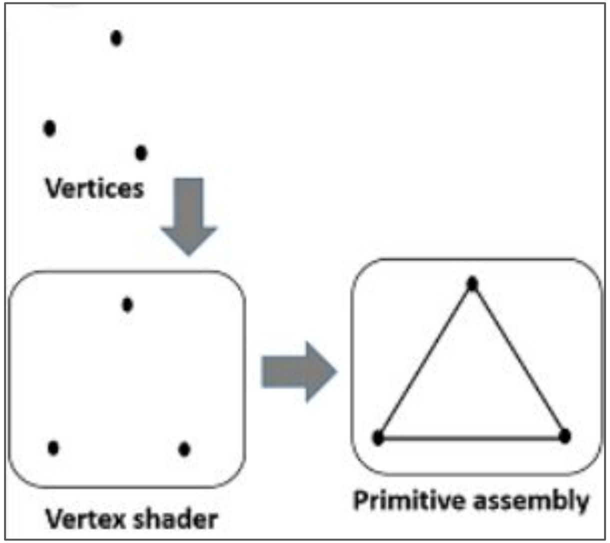
Geometry

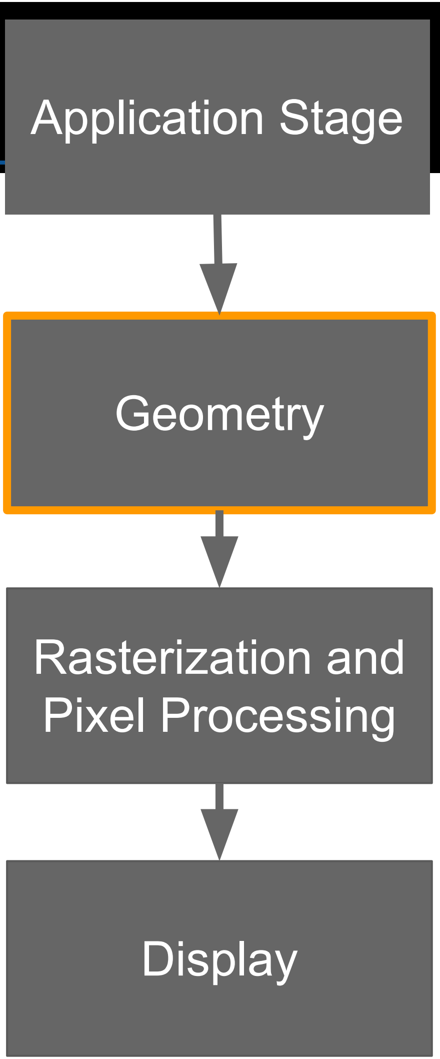
Rasterization and Pixel Processing

Display

Graphics Pipelines - Geometry Stage

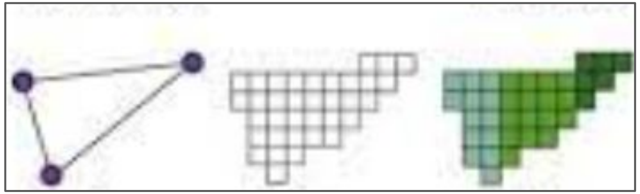
- At the geometry stage, we are now on the GPU
    - Data that has been sent to the GPU from the CPU is being assembled into primitives
    - Primitives may also be transformed (e.g. rotated, scaled, or translated)

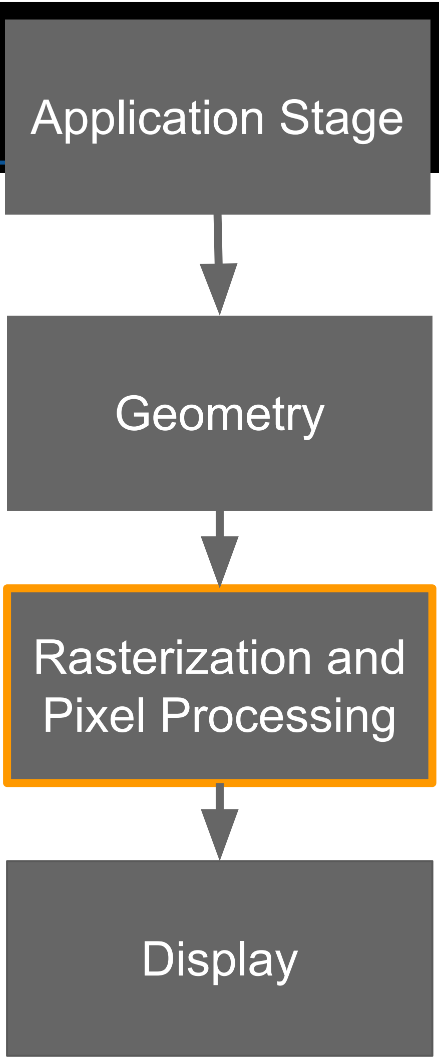




Graphics Pipelines - Rasterization

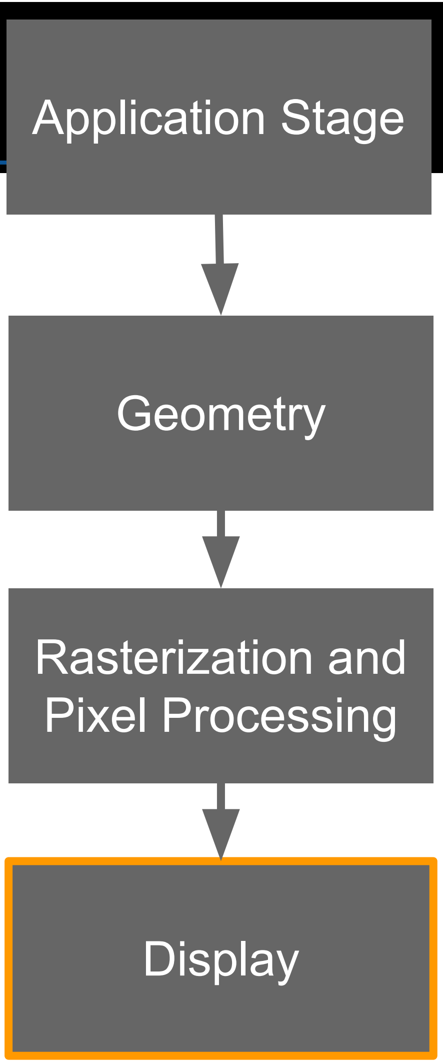
- At this stage, we represent our geometric shapes (e.g. triangles) as discrete pixels.
- We also color in those pixels based on their color and transparency





Graphics Pipelines - Display

- At the final stage you display the ‘frame’ that you have created.
  - This is stored in something known as a ‘framebuffer’ that at the least stores the colors of your pixels.



Application Stage

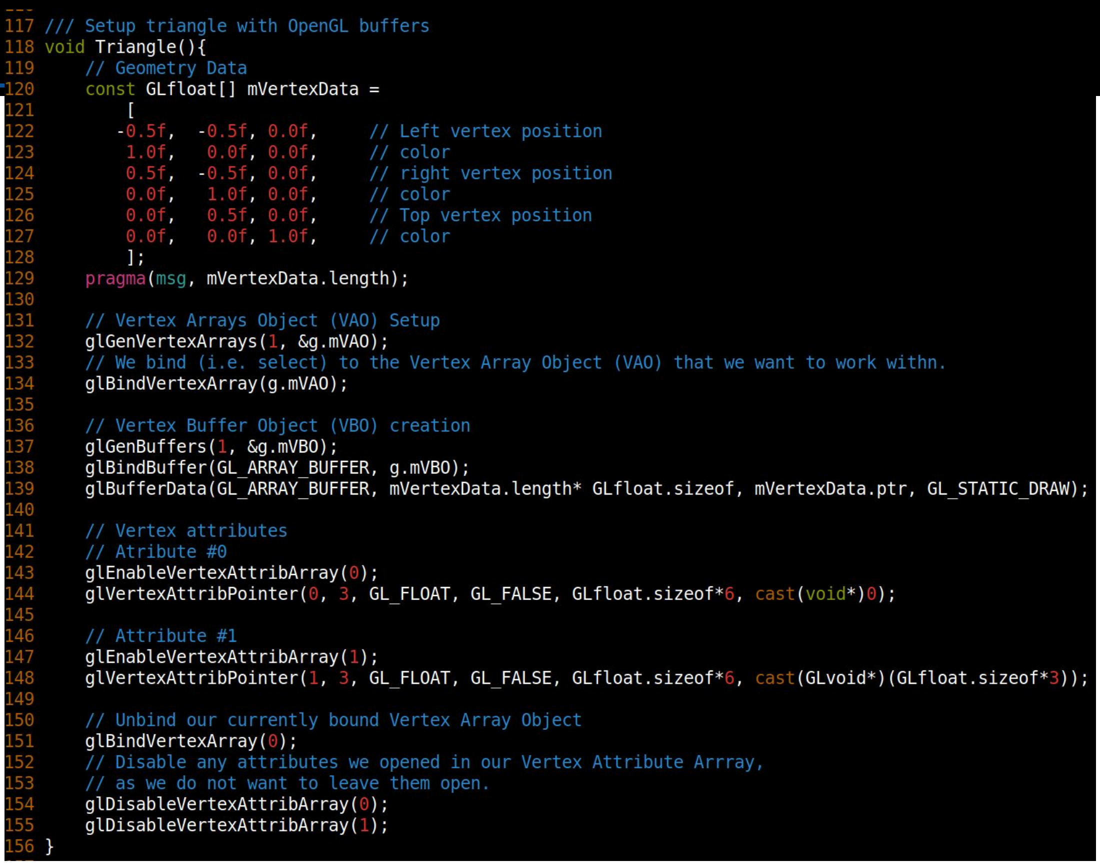
Geometry

Rasterization and Pixel Processing

Display

Displaying a Triangle (1/5)

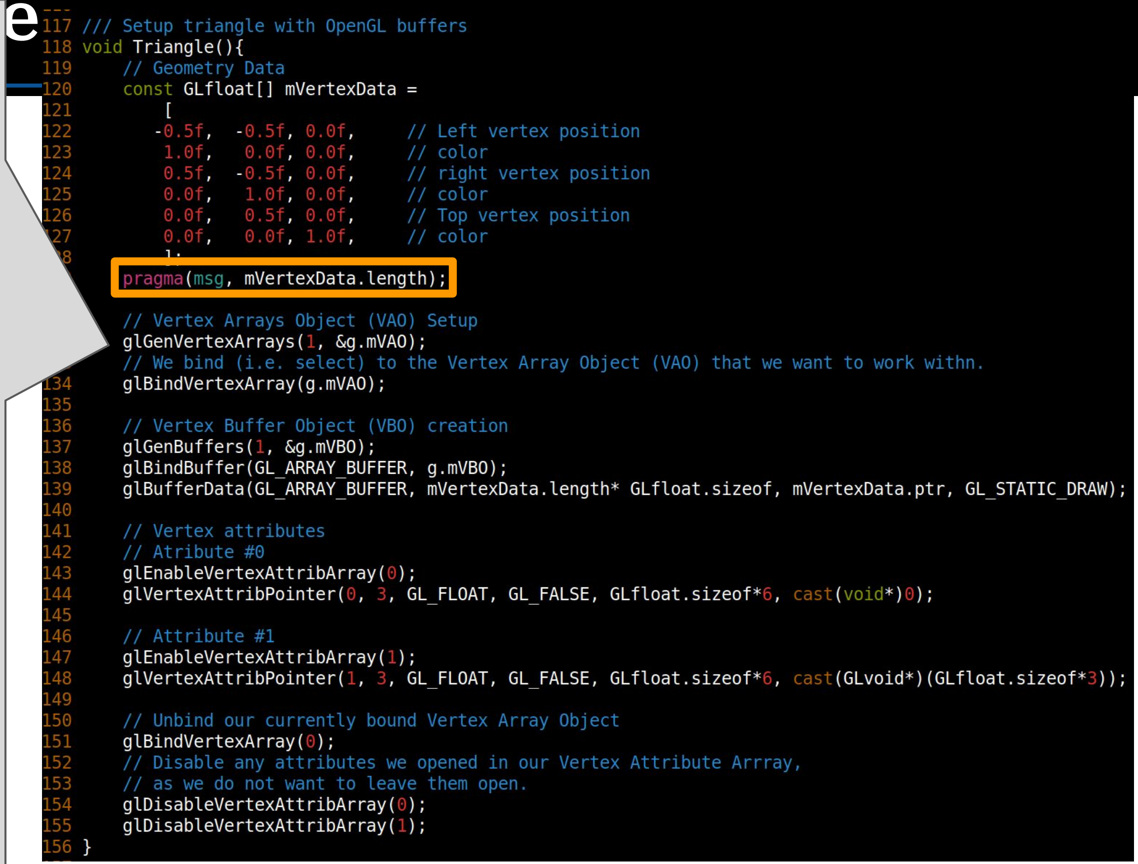
- To draw a triangle, we use OpenGL to upload data from the CPU to the GPU
    - For those who have done graphics programming -- this code is nearly the same as any C or C++ tutorial you will find
        - (i.e. all of the OpenGL functions are the same)



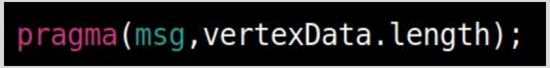
```
/// Setup triangle with OpenGL buffers
void Triangle(){
    // Geometry Data
    const GLfloat[] mVertexData =
        [
       -0.5f,  -0.5f, 0.0f,     // Left vertex position
        1.0f,   0.0f, 0.0f,     // color
        0.5f,  -0.5f, 0.0f,     // right vertex position
        0.0f,   1.0f, 0.0f,     // color
        0.0f,   0.5f, 0.0f,     // Top vertex position
        0.0f,   0.0f, 1.0f,     // color
        ];
    pragma(msg, mVertexData.length);

    // Vertex Arrays Object (VAO) Setup
    glGenVertexArrays(1, &g.mVAO);
    // We bind (i.e. select) to the Vertex Array Object (VAO) that we want to work withn.
    glBindVertexArray(g.mVAO);

    // Vertex Buffer Object (VBO) creation
    glGenBuffers(1, &g.mVBO);
    glBindBuffer(GL_ARRAY_BUFFER, g.mVBO);
    glBufferData(GL_ARRAY_BUFFER, mVertexData.length* GLfloat.sizeof, mVertexData.ptr, GL_STATIC_DRAW);

    // Vertex attributes
    // Atribute #0
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, GLfloat.sizeof*6, cast(void*)0);

    // Attribute #1
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, GLfloat.sizeof*6, cast(GLvoid*)(GLfloat.sizeof*3));

    // Unbind our currently bound Vertex Array Object
    glBindVertexArray(0);
    // Disable any attributes we opened in our Vertex Attribute Arrray,
    // as we do not want to leave them open.
    glDisableVertexAttribArray(0);
    glDisableVertexAttribArray(1);
}
```

```
pragma(msg,vertexData.length);
```

- One small change from C or C++ is this line above.
  - D’s Compile-Time Function Execution (CTFE) and general introspection capabilities can be useful for catching bugs at compile-time
- The pragma I stuck in here is to confirm at compile-time I have the right amount of data.
  - Arrays are also ‘bounds checked’ for safety (can be turned off if needed)

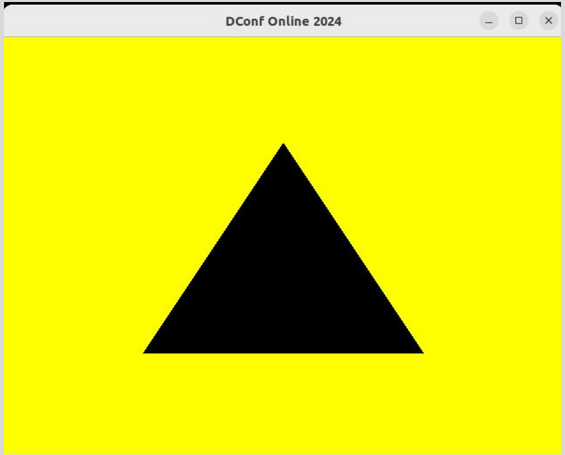


```
/// Setup triangle with OpenGL buffers
void Triangle(){
    // Geometry Data
    const GLfloat[] mVertexData =
        [
       -0.5f,  -0.5f, 0.0f,      // Left vertex position
        1.0f,   0.0f, 0.0f,      // color
        0.5f,  -0.5f, 0.0f,      // right vertex position
        0.0f,   1.0f, 0.0f,      // color
        0.0f,   0.5f, 0.0f,      // Top vertex position
        0.0f,   0.0f, 1.0f,      // color
        ];
    pragma(msg, mVertexData.length);

    // Vertex Arrays Object (VAO) Setup
    glGenVertexArrays(1, &g.mVAO);
    // We bind (i.e. select) to the Vertex Array Object (VAO) that we want to work withn.
    glBindVertexArray(g.mVAO);

    // Vertex Buffer Object (VBO) creation
    glGenBuffers(1, &g.mVBO);
    glBindBuffer(GL_ARRAY_BUFFER, g.mVBO);
    glBufferData(GL_ARRAY_BUFFER, mVertexData.length* GLfloat.sizeof, mVertexData.ptr, GL_STATIC_DRAW);

    // Vertex attributes
    // Atribute #0
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, GLfloat.sizeof*6, cast(void*)0);

    // Attribute #1
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, GLfloat.sizeof*6, cast(GLvoid*)(GLfloat.sizeof*3));

    // Unbind our currently bound Vertex Array Object
    glBindVertexArray(0);
    // Disable any attributes we opened in our Vertex Attribute Arrray,
    // as we do not want to leave them open.
    glDisableVertexAttribArray(0);
    glDisableVertexAttribArray(1);
}
```



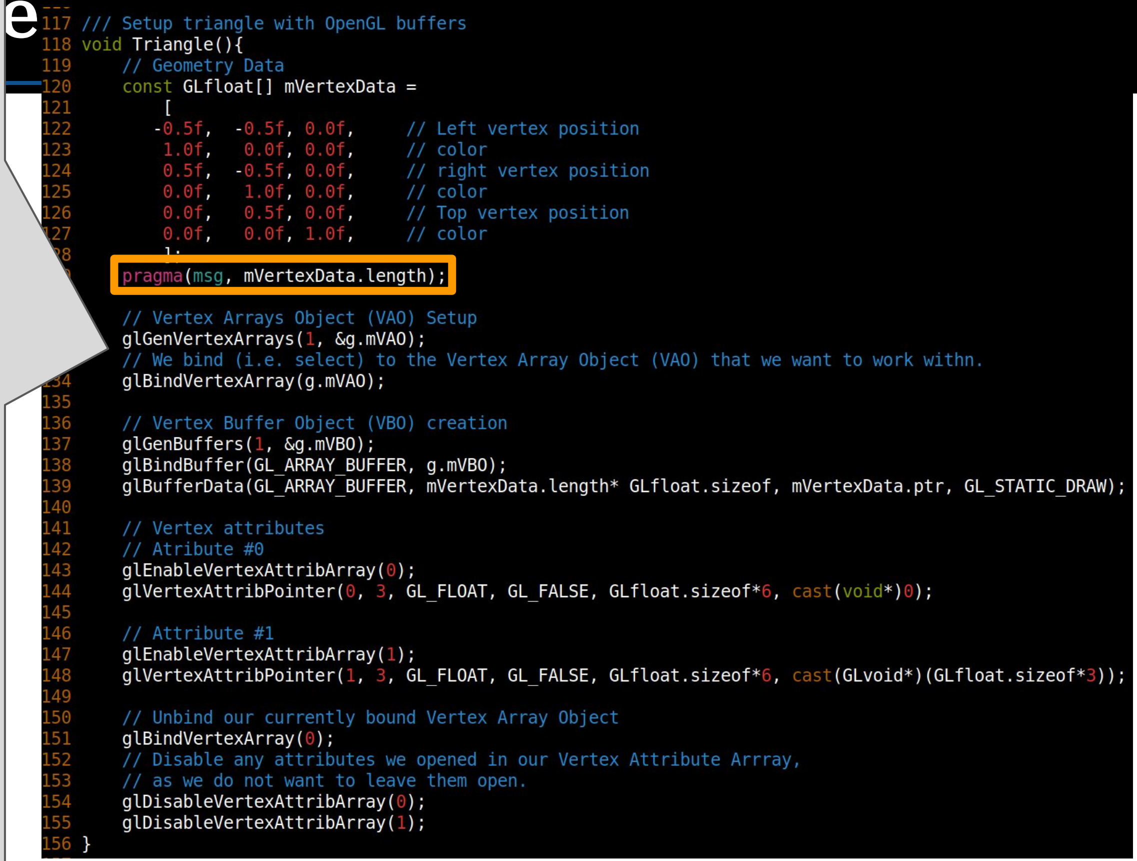
```
pragma(msg,vertexData.length);
```

- See this example below when I did not populate color data properly

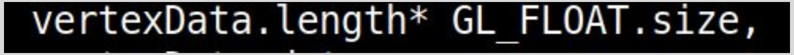


Example of a ‘mistake’ I made in preparation of the demo

- ‘static asserts’ can also be placed to further write code more solid code.

