

Attribution/License

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



FOSDEM'24

The **D** Programming Language for Modern Open Source Development

-- Programming in DLang
with Mike Shah

16:00 - 16:50 Sat, Feb 3, 2024
Location: k.1.105 (La Fontaine)
50 minutes | Introductory Audience

Social: [@MichaelShah](https://twitter.com/MichaelShah)

Web: mshah.io

Courses: courses.mshah.io

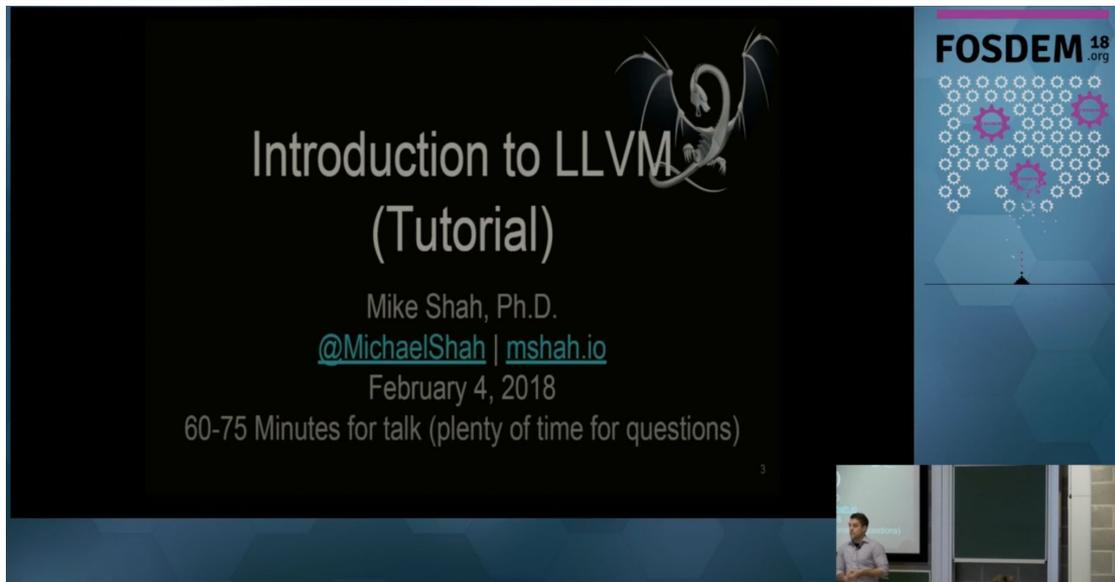
 **YouTube**

www.youtube.com/c/MikeShah

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

FOSDEM 2018

- It has been 6 years since my last FOSDEM talk!
 - My how time flies!
- Thank you very much again for having me -- we will have some fun today.
 - (And then I'll see you again in hopefully < 6 years)



Introduction to LLVM
(Tutorial)

Mike Shah, Ph.D.
[@MichaelShah](#) | [mshah.io](#)
February 4, 2018
60-75 Minutes for talk (plenty of time for questions)

FOSDEM 18.org

Introduction to LLVM Building simple program analysis tools and instrumentation

https://www.youtube.com/watch?v=VKIv_Bkp4pk

FOSDEM 2018

- It has been 6 years since my last FOSDEM talk!
 - My how time flies!
- Thank you very much again for having me -- we will have some fun today.
 - (And then I'll see you again in hopefully < 6 years)



Introduction to LLVM
(Tutorial)

Mike Shah, Ph.D.
[@MichaelShah](https://twitter.com/MichaelShah) | mshah.io
February 4, 2018
60-75 Minutes for talk (plenty of time for questions)

2:27 Sat, Feb 3
Internet Battery Saver
Do Not Disturb Wallet
Photos • 15m
6 years ago, today...
Look back at Feb 3, 2018
Manage Clear all

Introduction to LLVM Building simple program analysis tools and instrumentation

https://www.youtube.com/watch?v=VKIv_Bkp4pk

Your Tour Guide for Today

by Mike Shah

- **Associate Teaching Professor** at Northeastern University in Boston, Massachusetts.
 - I **love** teaching: courses in computer systems, computer graphics, geometry, and game engine development.
 - My **research** is divided into computer graphics (geometry) and software engineering (software analysis and visualization tools).
- I do **consulting** and **technical training** on modern C++, DLang, Concurrency, and Graphics Programming
 - Usually graphics or games related -- e.g. Building 3D application plugins
- Outside of work: guitar, running/weights, traveling and cooking are fun to talk about



Web

www.mshah.io



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

Non-Academic Courses

courses.mshah.io

Je parle une petit francais -- Bienvenue!