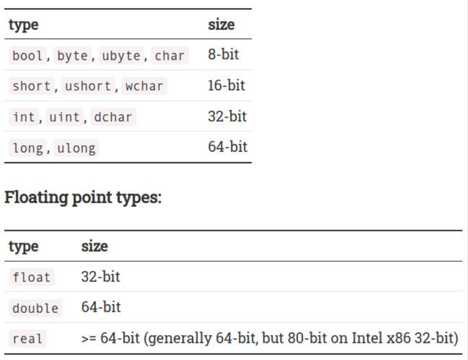


```
/// Setup triangle with OpenGL buffers
void Triangle(){
    // Geometry Data
    const GLfloat[] mVertexData =
        [
        -0.5f,  -0.5f, 0.0f,      // Left vertex position
         1.0f,   0.0f, 0.0f,      // color
         0.5f,  -0.5f, 0.0f,      // right vertex position
         0.0f,   1.0f, 0.0f,      // color
         0.0f,   0.5f, 0.0f,      // Top vertex position
         0.0f,   0.0f, 1.0f,      // color
        ];
    pragma(msg, mVertexData.length);

    // Vertex Arrays Object (VAO) Setup
    glGenVertexArrays(1, &g.mVAO);
    // We bind (i.e. select) to the Vertex Array Object (VAO) that we want to work withn.
    glBindVertexArray(g.mVAO);

    // Vertex Buffer Object (VBO) creation
    glGenBuffers(1, &g.mVBO);
    glBindBuffer(GL_ARRAY_BUFFER, g.mVBO);
    glBufferData(GL_ARRAY_BUFFER, mVertexData.length* GLfloat.sizeof, mVertexData.ptr, GL_STATIC_DRAW);

    // Vertex attributes
    // Atribute #0
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, GLfloat.sizeof*6, cast(void*)0);

    // Attribute #1
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, GLfloat.sizeof*6, cast(GLvoid*)(GLfloat.sizeof*3));

    // Unbind our currently bound Vertex Array Object
    glBindVertexArray(0);
    // Disable any attributes we opened in our Vertex Attribute Arrray,
    // as we do not want to leave them open.
    glDisableVertexAttribArray(0);
    glDisableVertexAttribArray(1);
}
```



```
vertexData.length* GL_FLOAT.size,
```

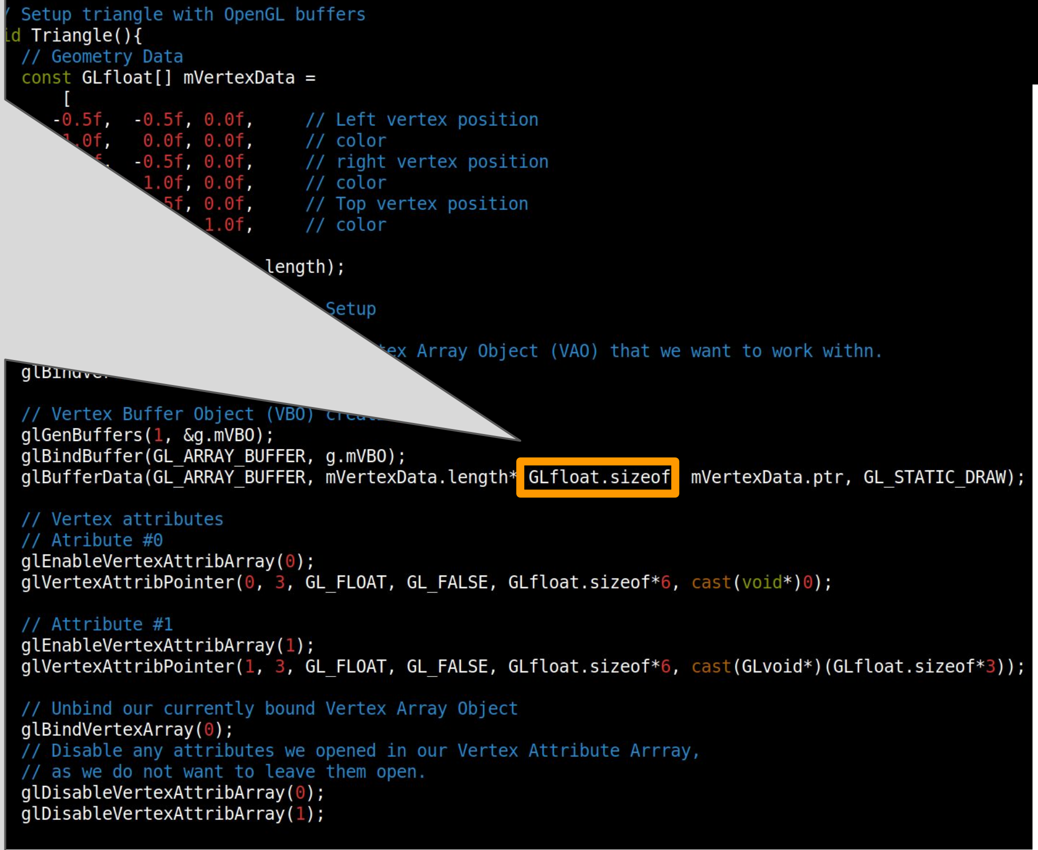
- The enum ‘GL_FLOAT’ above is actually an ‘integer’ type in the OpenGL API
    - The ‘float’ type we actually want is the ‘alias’ to GLfloat shown in the code
    - We could use a static assert at compile-time with `GLfloat.sizeof` to ensure it meets our size requirements
- Luckily however, D’s basic types have predictable **fixed sizes** [table]



type	size
`bool`, `byte`, `ubyte`, `char`	8-bit
`short`, `ushort`, `wchar`	16-bit
`int`, `uint`, `dchar`	32-bit
`long`, `ulong`	64-bit

Floating point types:

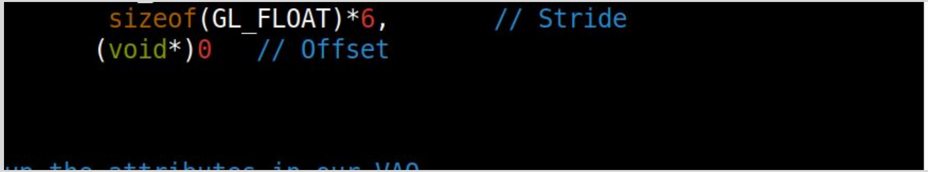
type	size
`float`	32-bit
`double`	64-bit
`real`	>= 64-bit (generally 64-bit, but 80-bit on Intel x86 32-bit)



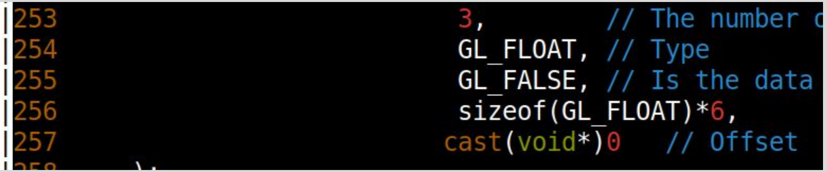
```
/ Setup triangle with OpenGL buffers
id Triangle(){
  // Geometry Data
  const GLfloat[] mVertexData =
      [
     -0.5f,  -0.5f, 0.0f,     // Left vertex position
      1.0f,   0.0f, 0.0f,     // color
          ,  -0.5f, 0.0f,     // right vertex position
              1.0f, 0.0f,     // color
               5f, 0.0f,     // Top vertex position
                    1.0f,     // color

                             length);

                                 Setup

                                     ex Array Object (VAO) that we want to work withn.
  glBindVe

  // Vertex Buffer Object (VBO) crea
  glGenBuffers(1, &g.mVBO);
  glBindBuffer(GL_ARRAY_BUFFER, g.mVBO);
  glBufferData(GL_ARRAY_BUFFER, mVertexData.length* GLfloat.sizeof, mVertexData.ptr, GL_STATIC_DRAW);

  // Vertex attributes
  // Atribute #0
  glEnableVertexAttribArray(0);
  glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, GLfloat.sizeof*6, cast(void*)0);

  // Attribute #1
  glEnableVertexAttribArray(1);
  glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, GLfloat.sizeof*6, cast(GLvoid*)(GLfloat.sizeof*3));

  // Unbind our currently bound Vertex Array Object
  glBindVertexArray(0);
  // Disable any attributes we opened in our Vertex Attribute Arrray,
  // as we do not want to leave them open.
  glDisableVertexAttribArray(0);
  glDisableVertexAttribArray(1);
```

Displaying a Triangle (5/5)

- Other quality of life features include things like explicit casting using the ‘cast’ keyword
    - (C on the left, and D on the right)

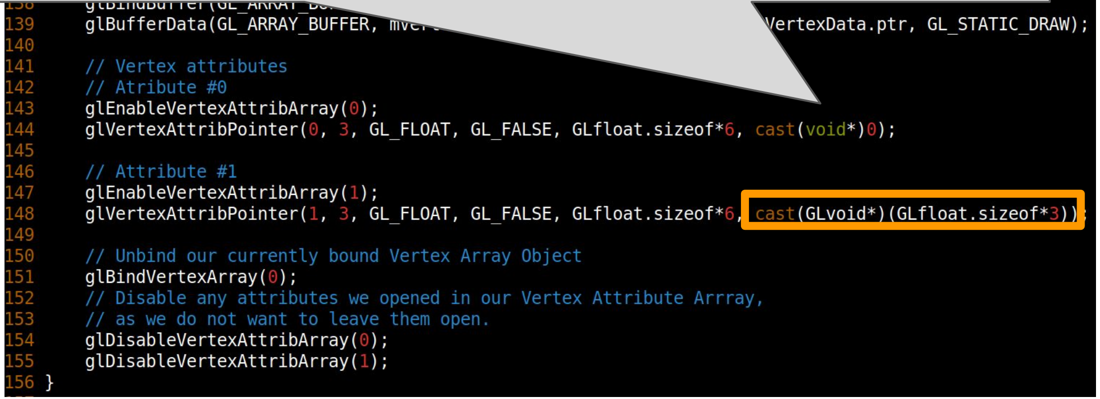


```
    sizeof(GL_FLOAT)*6,         // Stride
   (void*)0    // Offset
```



```
        3,         // The number o
        GL_FLOAT, // Type
        GL_FALSE, // Is the data
        sizeof(GL_FLOAT)*6,
       cast(void*)0    // Offset
```

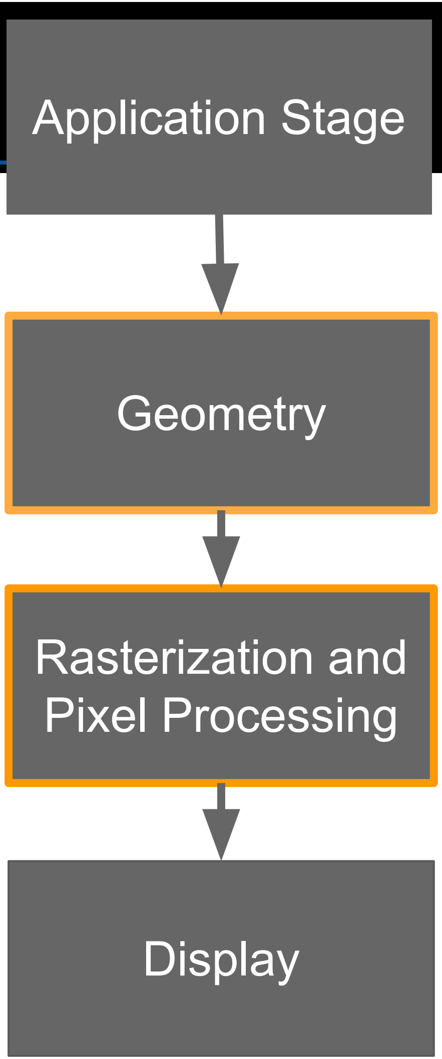
    - For those who have done graphics programming -- this code is nearly the same as any C or C++ tutorial you will find
        - (i.e. all of the OpenGL functions are the same)



```
/// Setup triangle with OpenGL buffers
    glBufferData(GL_ARRAY_BUFFER, mVer                                          VertexData.ptr, GL_STATIC_DRAW);

    // Vertex attributes
    // Atribute #0
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, GLfloat.sizeof*6, cast(void*)0);

    // Attribute #1
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, GLfloat.sizeof*6, cast(GLvoid*)(GLfloat.sizeof*3));

    // Unbind our currently bound Vertex Array Object
    glBindVertexArray(0);
    // Disable any attributes we opened in our Vertex Attribute Arrray,
    // as we do not want to leave them open.
    glDisableVertexAttribArray(0);
    glDisableVertexAttribArray(1);
}
```

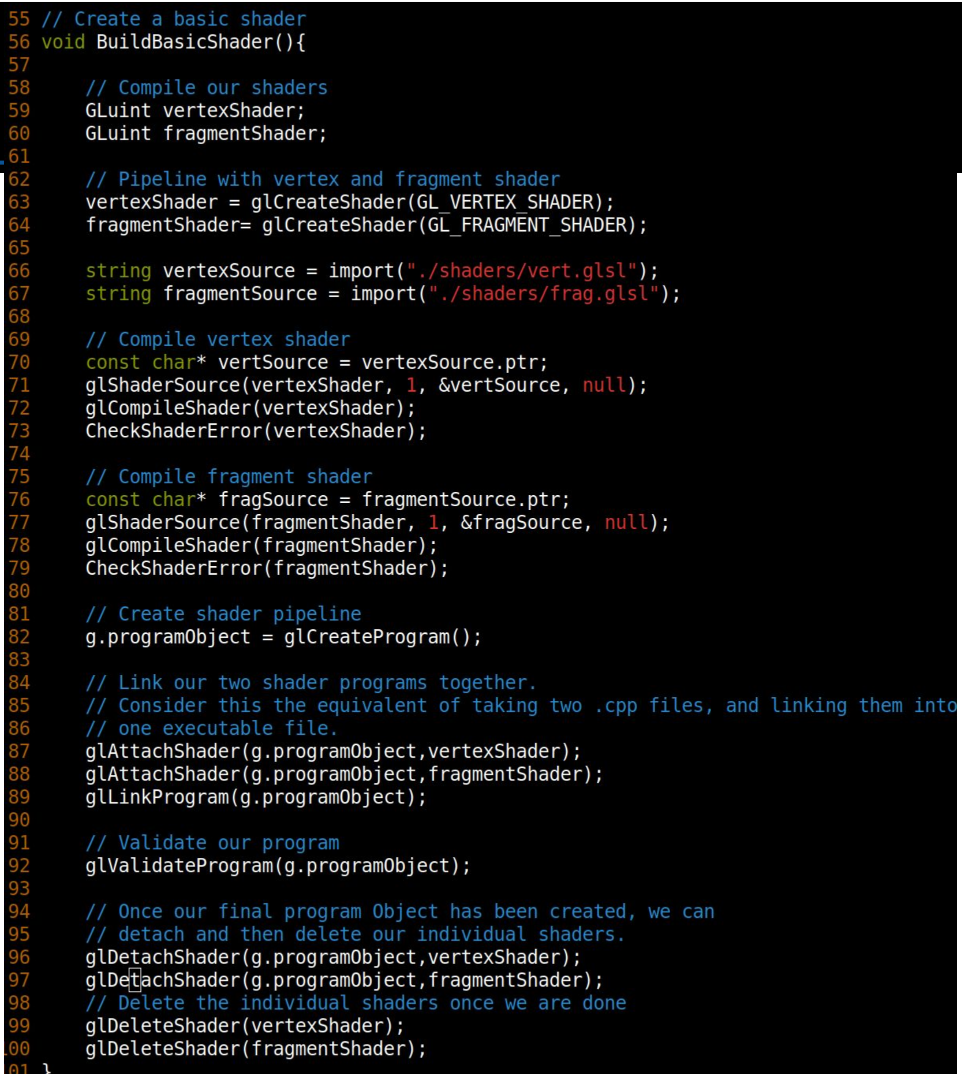
Graphics Pipelines - Shaders

- Now in order to actually do something, we have to create a graphics pipeline
  - This is done by processing our geometry in a GPU program called a 'vertex' or shader.
  - We then also write one other GPU program called a 'fragment' or 'pixel' shader



Shader Code (1/2)

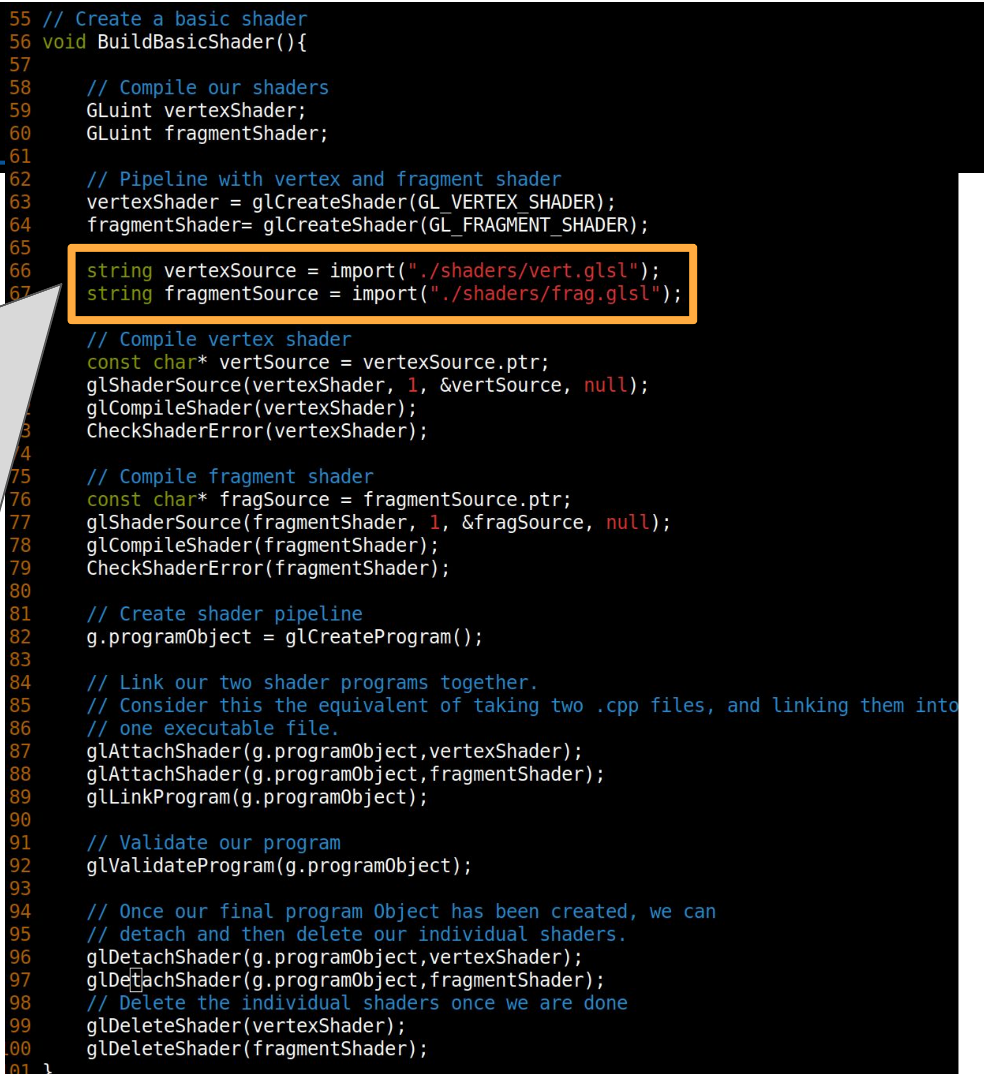
- To the right is all the shader code needed
  - (Error checking separated out into one other function)



```
// Create a basic shader
void BuildBasicShader(){

    // Compile our shaders
    GLuint vertexShader;
    GLuint fragmentShader;

    // Pipeline with vertex and fragment shader
    vertexShader = glCreateShader(GL_VERTEX_SHADER);
    fragmentShader= glCreateShader(GL_FRAGMENT_SHADER);

    string vertexSource = import("./shaders/vert.glsl");
    string fragmentSource = import("./shaders/frag.glsl");

    // Compile vertex shader
    const char* vertSource = vertexSource.ptr;
    glShaderSource(vertexShader, 1, &vertSource, null);
    glCompileShader(vertexShader);
    CheckShaderError(vertexShader);

    // Compile fragment shader
    const char* fragSource = fragmentSource.ptr;
    glShaderSource(fragmentShader, 1, &fragSource, null);
    glCompileShader(fragmentShader);
    CheckShaderError(fragmentShader);

    // Create shader pipeline
    g.programObject = glCreateProgram();

    // Link our two shader programs together.
    // Consider this the equivalent of taking two .cpp files, and linking them into
    // one executable file.
    glAttachShader(g.programObject,vertexShader);
    glAttachShader(g.programObject,fragmentShader);
    glLinkProgram(g.programObject);

    // Validate our program
    glValidateProgram(g.programObject);

    // Once our final program Object has been created, we can
    // detach and then delete our individual shaders.
    glDetachShader(g.programObject,vertexShader);
    glDetachShader(g.programObject,fragmentShader);
    // Delete the individual shaders once we are done
    glDeleteShader(vertexShader);
    glDeleteShader(fragmentShader);
}
```

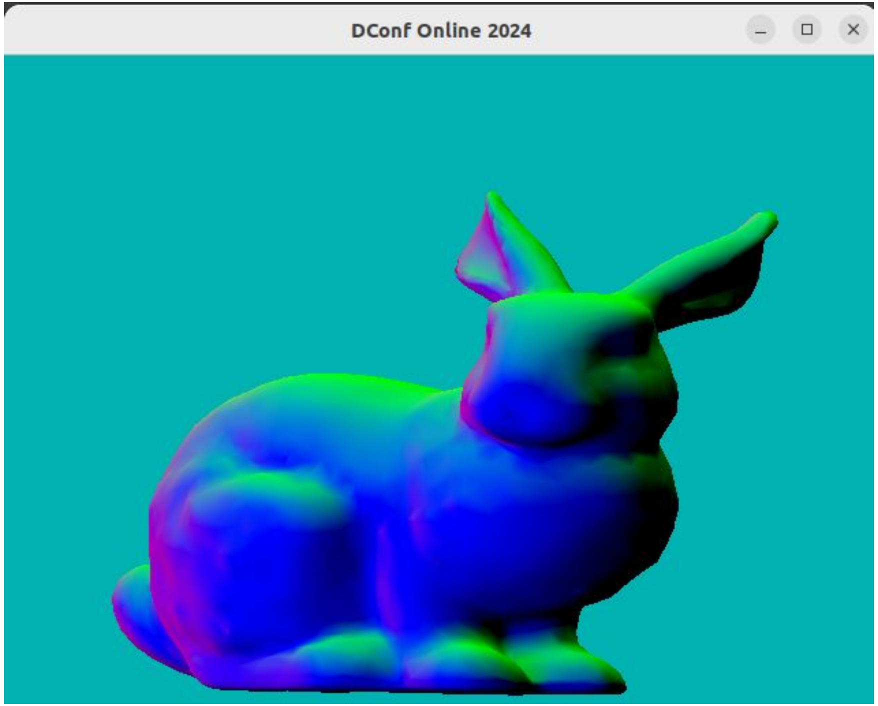
Shader Code (2/2)

- One interesting thing for this demo is I did not bother to write any code to load the shaders from a file on disk.
    - Instead, I just imported the code (similar to C23's upcoming #embed) feature.
- The advantage here is:
    - 1. primarily simplicity for small programs [more on working with C strings]
    - 2. If I do want to embed code as data, it's relatively straightforward if I do not want to go to disk

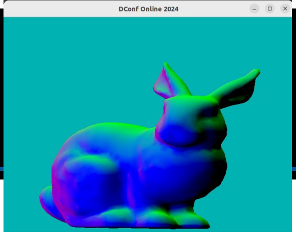


```
// Create a basic shader
void BuildBasicShader(){

    // Compile our shaders
    GLuint vertexShader;
    GLuint fragmentShader;

    // Pipeline with vertex and fragment shader
    vertexShader = glCreateShader(GL_VERTEX_SHADER);
    fragmentShader= glCreateShader(GL_FRAGMENT_SHADER);

    string vertexSource = import("./shaders/vert.glsl");
    string fragmentSource = import("./shaders/frag.glsl");

    // Compile vertex shader
    const char* vertSource = vertexSource.ptr;
    glShaderSource(vertexShader, 1, &vertSource, null);
    glCompileShader(vertexShader);
    CheckShaderError(vertexShader);

    // Compile fragment shader
    const char* fragSource = fragmentSource.ptr;
    glShaderSource(fragmentShader, 1, &fragSource, null);
    glCompileShader(fragmentShader);
    CheckShaderError(fragmentShader);

    // Create shader pipeline
    g.programObject = glCreateProgram();

    // Link our two shader programs together.
    // Consider this the equivalent of taking two .cpp files, and linking them into
    // one executable file.
    glAttachShader(g.programObject,vertexShader);
    glAttachShader(g.programObject,fragmentShader);
    glLinkProgram(g.programObject);

    // Validate our program
    glValidateProgram(g.programObject);

    // Once our final program Object has been created, we can
    // detach and then delete our individual shaders.
    glDetachShader(g.programObject,vertexShader);
    glDetachShader(g.programObject,fragmentShader);
    // Delete the individual shaders once we are done
    glDeleteShader(vertexShader);
    glDeleteShader(fragmentShader);
}
```

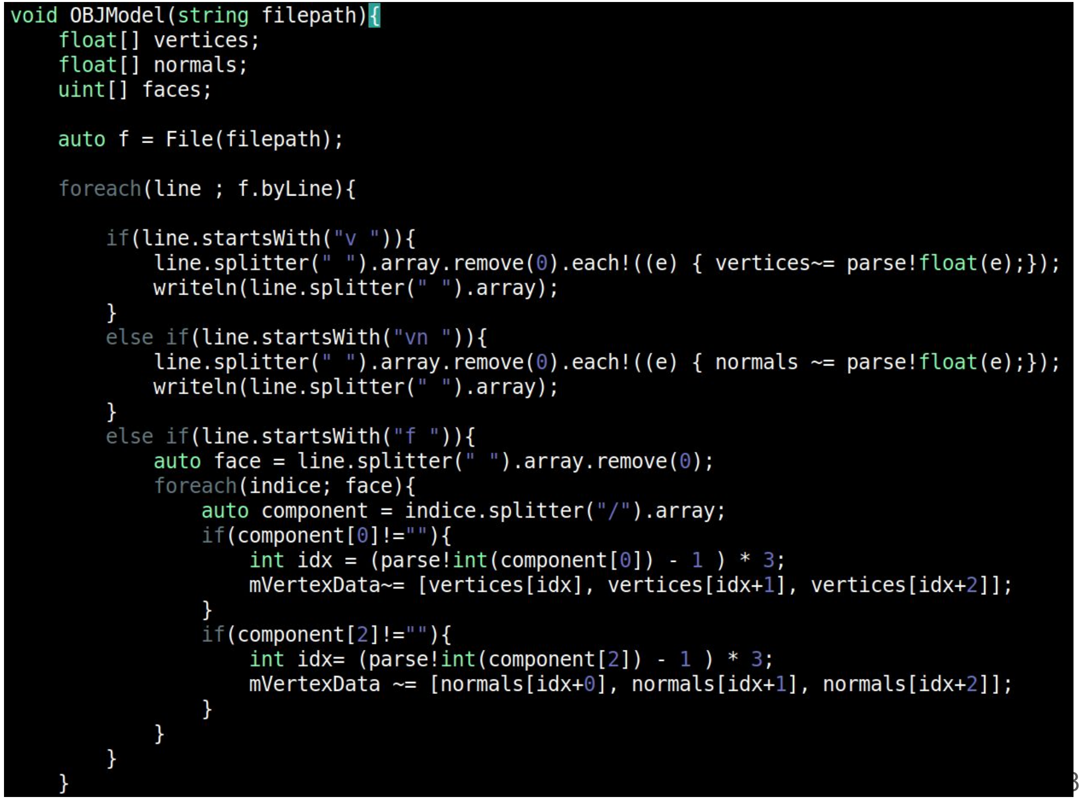
Demo 2
Objects



Parsing Structured Data



- If we want to draw something more interesting than triangles, we will load that data from a file.
- To the right -- is the entire parser for the .obj file.

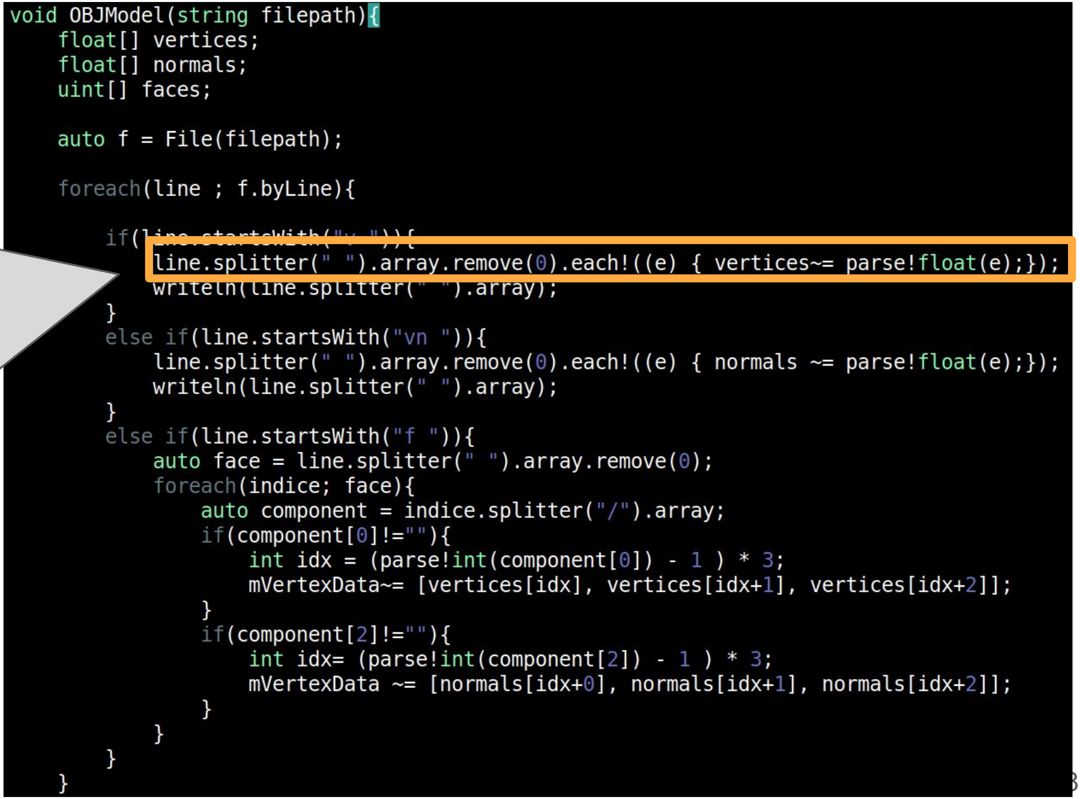


```
void OBJModel(string filepath){
    float[] vertices;
    float[] normals;
    uint[] faces;

    auto f = File(filepath);

    foreach(line ; f.byLine){

        if(line.startsWith("v ")){
            line.splitter(" ").array.remove(0).each!((e) { vertices~= parse!float(e);});
            writeln(line.splitter(" ").array);
        }
        else if(line.startsWith("vn ")){
            line.splitter(" ").array.remove(0).each!((e) { normals ~= parse!float(e);});
            writeln(line.splitter(" ").array);
        }
        else if(line.startsWith("f ")){
            auto face = line.splitter(" ").array.remove(0);
            foreach(indice; face){
                auto component = indice.splitter("/").array;
                if(component[0]!=""){
                    int idx = (parse!int(component[0]) - 1 ) * 3;
                    mVertexData~= [vertices[idx], vertices[idx+1], vertices[idx+2]];
                }
                if(component[2]!=""){
                    int idx= (parse!int(component[2]) - 1 ) * 3;
                    mVertexData ~= [normals[idx+0], normals[idx+1], normals[idx+2]];
                }
            }
        }
    }
```

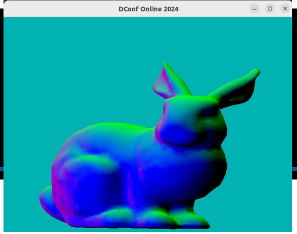
Parsing Structured Data

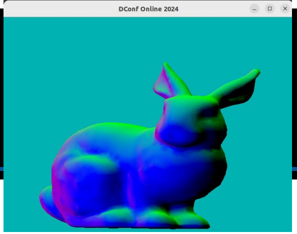
- If we want to draw something more

- Observe where uniform function call syntax (UFCS) really shines allowing us to right concise and readable code.



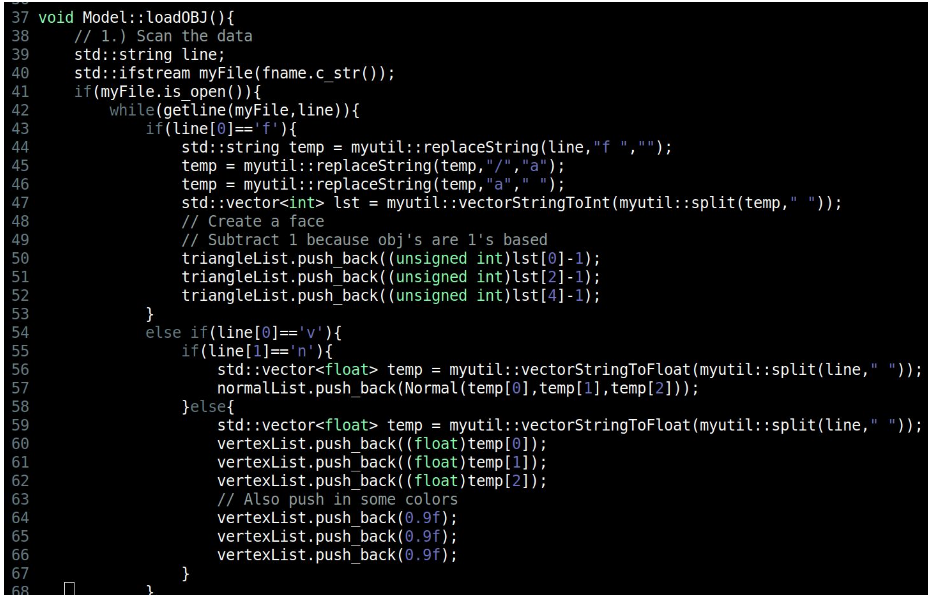
```
void OBJModel(string filepath){
    float[] vertices;
    float[] normals;
    uint[] faces;

    auto f = File(filepath);

    foreach(line ; f.byLine){

        if(line.startsWith("v ")){
            line.splitter(" ").array.remove(0).each!((e) { vertices~= parse!float(e);});
            writeln(line.splitter(" ").array);
        }
        else if(line.startsWith("vn ")){
            line.splitter(" ").array.remove(0).each!((e) { normals ~= parse!float(e);});
            writeln(line.splitter(" ").array);
        }
        else if(line.startsWith("f ")){
            auto face = line.splitter(" ").array.remove(0);
            foreach(indice; face){
                auto component = indice.splitter("/").array;
                if(component[0]!=""){
                    int idx = (parse!int(component[0]) - 1 ) * 3;
                    mVertexData~= [vertices[idx], vertices[idx+1], vertices[idx+2]];
                }
                if(component[2]!=""){
                    int idx= (parse!int(component[2]) - 1 ) * 3;
                    mVertexData ~= [normals[idx+0], normals[idx+1], normals[idx+2]];
                }
            }
        }
    }
```



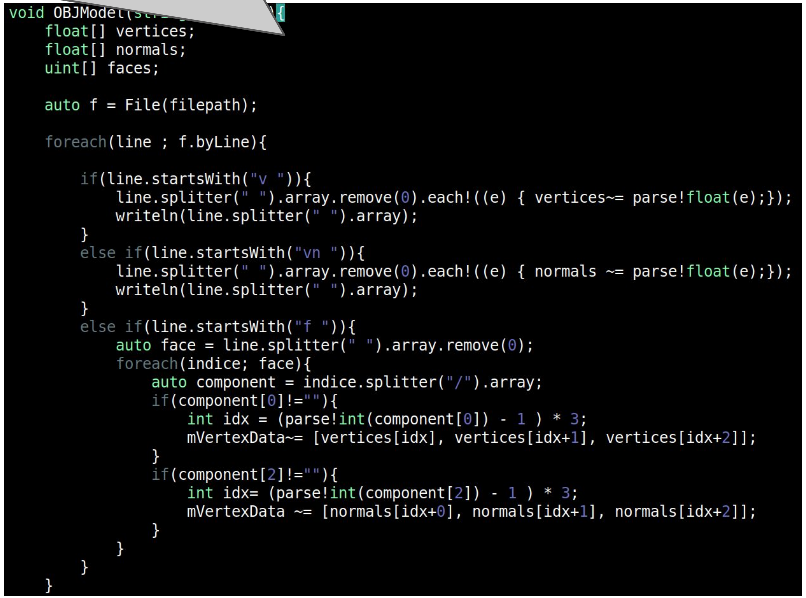


- On your own time you can zoom in and contrast the C++ (left) versus the D (right) code.
  - ○ When simple, both read about the same -- but as complexity goes up, the D code remains about the same complexity.