Je suis prefere le question en anglas pour le meillere result

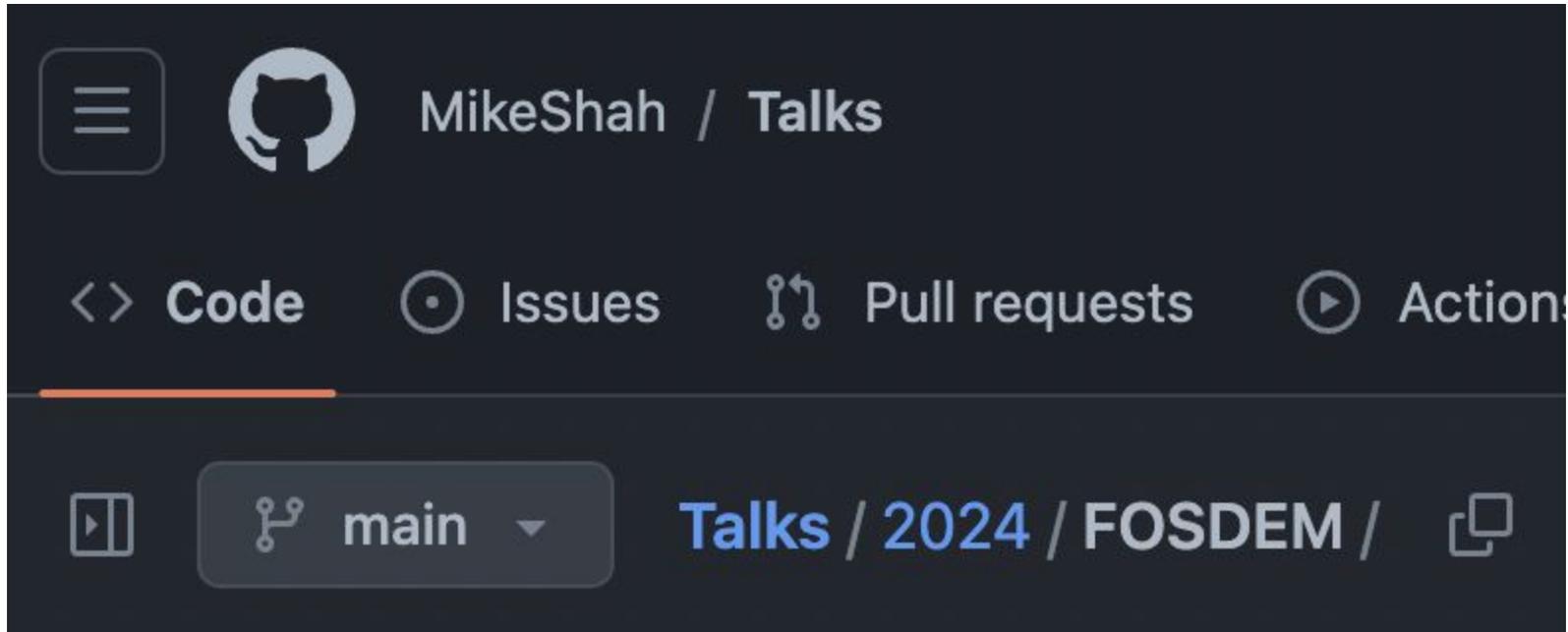
Abstract

The abstract that you read and enticed you to join me is here!

The D programming language has been quietly growing for well over two decades. This modern programming language supports multiple programming paradigms, a range of memory safety features, and an ecosystem with 3 open source compilers. So why should an open source developer consider learning or using the D programming language? In this talk I will show examples of how D has replaced all of my Python code for my projects, and why I think D truly is a language that allows you to "write fast, read fast, and run fast" code. I will introduce the language, several of my favorite productivity features, and tools in the D programming language ecosystem. Throughout the talk, the audience will also be pointed to several open source tools written in the D language to be inspired from. Audience members looking for a new language to learn, or otherwise the programming language enthusiast may also benefit from a tour of the D language and its features.

Code for the talk

- Located here: <https://github.com/MikeShah/Talks/tree/main/2024/FOSDEM>



What I want to do today...

- I want you to get excited or curious about an **open source project** -- a programming language!
- That language is of course... the **D programming language**!
- And maybe one day -- you will contribute to the compiler or ecosystem!