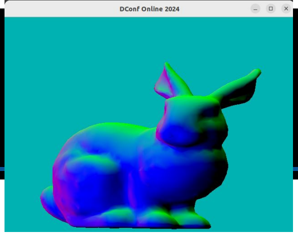
something more



```cpp
void Model::loadOBJ(){
    // 1.) Scan the data
    std::string line;
    std::ifstream myFile(fname.c_str());
    if(myFile.is_open()){
        while(getline(myFile,line)){
            if(line[0]=='f'){
                std::string temp = myutil::replaceString(line,"f ","");
                temp = myutil::replaceString(temp,"/","a");
                temp = myutil::replaceString(temp,"a"," ");
                std::vector<int> lst = myutil::vectorStringToInt(myutil::split(temp," "));
                // Create a face
                // Subtract 1 because obj's are 1's based
                triangleList.push_back((unsigned int)lst[0]-1);
                triangleList.push_back((unsigned int)lst[2]-1);
                triangleList.push_back((unsigned int)lst[4]-1);
            }
            else if(line[0]=='v'){
                if(line[1]=='n'){
                    std::vector<float> temp = myutil::vectorStringToFloat(myutil::split(line," "));
                    normalList.push_back(Normal(temp[0],temp[1],temp[2]));
                }else{
                    std::vector<float> temp = myutil::vectorStringToFloat(myutil::split(line," "));
                    vertexList.push_back((float)temp[0]);
                    vertexList.push_back((float)temp[1]);
                    vertexList.push_back((float)temp[2]);
                    // Also push in some colors
                    vertexList.push_back(0.9f);
                    vertexList.push_back(0.9f);
                    vertexList.push_back(0.9f);
                }
            }
```

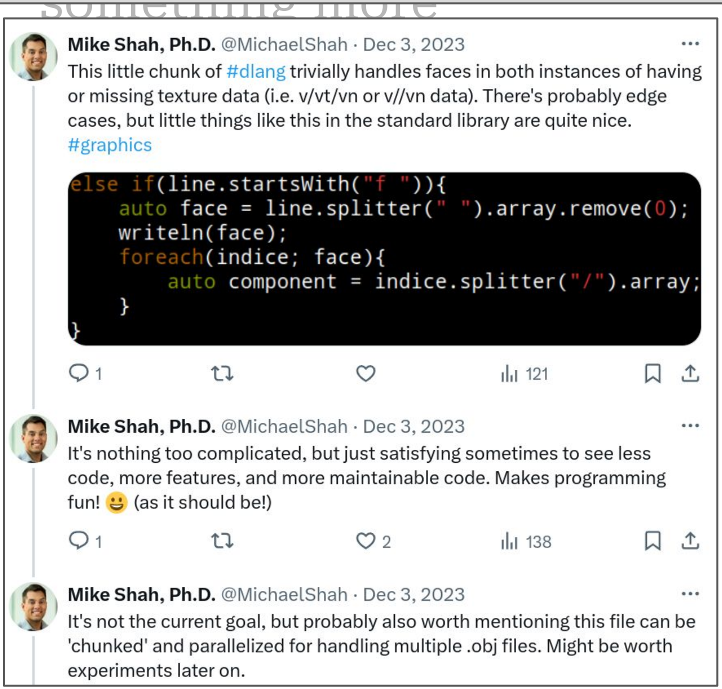


```d
void OBJModel(st  ){
    float[] vertices;
    float[] normals;
    uint[] faces;

    auto f = File(filepath);

    foreach(line ; f.byLine){

        if(line.startsWith("v ")){
            line.splitter(" ").array.remove(0).each!((e) { vertices~= parse!float(e);});
            writeln(line.splitter(" ").array);
        }
        else if(line.startsWith("vn ")){
            line.splitter(" ").array.remove(0).each!((e) { normals ~= parse!float(e);});
            writeln(line.splitter(" ").array);
        }
        else if(line.startsWith("f ")){
            auto face = line.splitter(" ").array.remove(0);
            foreach(indice; face){
                auto component = indice.splitter("/").array;
                if(component[0]!=""){
                    int idx = (parse!int(component[0]) - 1 ) * 3;
                    mVertexData~= [vertices[idx], vertices[idx+1], vertices[idx+2]];
                }
                if(component[2]!=""){
                    int idx= (parse!int(component[2]) - 1 ) * 3;
                    mVertexData ~= [normals[idx+0], normals[idx+1], normals[idx+2]];
                }
            }
        }
    }
```



- It remains a future experiment -- but I think with D's built-in concurrency (std.concurrency) I could probably speed this up quite a bit.
    - It's an open challenge to myself (and anyone else) to see if you can build the fastest .obj parser.

something more



Mike Shah, Ph.D. @MichaelShah · Dec 3, 2023

This little chunk of #dlang trivially handles faces in both instances of having or missing texture data (i.e. v/vt/vn or v//vn data). There's probably edge cases, but little things like this in the standard library are quite nice. #graphics

```
else if(line.startsWith("f ")){
    auto face = line.splitter(" ").array.remove(0);
    writeln(face);
    foreach(indice; face){
        auto component = indice.splitter("/").array;
    }
}
```

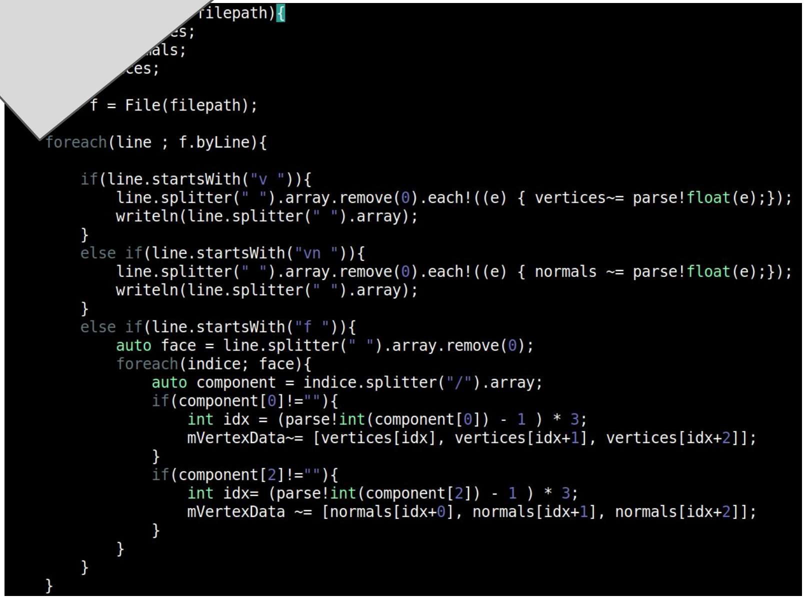
Mike Shah, Ph.D. @MichaelShah · Dec 3, 2023

It's nothing too complicated, but just satisfying sometimes to see less code, more features, and more maintainable code. Makes programming fun! 😃 (as it should be!)

Mike Shah, Ph.D. @MichaelShah · Dec 3, 2023

It's not the current goal, but probably also worth mentioning this file can be 'chunked' and parallelized for handling multiple .obj files. Might be worth experiments later on.

RIP https://twitter.com/MichaelShah/status/1731522845191057919

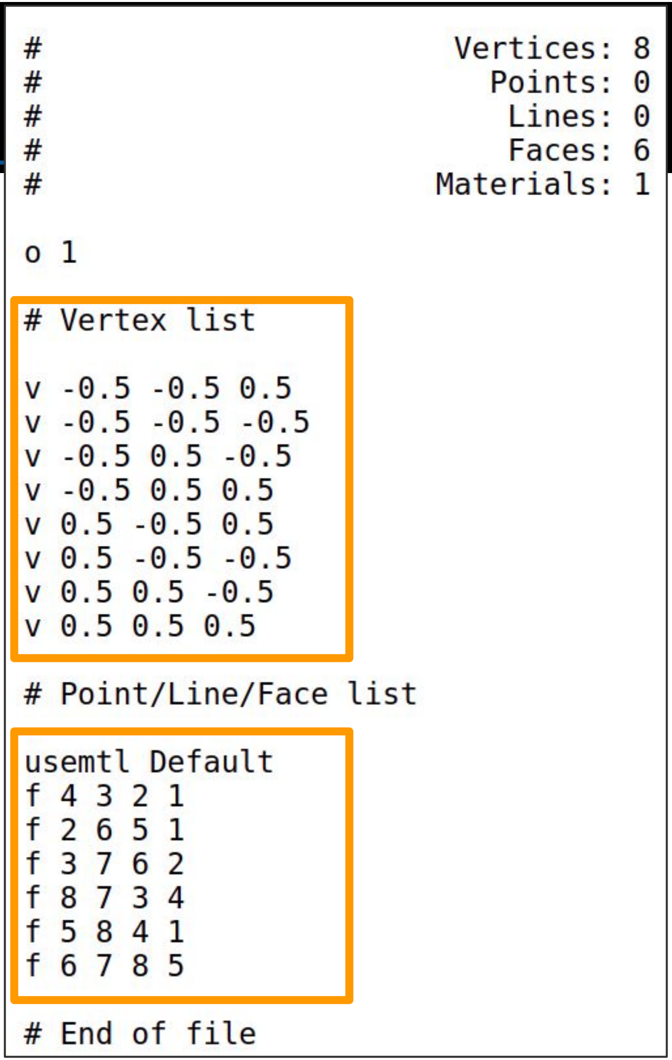


```
              filepath){
            es;
           als;
          ces;

     f = File(filepath);

    foreach(line ; f.byLine){

        if(line.startsWith("v ")){
            line.splitter(" ").array.remove(0).each!((e) { vertices~= parse!float(e);});
            writeln(line.splitter(" ").array);
        }
        else if(line.startsWith("vn ")){
            line.splitter(" ").array.remove(0).each!((e) { normals ~= parse!float(e);});
            writeln(line.splitter(" ").array);
        }
        else if(line.startsWith("f ")){
            auto face = line.splitter(" ").array.remove(0);
            foreach(indice; face){
                auto component = indice.splitter("/").array;
                if(component[0]!=""){
                    int idx = (parse!int(component[0]) - 1 ) * 3;
                    mVertexData~= [vertices[idx], vertices[idx+1], vertices[idx+2]];
                }
                if(component[2]!=""){
                    int idx= (parse!int(component[2]) - 1 ) * 3;
                    mVertexData ~= [normals[idx+0], normals[idx+1], normals[idx+2]];
                }
            }
        }
    }
```

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

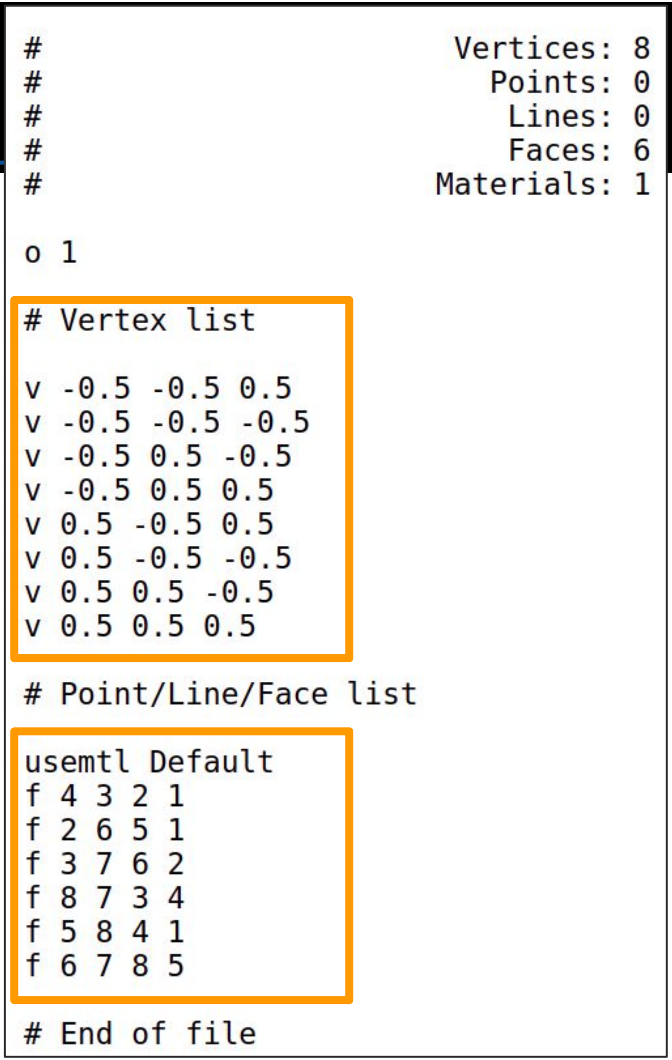
See DConf 2024 Online talk for how short the parsing code can be!



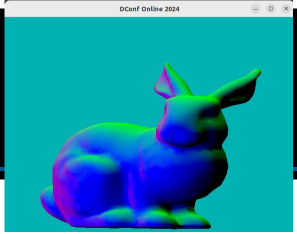
```
#                          Vertices: 8
#                            Points: 0
#                             Lines: 0
#                             Faces: 6
#                         Materials: 1

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
```

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.



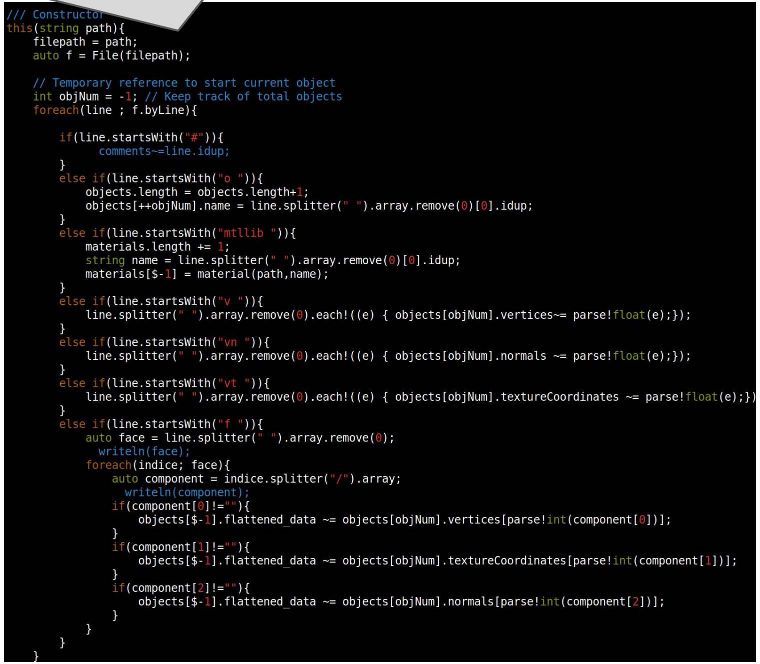
```
#                              Vertices: 8
#                                Points: 0
#                                 Lines: 0
#                                 Faces: 6
#                             Materials: 1

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
```



- Anyways... with a little bit more code, I was able to extend my parser to handle .obj files that contain multiple models and materials.
  - A mix of functional and object-oriented paradigms made this quite nice!

something more interesting than triangles, we will load that data from a file.

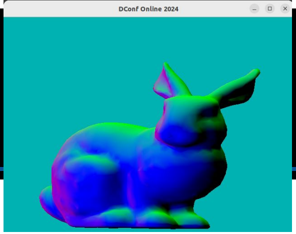
- To the right -- is the entire parser for the .obj file.



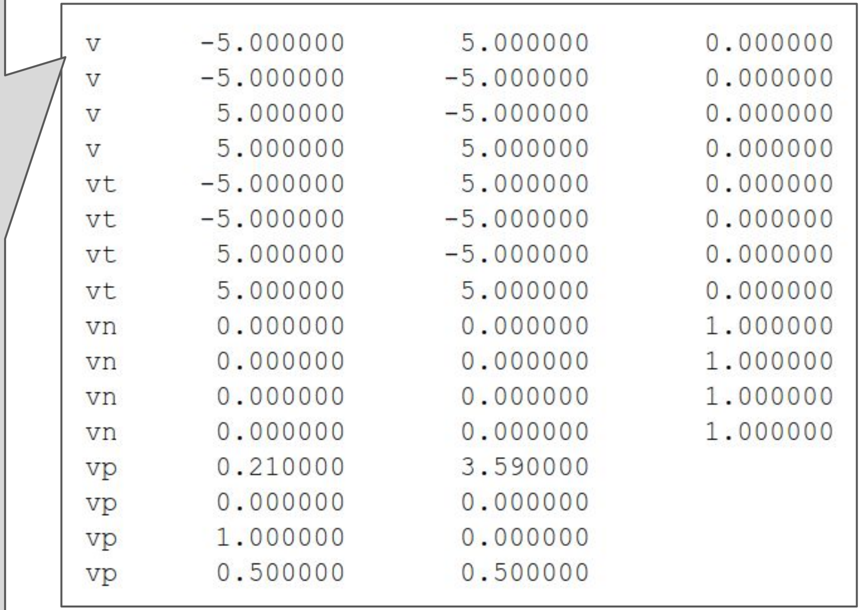
```
/// Constructor
this(string path){
    filepath = path;
    auto f = File(filepath);

    // Temporary reference to start current object
    int objNum = -1; // Keep track of total objects
    foreach(line ; f.byLine){

        if(line.startsWith("#")){
              comments~=line.idup;
        }
        else if(line.startsWith("o ")){
            objects.length = objects.length+1;
            objects[++objNum].name = line.splitter(" ").array.remove(0)[0].idup;
        }
        else if(line.startsWith("mtllib ")){
            materials.length += 1;
            string name = line.splitter(" ").array.remove(0)[0].idup;
            materials[$-1] = material(path,name);
        }
        else if(line.startsWith("v ")){
            line.splitter(" ").array.remove(0).each!((e) { objects[objNum].vertices~= parse!float(e);});
        }
        else if(line.startsWith("vn ")){
            line.splitter(" ").array.remove(0).each!((e) { objects[objNum].normals ~= parse!float(e);});
        }
        else if(line.startsWith("vt ")){
            line.splitter(" ").array.remove(0).each!((e) { objects[objNum].textureCoordinates ~= parse!float(e);})
        }
        else if(line.startsWith("f ")){
            auto face = line.splitter(" ").array.remove(0);
              writeln(face);
            foreach(indice; face){
                auto component = indice.splitter("/").array;
                  writeln(component);
                if(component[0]!=""){
                    objects[$-1].flattened_data ~= objects[objNum].vertices[parse!int(component[0])];
                }
                if(component[1]!=""){
                    objects[$-1].flattened_data ~= objects[objNum].textureCoordinates[parse!int(component[1])];
                }
                if(component[2]!=""){
                    objects[$-1].flattened_data ~= objects[objNum].normals[parse!int(component[2])];
                }
            }
        }
    }
```

https://twitter.com/MichaelShah/status/1731522845191057919

Parsing Structured Data

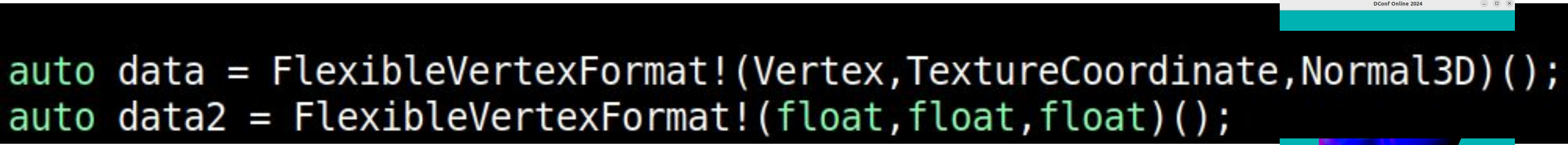


- The other thing to note -- is that complexity often arises with the many variations of 3D data.
  - A 3D model can contain vertices or a number of other attributes such as texture coordinates, vertex normals, or other primitives.

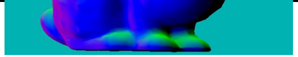


```
v       -5.000000      5.000000      0.000000
v       -5.000000     -5.000000      0.000000
v        5.000000     -5.000000      0.000000
v        5.000000      5.000000      0.000000
vt      -5.000000      5.000000      0.000000
vt      -5.000000     -5.000000      0.000000
vt       5.000000     -5.000000      0.000000
vt       5.000000      5.000000      0.000000
vn       0.000000      0.000000      1.000000
vn       0.000000      0.000000      1.000000
vn       0.000000      0.000000      1.000000
vn       0.000000      0.000000      1.000000
vp       0.210000      3.590000
vp       0.000000      0.000000
vp       1.000000      0.000000
vp       0.500000      0.500000
```

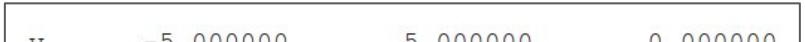
https://paulbourke.net/dataformats/obj/



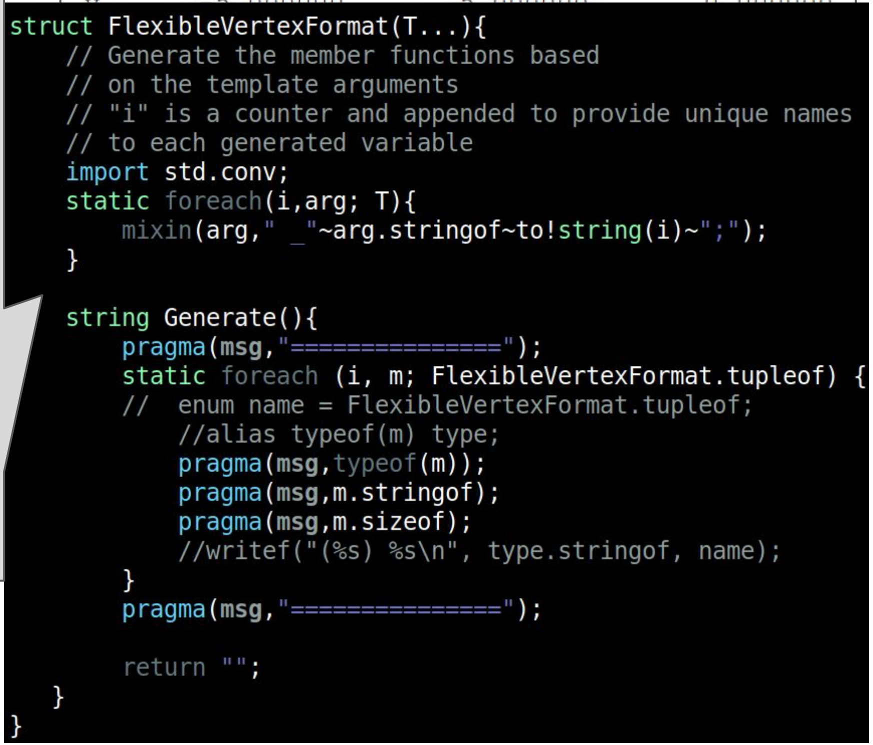
```
auto data = FlexibleVertexFormat!(Vertex,TextureCoordinate,Normal3D)();
auto data2 = FlexibleVertexFormat!(float,float,float)();
```



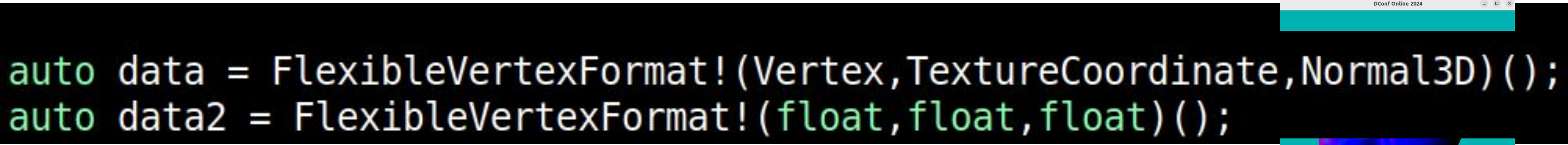
- With D’s metaprogramming capabilities, you can generate the variations you need for your geometry data.



```
V      5.000000      5.000000      0.000000
```

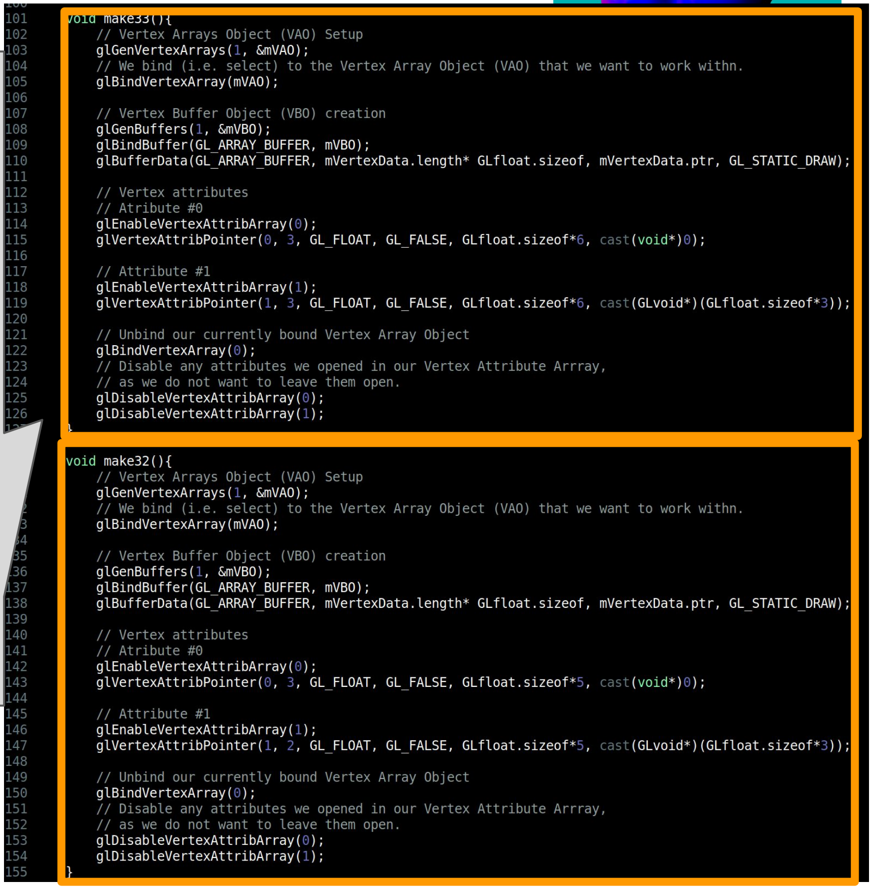


```
struct FlexibleVertexFormat(T...){
    // Generate the member functions based
    // on the template arguments
    // "i" is a counter and appended to provide unique names
    // to each generated variable
    import std.conv;
    static foreach(i,arg; T){
        mixin(arg," _"~arg.stringof~to!string(i)~";");
    }

    string Generate(){
        pragma(msg,"===============");
        static foreach (i, m; FlexibleVertexFormat.tupleof) {
        //  enum name = FlexibleVertexFormat.tupleof;
            //alias typeof(m) type;
            pragma(msg,typeof(m));
            pragma(msg,m.stringof);
            pragma(msg,m.sizeof);
            //writef("(%s) %s\n", type.stringof, name);
        }
        pragma(msg,"===============");

        return "";
   }
}
```



```
auto data = FlexibleVertexFormat!(Vertex,TextureCoordinate,Normal3D)();
auto data2 = FlexibleVertexFormat!(float,float,float)();
```

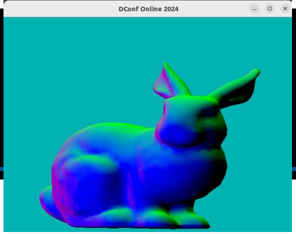
- With D’s metaprogramming capabilities, you can generate the variations you need for your geometry data.
  - This could also include setting up the various layouts needed for passing data to OpenGL
  - Observe the the right two different layouts
    - Why write this error prone boilerplate, when we could otherwise generate it?



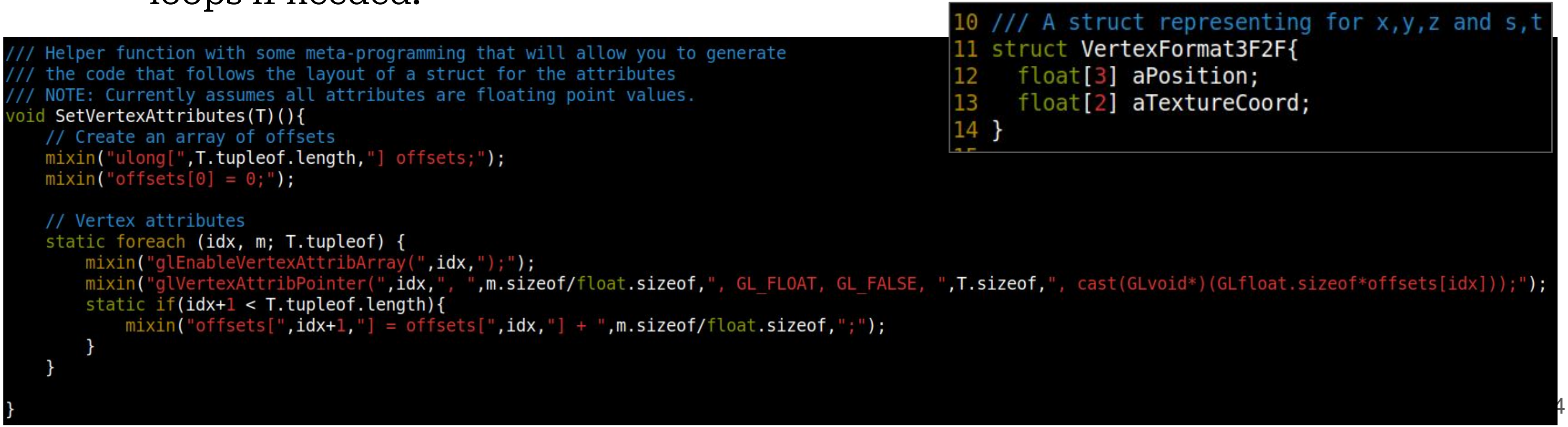
```
void make33(){
    // Vertex Arrays Object (VAO) Setup
    glGenVertexArrays(1, &mVAO);
    // We bind (i.e. select) to the Vertex Array Object (VAO) that we want to work withn.
    glBindVertexArray(mVAO);

    // Vertex Buffer Object (VBO) creation
    glGenBuffers(1, &mVBO);
    glBindBuffer(GL_ARRAY_BUFFER, mVBO);
    glBufferData(GL_ARRAY_BUFFER, mVertexData.length* GLfloat.sizeof, mVertexData.ptr, GL_STATIC_DRAW);

    // Vertex attributes
    // Atribute #0
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, GLfloat.sizeof*6, cast(void*)0);

    // Attribute #1
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, GLfloat.sizeof*6, cast(GLvoid*)(GLfloat.sizeof*3));

    // Unbind our currently bound Vertex Array Object
    glBindVertexArray(0);
    // Disable any attributes we opened in our Vertex Attribute Arrray,
    // as we do not want to leave them open.
    glDisableVertexAttribArray(0);
    glDisableVertexAttribArray(1);
}
```

```
void make32(){
    // Vertex Arrays Object (VAO) Setup
    glGenVertexArrays(1, &mVAO);
    // We bind (i.e. select) to the Vertex Array Object (VAO) that we want to work withn.
    glBindVertexArray(mVAO);

    // Vertex Buffer Object (VBO) creation
    glGenBuffers(1, &mVBO);
    glBindBuffer(GL_ARRAY_BUFFER, mVBO);
    glBufferData(GL_ARRAY_BUFFER, mVertexData.length* GLfloat.sizeof, mVertexData.ptr, GL_STATIC_DRAW);

    // Vertex attributes
    // Atribute #0
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, GLfloat.sizeof*5, cast(void*)0);

    // Attribute #1
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, GLfloat.sizeof*5, cast(GLvoid*)(GLfloat.sizeof*3));

    // Unbind our currently bound Vertex Array Object
    glBindVertexArray(0);
    // Disable any attributes we opened in our Vertex Attribute Arrray,
    // as we do not want to leave them open.
    glDisableVertexAttribArray(0);
    glDisableVertexAttribArray(1);
}
```

Parsing Structured Data



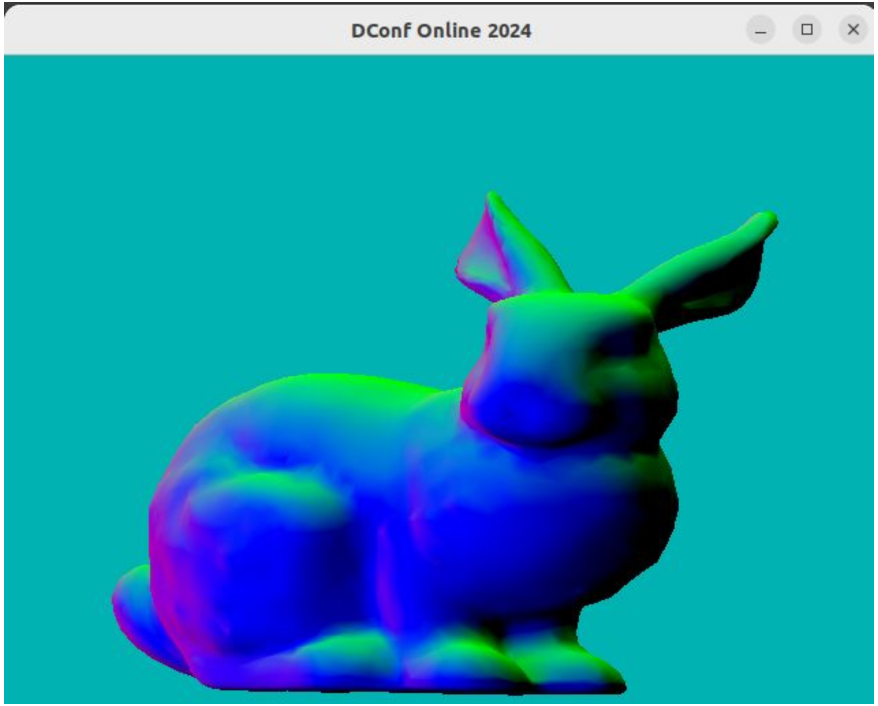
- Here is an example of using a ‘SetVertexAttributes’ that is templated and builds the code based off of a struct passed in.
- This generates the correct layout for a mesh given:
  - `SetVertexAttributes!VertexFormat3F2F();`
- Pro Tip: Don’t be afraid to introduce a new ‘scope’ with {}’s in your static foreach loops if needed.



```
/// A struct representing for x,y,z and s,t
struct VertexFormat3F2F{
   float[3] aPosition;
   float[2] aTextureCoord;
}
```

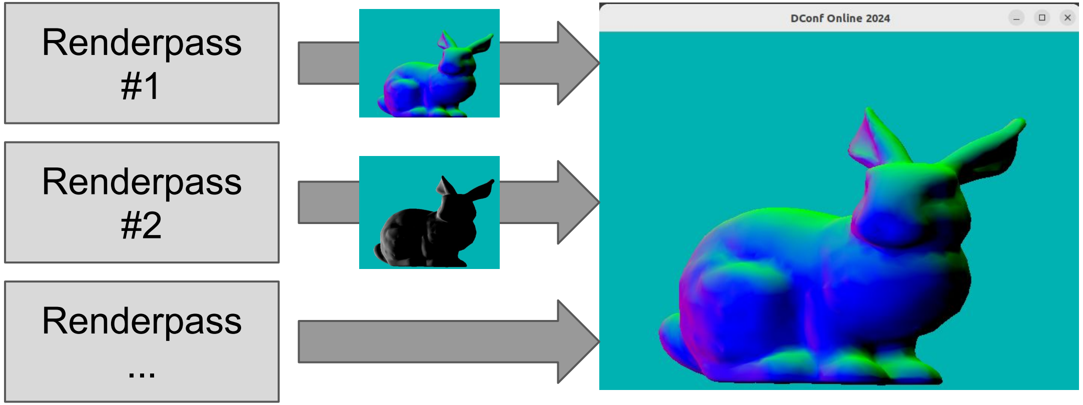
```
/// Helper function with some meta-programming that will allow you to generate
/// the code that follows the layout of a struct for the attributes
/// NOTE: Currently assumes all attributes are floating point values.
void SetVertexAttributes(T)(){
    // Create an array of offsets
    mixin("ulong[",T.tupleof.length,"] offsets;");
    mixin("offsets[0] = 0;");

    // Vertex attributes
    static foreach (idx, m; T.tupleof) {
        mixin("glEnableVertexAttribArray(",idx,");");
        mixin("glVertexAttribPointer(",idx,", ",m.sizeof/float.sizeof,", GL_FLOAT, GL_FALSE, ",T.sizeof,", cast(GLvoid*)(GLfloat.sizeof*offsets[idx]));");
        static if(idx+1 < T.tupleof.length){
            mixin("offsets[",idx+1,"] = offsets[",idx,"] + ",m.sizeof/float.sizeof,";");
        }
    }

}
```

Demo 3
Render Targets



Multiple Render Targets (1/2)

- What the acute watcher will observe is that the last two demos are almost exactly the same
  - The difference is that this final demo renders to an offscreen texture, before rendering the object



Renderpass #1

Renderpass #2

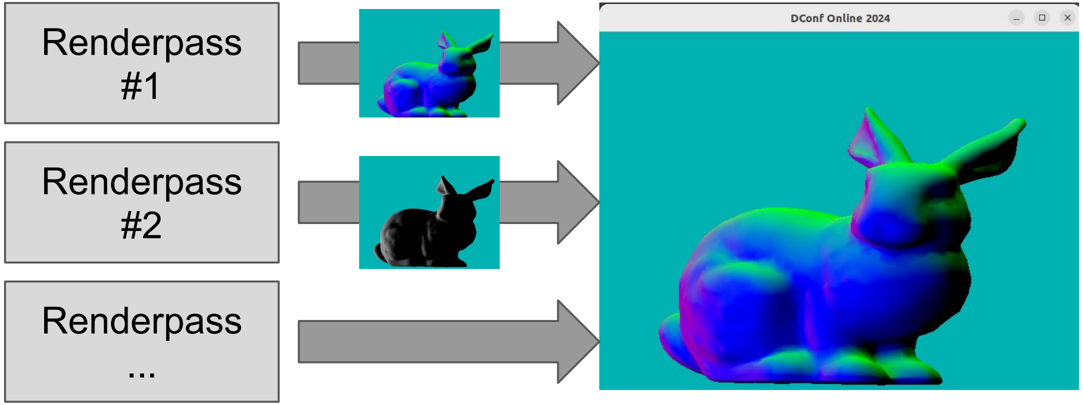
Renderpass …

Final image is composed of the ‘data’ from other intermediate renderings.

Often we defer expensive calculations to the end to only compute them once (e.g. deferred rendering)

Multiple Render Targets (2/2)

- There is actually nothing D specific here -- this is just a function of the API
- And that’s exactly my point -- if you’ve seen it done in other languages with graphics APIs, you can do the same work with D, and take advantage of D’s productivity.



Renderpass #1

Renderpass #2

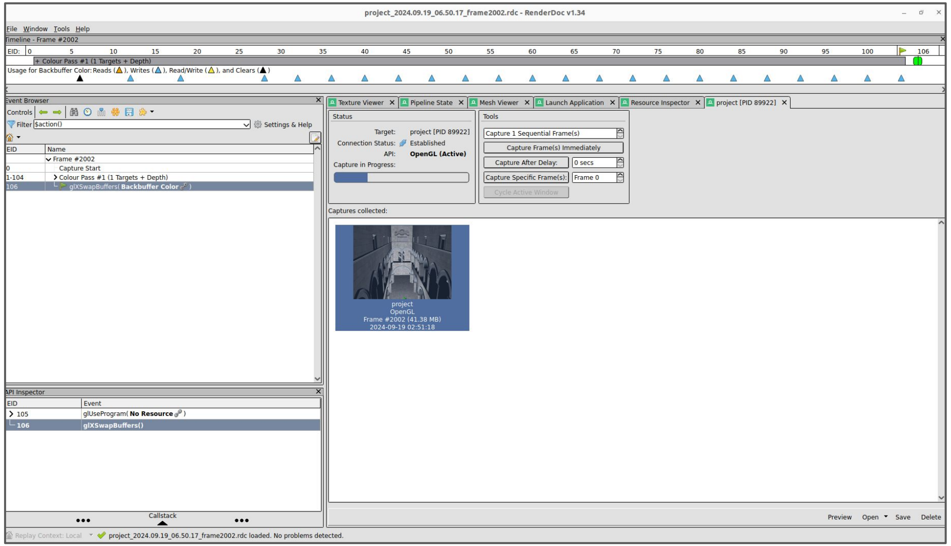
Renderpass ...

Final image is composed of the ‘data’ from other intermediate renderings.

Often we defer expensive calculations to the end to only compute them once (e.g. deferred rendering)

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.)



project_2024.09.19_06.50.17_frame2002.rdc - RenderDoc v1.34

Timeline - Frame #2002

EID: 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 106

+ Colour Pass #1 (1 Targets + Depth)

Usage for Backbuffer Color: Reads (△), Writes (△), Read/Write (△), and Clears (▲)

Event Browser

Filter $action()

EID	Name
	Frame #2002
0	Capture Start
1-104	Colour Pass #1 (1 Targets + Depth)
106	glXSwapBuffers(**Backbuffer Color**)

Status

Target: project [PID 89922]
Connection Status: Established
API: **OpenGL (Active)**
Capture in Progress:

Tools

Capture 1 Sequential Frame(s)
Capture After Delay: 0 secs
Capture Specific Frame(s): Frame 0

Captures collected:

project
OpenGL
Frame #2002 (41.38 MB)
2024-09-19 02:51:18

API Inspector

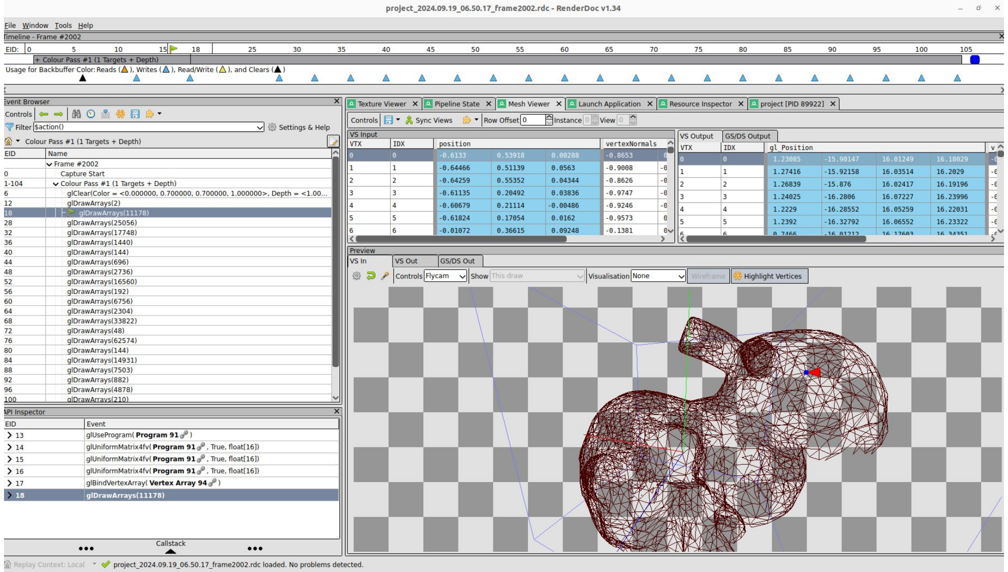
EID	Event
105	glUseProgram(**No Resource**)
106	**glXSwapBuffers()**

Replay Context: Local ✓ project_2024.09.19_06.50.17_frame2002.rdc loaded. No problems detected.

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.



project_2024.09.19_06.50.17_frame2002.rdc - RenderDoc v1.34

Filter $action()

Colour Pass #1 (1 Targets + Depth)

EID	Name
	Frame #2002
0	Capture Start
1-104	Colour Pass #1 (1 Targets + Depth)
6	glClear(Color = <0.000000, 0.700000, 0.700000, 1.000000>, Depth = <1.00...
12	glDrawArrays(2)
18	glDrawArrays(11178)
28	glDrawArrays(25056)
32	glDrawArrays(17748)
36	glDrawArrays(1440)
40	glDrawArrays(144)
44	glDrawArrays(696)
48	glDrawArrays(2736)
52	glDrawArrays(16560)
56	glDrawArrays(192)
60	glDrawArrays(6756)
64	glDrawArrays(2304)
68	glDrawArrays(33822)
72	glDrawArrays(48)
76	glDrawArrays(62574)
80	glDrawArrays(144)
84	glDrawArrays(14931)
88	glDrawArrays(7503)
92	glDrawArrays(882)
96	glDrawArrays(4878)
100	glDrawArrays(210)

API Inspector

EID	Event
13	glUseProgram(**Program 91**)
14	glUniformMatrix4fv(**Program 91** , True, float[16])
15	glUniformMatrix4fv(**Program 91** , True, float[16])
16	glUniformMatrix4fv(**Program 91** , True, float[16])
17	glBindVertexArray(**Vertex Array 94**)
18	**glDrawArrays(11178)**

Row Offset 0 Instance 0 View 0

VS Input

VTX	IDX	position			vertexNormals
0	0	-0.6133	0.53918	0.00288	-0.8653
1	1	-0.64466	0.51139	0.0563	-0.9008
2	2	-0.64259	0.55352	0.04344	-0.8626
3	3	-0.61135	0.20492	0.03836	-0.9747
4	4	-0.60679	0.21114	-0.00486	-0.9246
5	5	-0.61824	0.17054	0.0162	-0.9573
6	6	-0.01072	0.36615	0.09248	-0.1381

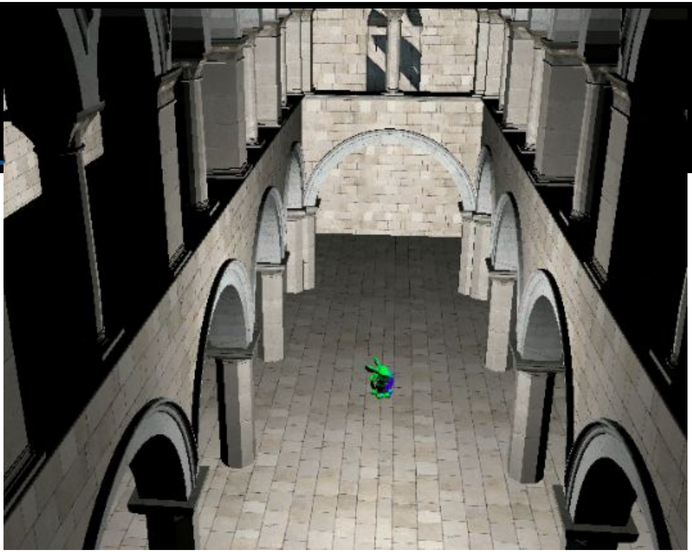
VS Output | GS/DS Output

VTX	IDX	gl_Position			
0	0	1.23085	-15.90147	16.01249	16.18029
1	1	1.27416	-15.92158	16.03514	16.2029
2	2	1.26839	-15.876	16.02417	16.19196
3	3	1.24025	-16.2806	16.07227	16.23996
4	4	1.2229	-16.28552	16.05259	16.22031
5	5	1.2392	-16.32792	16.06552	16.23322
6	6	0.7466	-16.01212	16.17603	16.34351

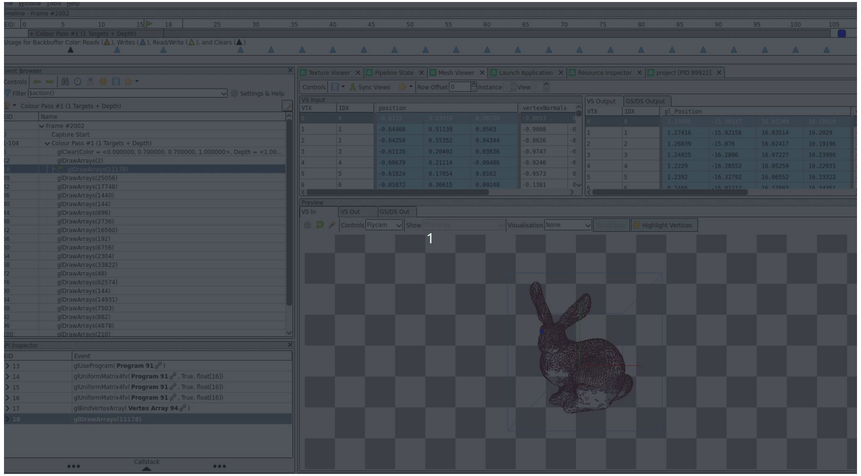
Controls Flycam Show This draw Visualisation None

Replay Context: Local ✓ project_2024.09.19_06.50.17_frame2002.rdc loaded. No problems detected.

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**



Filter: $action()

Colour Pass #1 (1 Targets + Depth)

EID	Name
	Frame #2002
0	Capture Start
1-104	Colour Pass #1 (1 Targets + Depth)
5	glClear(Color = <0.000000, 0.700000, 0.700000, 1.000000>, Depth = <1.00...
12	glDrawArrays(2)
18	glDrawArrays(11178)
28	glDrawArrays(25056)
32	glDrawArrays(17748)
36	glDrawArrays(1440)
40	glDrawArrays(144)
44	glDrawArrays(696)
48	glDrawArrays(2736)
52	glDrawArrays(16560)
56	glDrawArrays(192)
60	glDrawArrays(6756)
64	glDrawArrays(2304)
68	glDrawArrays(33822)
72	glDrawArrays(48)
76	glDrawArrays(62574)
80	glDrawArrays(144)
84	glDrawArrays(14931)
88	glDrawArrays(7503)
92	glDrawArrays(882)
96	glDrawArrays(4878)
100	glDrawArrays(210)

API Inspector

EID	Event
13	glUseProgram(**Program 91**)
14	glUniformMatrix4fv(**Program 91** , True, float[16])
15	glUniformMatrix4fv(**Program 91** , True, float[16])
16	glUniformMatrix4fv(**Program 91** , True, float[16])
17	glBindVertexArray(**Vertex Array 94**)
18	glDrawArrays(11178)

Row Offset: 0

VS Input

VTX	IDX	position			vertexNormals	
0	0	-0.6133	0.53918	0.00288	-0.8653	0
1	1	-0.64466	0.51139	0.0563	-0.9008	-0
2	2	-0.64259	0.55352	0.04344	-0.8626	-0
3	3	-0.61135	0.20492	0.03836	-0.9747	-0
4	4	-0.60679	0.21114	-0.00486	-0.9246	-0
5	5	-0.61824	0.17054	0.0162	-0.9573	0
6	6	-0.01072	0.36615	0.09248	-0.1381	0

VS Output | GS/DS Output

VTX	IDX	gl_Position			
0	0	1.23086	-15.90147	16.01249	16.18829
1	1	1.27416	-15.92158	16.03514	16.2029
2	2	1.26839	-15.876	16.02417	16.19196
3	3	1.24025	-16.2806	16.07227	16.23996
4	4	1.2229	-16.28552	16.05259	16.22031
5	5	1.2392	-16.32792	16.06552	16.23322
6	6	0.7466	-16.01212	16.17603	16.34351

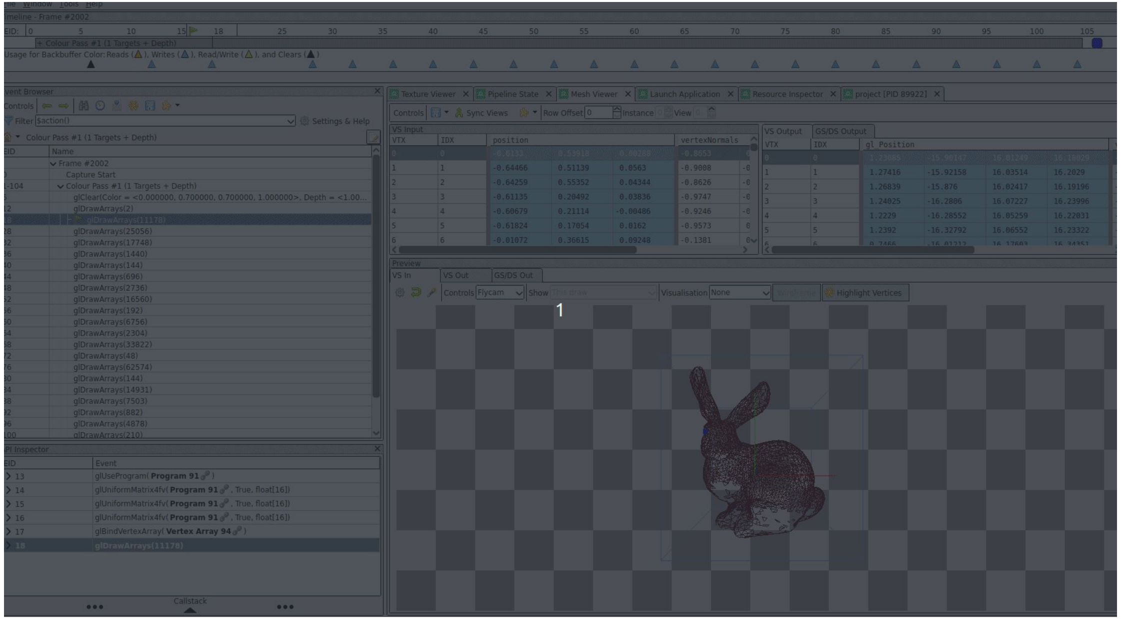
Controls: Flycam

Show: This Draw

Visualisation: None

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.



EID	Name
	Frame #2002
0	Capture Start
1-104	Colour Pass #1 (1 Targets + Depth)
5	glClear(Color = <0.000000, 0.700000, 0.700000, 1.000000>, Depth = <1.00...
12	glDrawArrays(2)
18	glDrawArrays(11178)
28	glDrawArrays(25056)
32	glDrawArrays(17748)
36	glDrawArrays(1440)
40	glDrawArrays(144)
44	glDrawArrays(696)
48	glDrawArrays(2736)
52	glDrawArrays(16560)
56	glDrawArrays(192)
60	glDrawArrays(6756)
64	glDrawArrays(2304)
68	glDrawArrays(33822)
72	glDrawArrays(48)
76	glDrawArrays(62574)
80	glDrawArrays(144)
84	glDrawArrays(14931)
88	glDrawArrays(7503)
92	glDrawArrays(882)
96	glDrawArrays(4878)
100	glDrawArrays(210)

EID	Event
13	glUseProgram(**Program 91**)
14	glUniformMatrix4fv(**Program 91** , True, float[16])
15	glUniformMatrix4fv(**Program 91** , True, float[16])
16	glUniformMatrix4fv(**Program 91** , True, float[16])
17	glBindVertexArray(**Vertex Array 94**)
18	glDrawArrays(11178)

VTX	IDX	position			vertexNormals	
0	0	-0.6133	0.53918	0.00288	-0.8653	0
1	1	-0.64466	0.51139	0.0563	-0.9008	-0
2	2	-0.64259	0.55352	0.04344	-0.8626	-0
3	3	-0.61135	0.20492	0.03836	-0.9747	-0
4	4	-0.60679	0.21114	-0.00486	-0.9246	-0
5	5	-0.61824	0.17054	0.0162	-0.9573	0
6	6	-0.01072	0.36615	0.09248	-0.1381	0

VTX	IDX	gl_Position			
0	0	1.23085	-15.90147	16.01249	16.18029
1	1	1.27416	-15.92158	16.03514	16.2029
2	2	1.26839	-15.876	16.02417	16.19196
3	3	1.24025	-16.2806	16.07227	16.23996
4	4	1.2229	-16.28552	16.05259	16.22031
5	5	1.2392	-16.32792	16.06552	16.23322
6	6	0.7466	-16.01212	16.17603	16.34351

Demo 4
Graphics Engine

Quick Demo of some objects

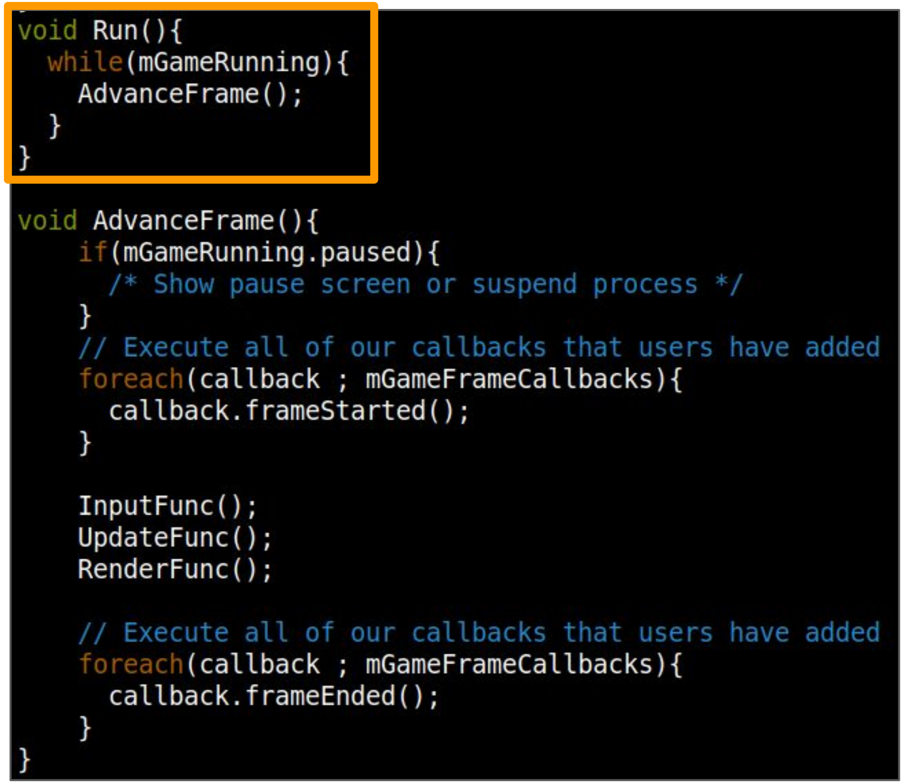
- So here’s a little capture of a scene I’m working on
  - There's A little bit of lighting, and a few models loaded, and about 260,000 triangles to draw the scene.
    - It’s a purposefully ‘unoptimized set of art assets’ to stress the system
  - This is the classic ‘Sponza’ scene used in graphics with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

Graphics Engine Design

Working Backwards a Bit

Structure of a Game (1/2)

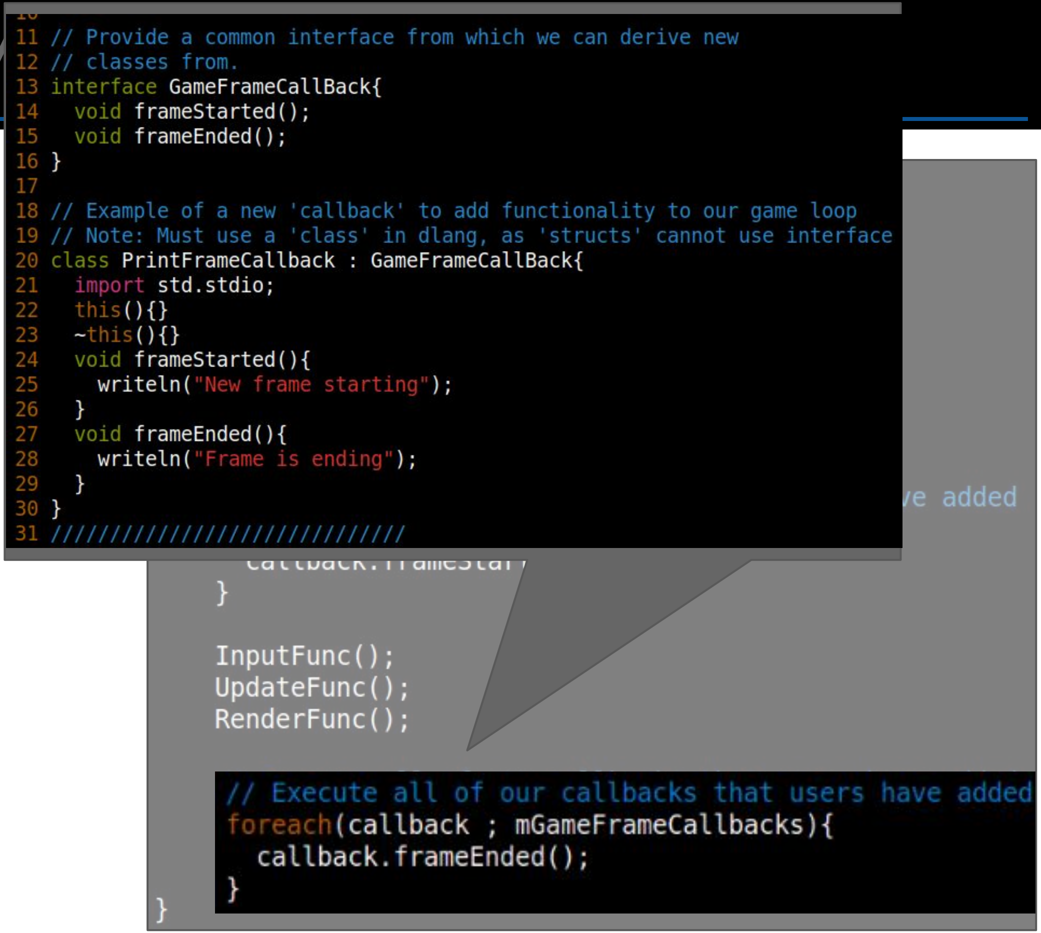
- So moving away from the graphics stuff for a moment, the infrastructure for these projects is pretty neat at the ‘core game loop’
  - Basically it’s just an input/update/render function
  - I like to separate that out to another function (`AdvanceFrame()`) for more control



```
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

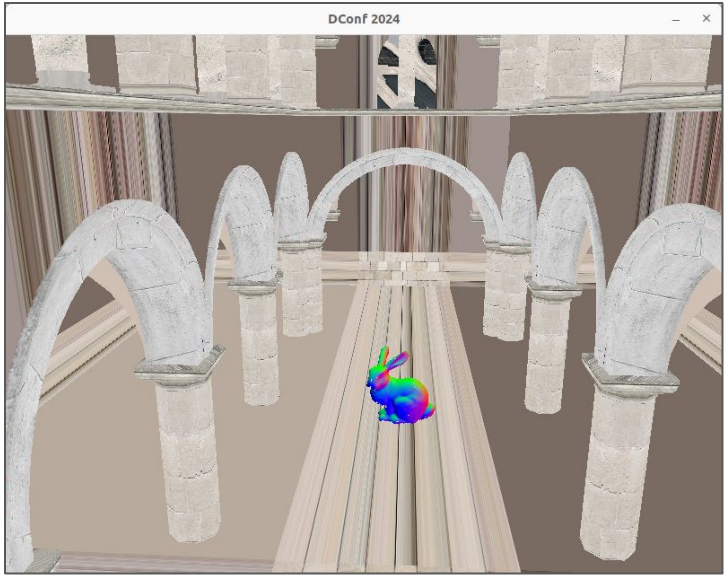


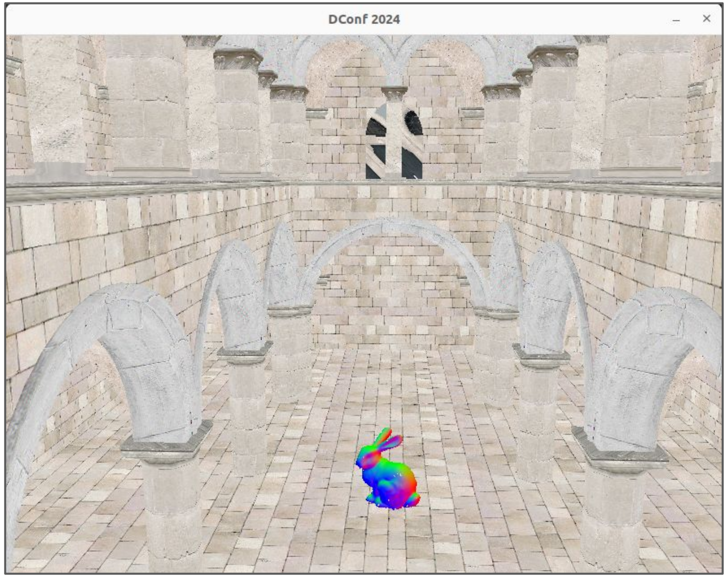
```
// Provide a common interface from which we can derive new
// classes from.
interface GameFrameCallBack{
  void frameStarted();
  void frameEnded();
}

// Example of a new 'callback' to add functionality to our game loop
// Note: Must use a 'class' in dlang, as 'structs' cannot use interface
class PrintFrameCallback : GameFrameCallBack{
  import std.stdio;
  this(){}
  ~this(){}
  void frameStarted(){
    writeln("New frame starting");
  }
  void frameEnded(){
    writeln("Frame is ending");
  }
}
//////////////////////////////
```

```
ve added
    }

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

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

Rapid Iteration Time Matters A lot (Blooper 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!



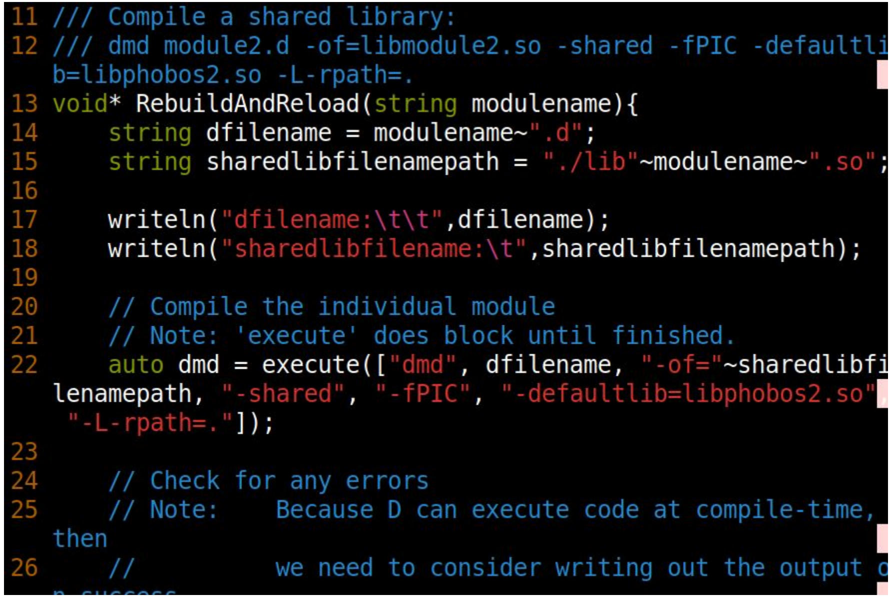


Hot Reloading Shaders (GPU)

- 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.



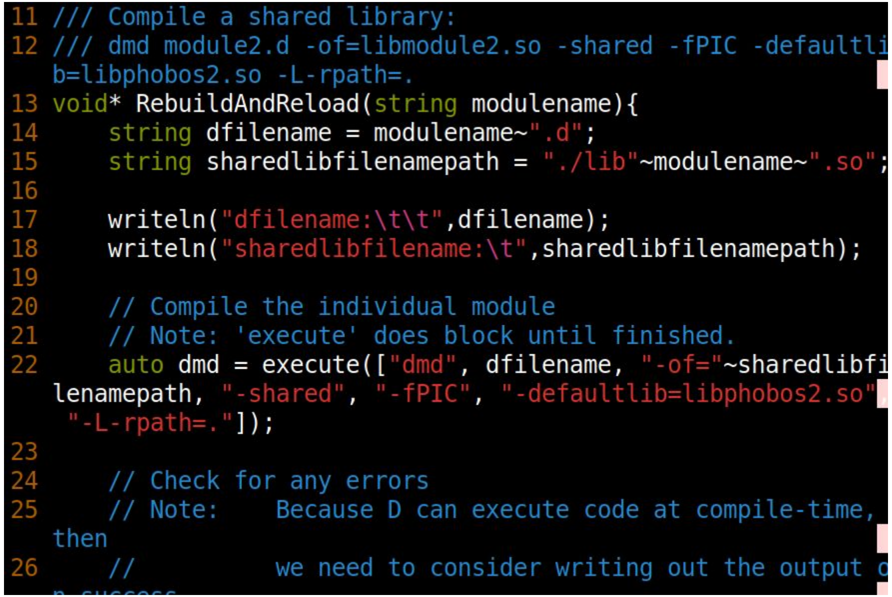
```
/// Compile a shared library:
/// dmd module2.d -of=libmodule2.so -shared -fPIC -defaultli
b=libphobos2.so -L-rpath=.
void* RebuildAndReload(string modulename){
    string dfilename = modulename~".d";
    string sharedlibfilenamepath = "./lib"~modulename~".so";

    writeln("dfilename:\t\t",dfilename);
    writeln("sharedlibfilename:\t",sharedlibfilenamepath);

    // Compile the individual module
    // Note: 'execute' does block until finished.
    auto dmd = execute(["dmd", dfilename, "-of="~sharedlibfi
lenamepath, "-shared", "-fPIC", "-defaultlib=libphobos2.so",
 "-L-rpath=."]);

    // Check for any errors
    // Note:    Because D can execute code at compile-time,
then
    //          we need to consider writing out the output o
n success
```

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)

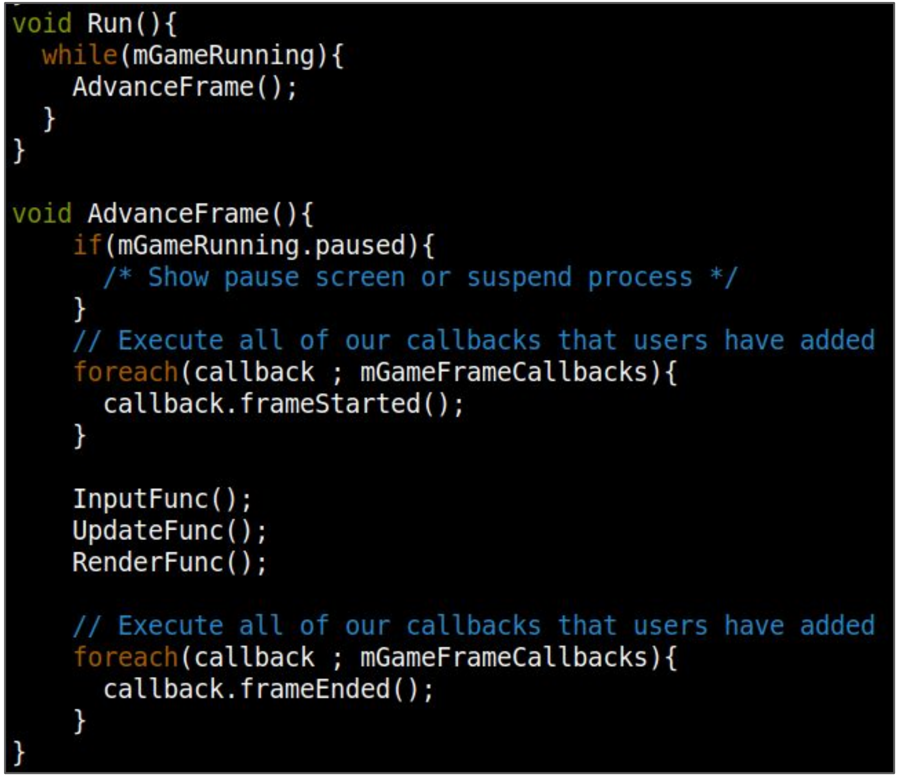
- So where this became handy was in the little ‘callback’ system I had
  - I could trigger a `RebuildAndReload` and add (or remove) callbacks to my system to change behavior without having to stop.
  - D’s compile times are more than fast enough for this small project -- but I like speed!
- I know there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



```d
/// Compile a shared library:
/// dmd module2.d -of=libmodule2.so -shared -fPIC -defaultlib=libphobos2.so -L-rpath=.
void* RebuildAndReload(string modulename){
    string dfilename = modulename~".d";
    string sharedlibfilenamepath = "./lib"~modulename~".so";

    writeln("dfilename:\t\t",dfilename);
    writeln("sharedlibfilename:\t",sharedlibfilenamepath);

    // Compile the individual module
    // Note: 'execute' does block until finished.
    auto dmd = execute(["dmd", dfilename, "-of="~sharedlibfilenamepath, "-shared", "-fPIC", "-defaultlib=libphobos2.so", "-L-rpath=."]);

    // Check for any errors
    // Note:    Because D can execute code at compile-time, then
    //          we need to consider writing out the output on success
```

unittests for games

- Depending on how you structure your game loop, you can push each update to a frame into some sort of queue structure.
  - Game/Graphics events can then be played in a unittest to simulate the game.
  - This is a good idea!
  - Languages that have built-in unittesting benefit quite a bit from this!

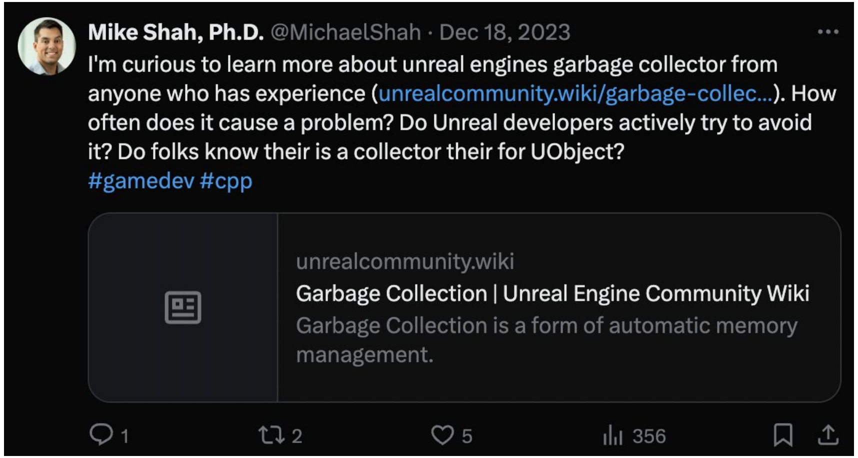


```
void Run(){
  while(mGameRunning){
    AdvanceFrame();
  }
}

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();
    }
}
```

Common Game / Graphics Concerns

Other “big things” - Dlang and GC

- People are afraid of Garbage Collection
    - But you get memory safety effectively for free.
    - Allocation is just as fast as with ‘new’ or ‘malloc’
        - The scan/pause is the part that probably needs work on the allocator.
    - You don’t have to use the garbage collector (as previously shown)
    - It looks like high powered C++ game engines have portions that are collected
        - Maybe someone can respond to my tweet (Is it a GC, reference counted, arena -- help me if you know!)
    - See more on dlang garbage collection: https://dlang.org/blog/the-gc-series/



Mike Shah, Ph.D. @MichaelShah · Dec 18, 2023

I'm curious to learn more about unreal engines garbage collector from anyone who has experience (unrealcommunity.wiki/garbage-collec...). How often does it cause a problem? Do Unreal developers actively try to avoid it? Do folks know their is a collector their for UObject? #gamedev #cpp

unrealcommunity.wiki

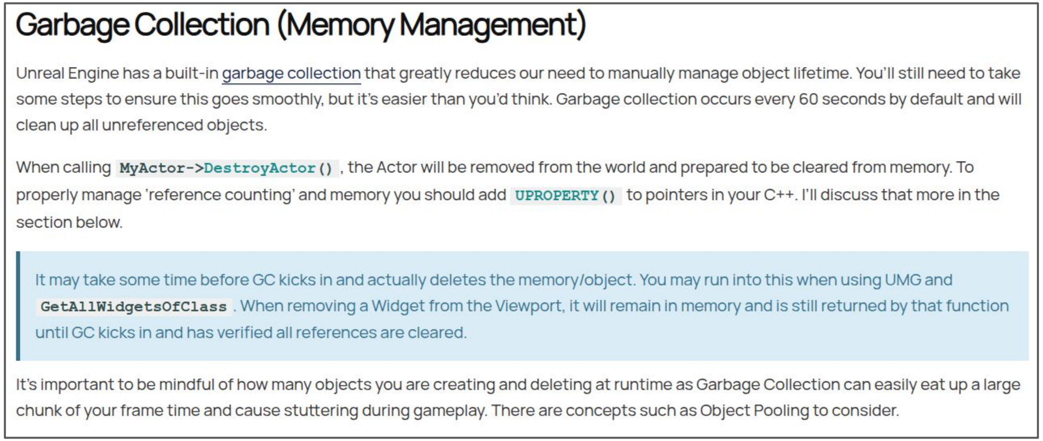
Garbage Collection | Unreal Engine Community Wiki

Garbage Collection is a form of automatic memory management.

https://twitter.com/MichaelShah/status/1736695501259415873

Garbage Collection in Unreal Engine

- https://www.tomlooman.com/unreal-engine-cpp-guide/
- It's amazing how many developers do not know that Unreal Engine has Garbage Collection available as a memory management strategy
  - Ideally, just make sure this does not happen in the 'hot' part of your code
    - (i.e. allocate everything ahead of time)



Garbage Collection (Memory Management)

Unreal Engine has a built-in garbage collection that greatly reduces our need to manually manage object lifetime. You'll still need to take some steps to ensure this goes smoothly, but it's easier than you'd think. Garbage collection occurs every 60 seconds by default and will clean up all unreferenced objects.

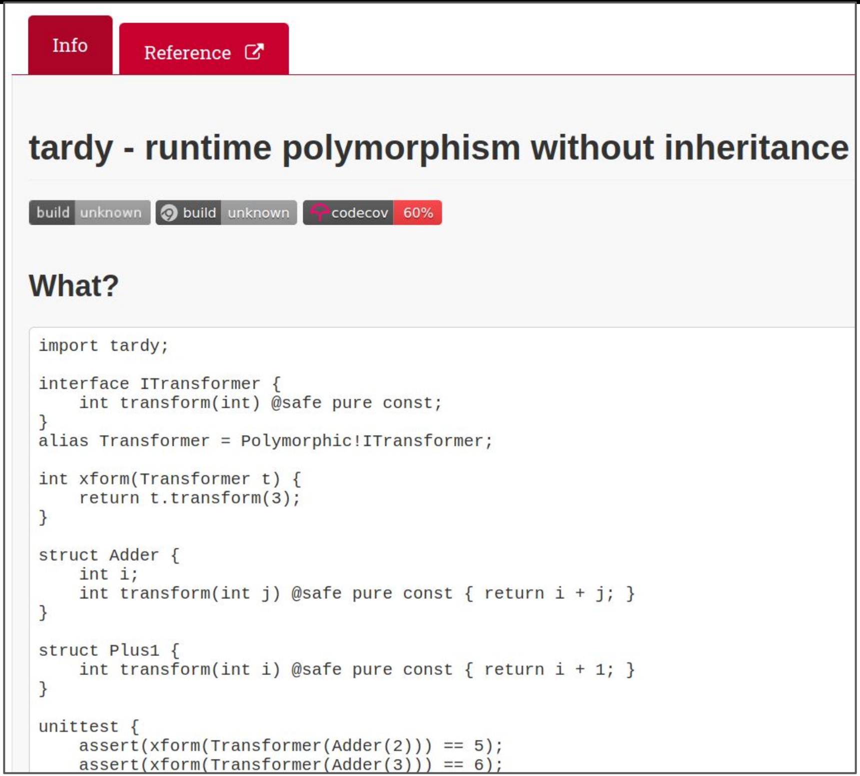
When calling `MyActor->DestroyActor()`, the Actor will be removed from the world and prepared to be cleared from memory. To properly manage 'reference counting' and memory you should add `UPROPERTY()` to pointers in your C++. I'll discuss that more in the section below.

> It may take some time before GC kicks in and actually deletes the memory/object. You may run into this when using UMG and `GetAllWidgetsOfClass`. When removing a Widget from the Viewport, it will remain in memory and is still returned by that function until GC kicks in and has verified all references are cleared.

It's important to be mindful of how many objects you are creating and deleting at runtime as Garbage Collection can easily eat up a large chunk of your frame time and cause stuttering during gameplay. There are concepts such as Object Pooling to consider.

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



tardy - runtime polymorphism without inheritance

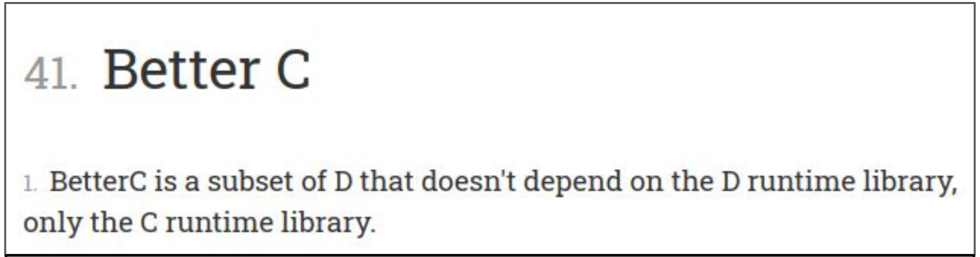
What?

```
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

betterC

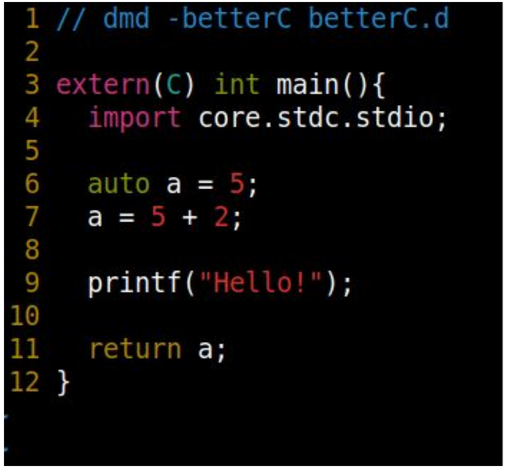
- If you don’t want the language run-time and standard library, you can use the ‘betterC’ mode to disable them.



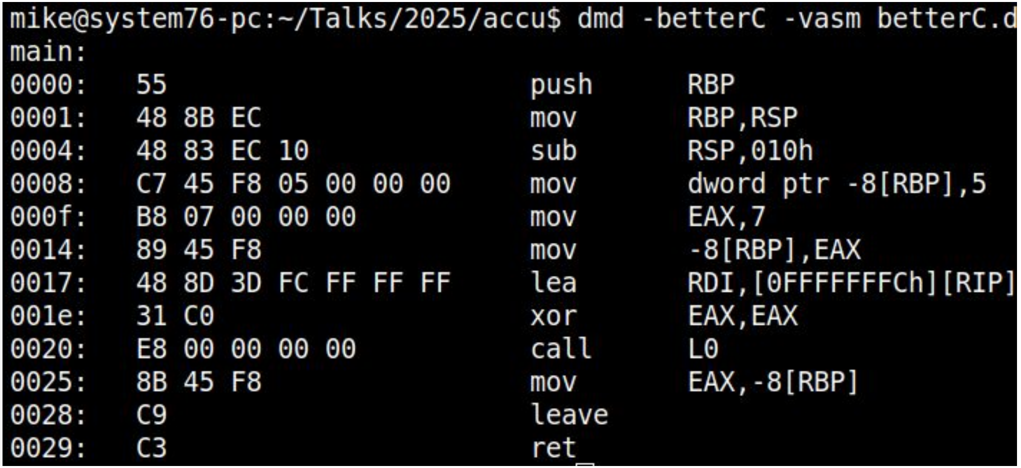
41. Better C

1. BetterC is a subset of D that doesn't depend on the D runtime library, only the C runtime library.

https://dlang.org/spec/betterc.html



```d
// dmd -betterC betterC.d

extern(C) int main(){
  import core.stdc.stdio;

  auto a = 5;
  a = 5 + 2;

  printf("Hello!");

  return a;
}
```



```
mike@system76-pc:~/Talks/2025/accu$ dmd -betterC -vasm betterC.d
main:
0000:    55                          push      RBP
0001:    48 8B EC                    mov       RBP,RSP
0004:    48 83 EC 10                 sub       RSP,010h
0008:    C7 45 F8 05 00 00 00        mov       dword ptr -8[RBP],5
000f:    B8 07 00 00 00              mov       EAX,7
0014:    89 45 F8                    mov       -8[RBP],EAX
0017:    48 8D 3D FC FF FF FF        lea       RDI,[0FFFFFFFCh][RIP]
001e:    31 C0                       xor       EAX,EAX
0020:    E8 00 00 00 00              call      L0
0025:    8B 45 F8                    mov       EAX,-8[RBP]
0028:    C9                          leave
0029:    C3                          ret
```

A Few Other Things Handy Things (1/4)

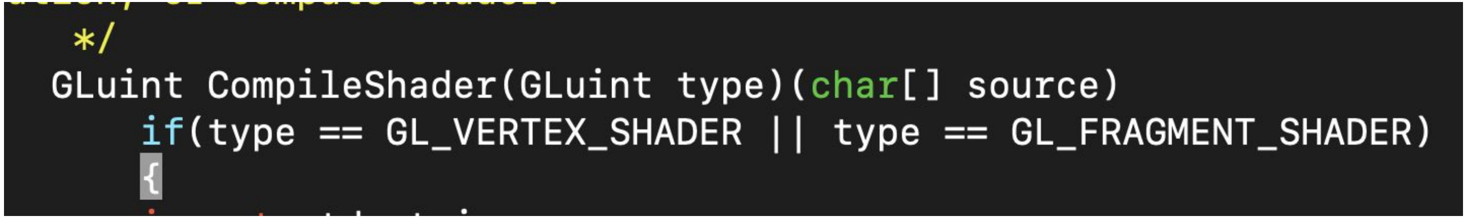
- Post condition and ‘invariant’ have been useful constructs in my code for early exit
  - e.g.
    - Checking for NaN and ensure we are always in a good state after vector operations.
    - Anytime I am creating unit Vectors (and I do so frequently) -- it’s good to not divide by 0!
  - https://dlang.org/spec/contracts.html

A Few Other Things Handy Things (2/4)

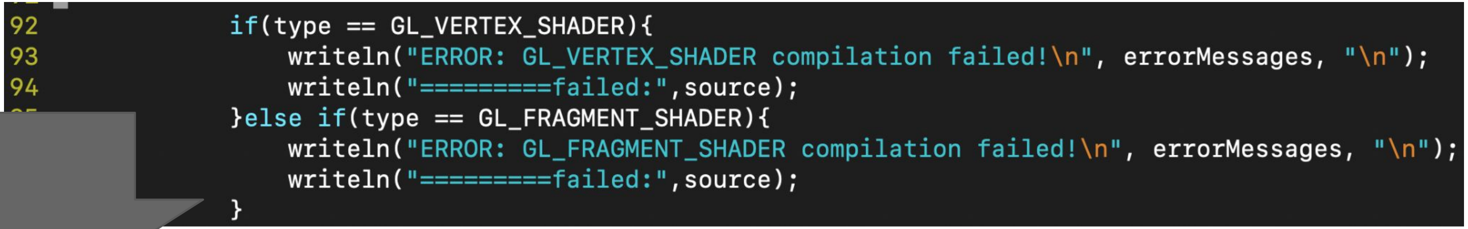
- Easy to template code
  - Able to make function templates to eliminate branches in code (i.e. which shader type to compile at run-time
    - Instead make a templated function
    - Also can apply a ‘template constraint’ to avoid illegal types from being created
      - Enforced at compile-time, again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

A Few Other Things Handy Things (3/4)

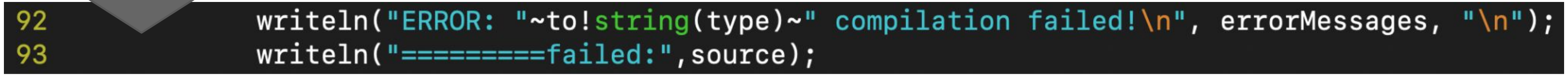
- In many places where I have enums in OpenGL, I can template them away -- often making my codebase more robust.
    - Code can be self-documenting for what ‘enum’ types are legal
    - (i.e. Use a function as a template constraint to check valid enums)



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



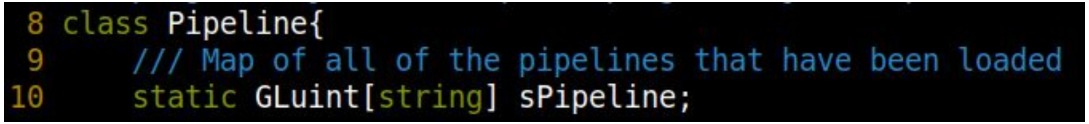
```
if(type == GL_VERTEX_SHADER){
    writeln("ERROR: GL_VERTEX_SHADER compilation failed!\n", errorMessages, "\n");
    writeln("=========failed:",source);
}else if(type == GL_FRAGMENT_SHADER){
    writeln("ERROR: GL_FRAGMENT_SHADER compilation failed!\n", errorMessages, "\n");
    writeln("=========failed:",source);
}
```



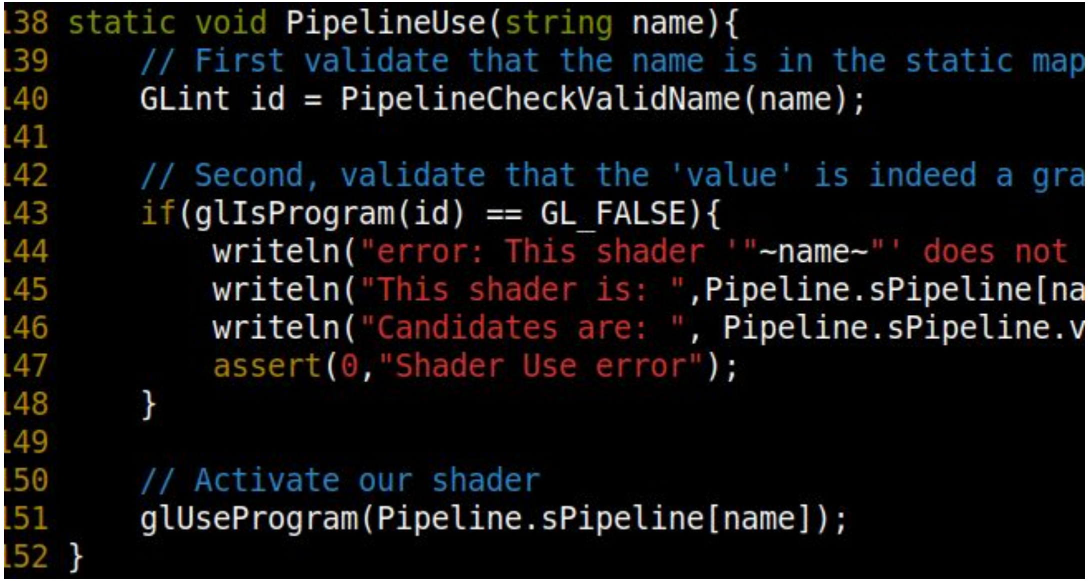
```
writeln("ERROR: "~to!string(type)~" compilation failed!\n", errorMessages, "\n");
writeln("=========failed:",source);
```

A Few Other Things Handy Things (4/4)

- Associative arrays are built-in to the D language
- It’s quite common for me to map a ‘string’ to an ‘id’ so that I can refer to things in a readable manner in my code (and print better error messages!).



```
class Pipeline{
    /// Map of all of the pipelines that have been loaded
    static GLuint[string] sPipeline;
```

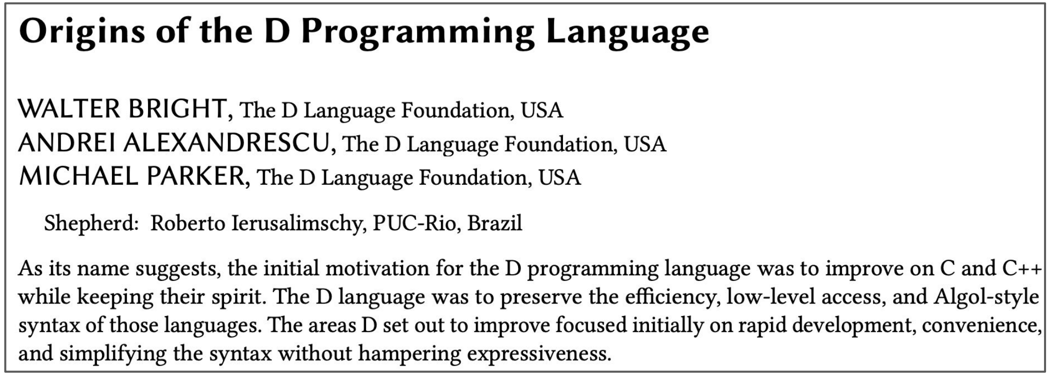


```
static void PipelineUse(string name){
    // First validate that the name is in the static map
    GLint id = PipelineCheckValidName(name);

    // Second, validate that the 'value' is indeed a gra
    if(glIsProgram(id) == GL_FALSE){
        writeln("error: This shader '"~name~"' does not
        writeln("This shader is: ",Pipeline.sPipeline[na
        writeln("Candidates are: ", Pipeline.sPipeline.v
        assert(0,"Shader Use error");
    }

    // Activate our shader
    glUseProgram(Pipeline.sPipeline[name]);
}
```

Learning More About the D Language

Further Understanding the Case for Dlang

- In 2020 the ACM's History of Programming Languages (HOPL) had an article published by Walter, Andrei, and Mike Parker to understand the origins of the language
  - I would encourage D programmers and newcomers to read the article which motivates the language and the 'why' behind its design decision.



Origins of the D Programming Language

WALTER BRIGHT, The D Language Foundation, USA
ANDREI ALEXANDRESCU, The D Language Foundation, USA
MICHAEL PARKER, The D Language Foundation, USA

Shepherd: Roberto Ierusalimschy, PUC-Rio, Brazil

As its name suggests, the initial motivation for the D programming language was to improve on C and C++ while keeping their spirit. The D language was to preserve the efficiency, low-level access, and Algol-style syntax of those languages. The areas D set out to improve focused initially on rapid development, convenience, and simplifying the syntax without hampering expressiveness.

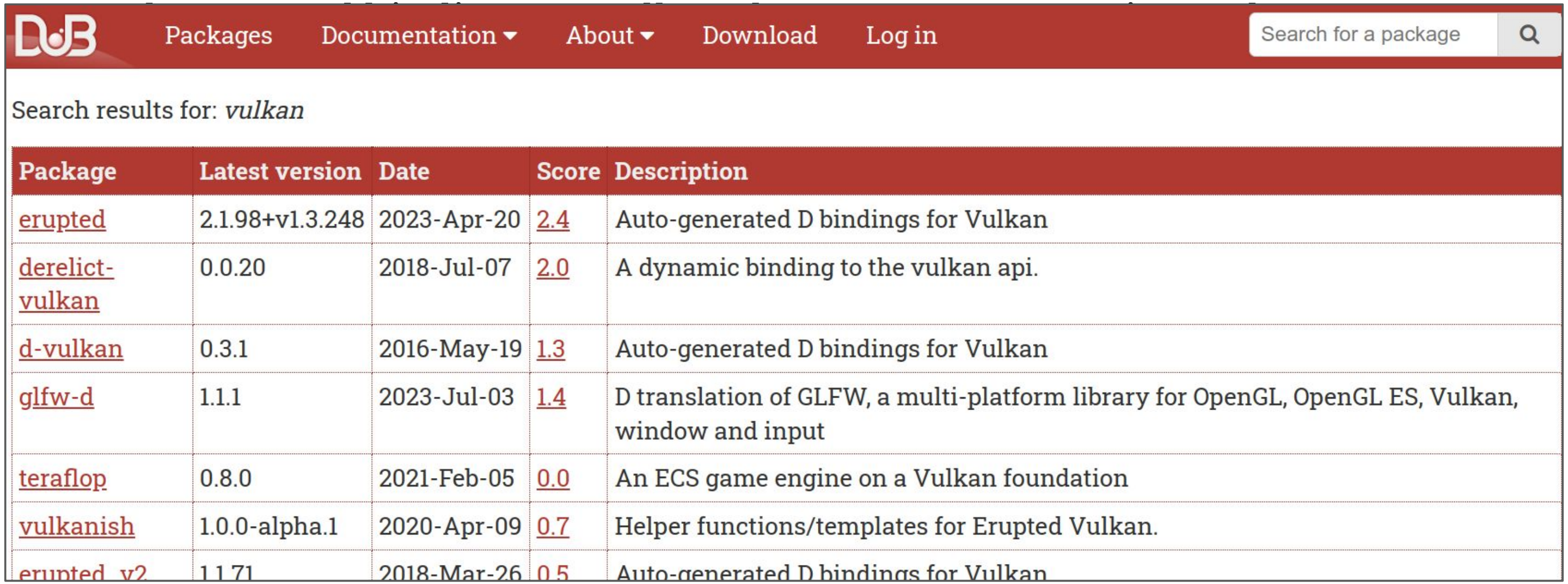
https://dl.acm.org/doi//10.1145/3386323

Further resources and training materials

- Tons of talks by others (Games, graphics, servers, etc.)
  - https://wiki.dlang.org/Videos#Tutorials
- 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
- All of my other conference talks (many related to D)
  - http://tinyurl.com/mike-talks

Vulkan

- Most folks will probably point you to Vulkan as a modern graphics API to learn
  - They are probably right -- as Vulkan allows you to create pipelines that execute much better concurrently.

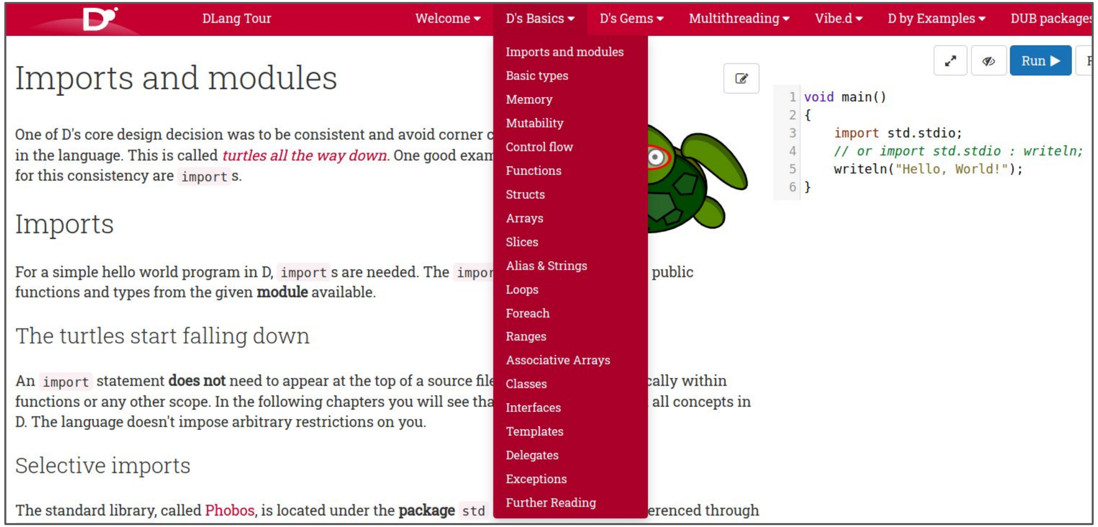


Search results for: *vulkan*

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	1.1.71	2018-Mar-26	0.5	Auto-generated D bindings for Vulkan

The D language tour

- Nice set of online tutorials that you can work through in 1 day
  - Found directly on the D language website under ‘Learn’



Imports and modules

One of D's core design decision was to be consistent and avoid corner c
in the language. This is called *turtles all the way down*. One good exam
for this consistency are `import` s.

Imports

For a simple hello world program in D, `import` s are needed. The `impor` public
functions and types from the given **module** available.

The turtles start falling down

An `import` statement **does not** need to appear at the top of a source file ally within
functions or any other scope. In the following chapters you will see tha all concepts in
D. The language doesn't impose arbitrary restrictions on you.

Selective imports

The standard library, called Phobos, is located under the **package** `std` erenced through

```
void main()
{
    import std.stdio;
    // or import std.stdio : writeln;
    writeln("Hello, World!");
}
```

https://tour.dlang.org/

Summary

- I hope you have enjoyed learning a bit about the D programming language
    - It integrates well with other tools (renderdoc, gdb) and programming skills (e.g. C or C++) you already have!
- I hope you otherwise may try to expand your horizons and try out a new language in your respective domain -- see if it gives you a competitive advantage, or otherwise improves your skills as an engineer!

accu conference 2025

Thank you ACCU 2025!

The Case for Graphics Programming Using the D Language

Mike Shah

14:00 - 15:30 Tue, April 1, 2025

90 minutes | Introductory Audience

Social: @MichaelShah
Web: mshah.io
Courses: courses.mshah.io
YouTube
www.youtube.com/c/MikeShah
http://tinyurl.com/mike-talks

Thank you!