GitHub

<https://github.com> › [dlang](#) › [dmd](#) ⋮

dlang/dmd: dmd D Programming Language compiler

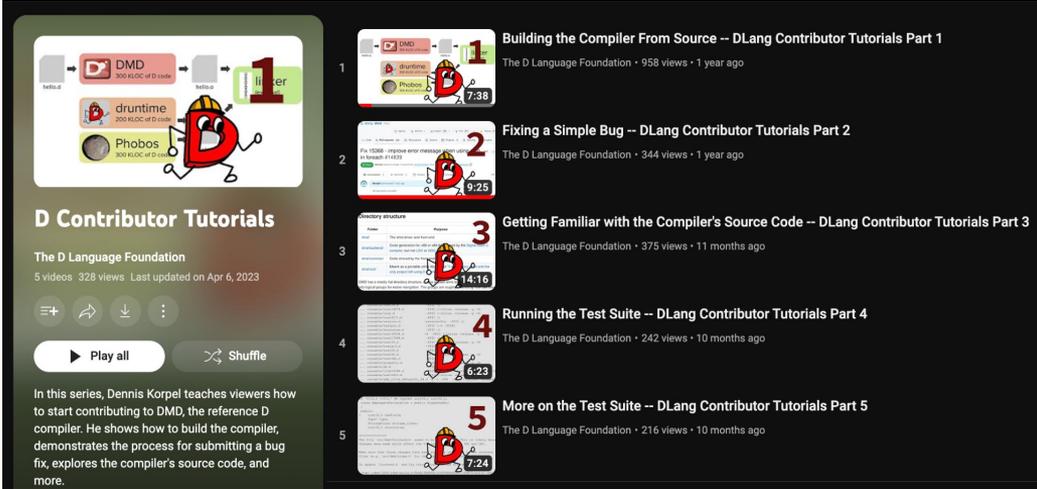
DMD is the reference compiler for the D programming language. Releases, language specification and other resources can be found on the homepage.

[Projects 4](#) · [LICENSE.txt](#) · [Pull requests 262](#) · [CONTRIBUTING.md](#)

<https://github.com/dlang/dmd>

(Pssst...My dream for you is to get excited enough to contribute)

- There's a great playlist (linked below) where you can learn about hacking on the compiler and contributing to this project
 - I think there's also plenty to learn just looking at the source code of a **D's very fast reference compiler (DMD)**
 - And maybe you'll one day fix a bug or two!
- Okay -- now that you know what my dream is for you -- let's do the rest of the talk.



D Contributor Tutorials

The D Language Foundation
5 videos · 328 views · Last updated on Apr 6, 2023

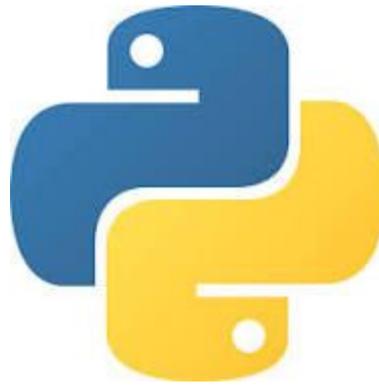
Play all Shuffle

In this series, Dennis Korpel teaches viewers how to start contributing to DMD, the reference D compiler. He shows how to build the compiler, demonstrates the process for submitting a bug fix, explores the compiler's source code, and more.

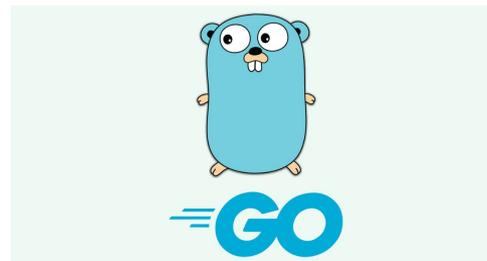
- 1 Building the Compiler From Source -- DLang Contributor Tutorials Part 1
The D Language Foundation · 958 views · 1 year ago · 7:38
- 2 Fixing a Simple Bug -- DLang Contributor Tutorials Part 2
The D Language Foundation · 344 views · 1 year ago · 9:25
- 3 Getting Familiar with the Compiler's Source Code -- DLang Contributor Tutorials Part 3
The D Language Foundation · 375 views · 11 months ago · 14:16
- 4 Running the Test Suite -- DLang Contributor Tutorials Part 4
The D Language Foundation · 242 views · 10 months ago · 6:23
- 5 More on the Test Suite -- DLang Contributor Tutorials Part 5
The D Language Foundation · 216 views · 10 months ago · 7:24

D Contributor Tutorials

<https://www.youtube.com/playlist?list=PLIldXzSkPUXXSkM5NjBAGNIIdkd4Q2Zf0R>



So I'm a bit of a programming language enthusiast



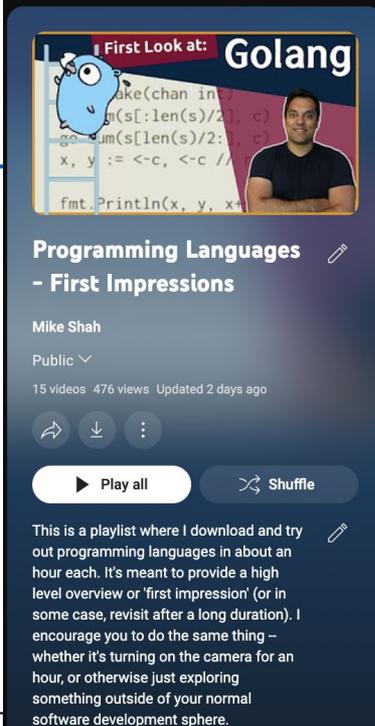
The past few months...

- I've been spending ~1-hour trying new programming languages
 - Most languages are new to me.
 - Some languages are very popular
 - Some languages are less mainstream

My recordings of 18 (and counting) programming languages can be found on the playlist below

Playlist:

<https://www.youtube.com/playlist?list=PLvv0ScY6vfd-5hJ47DNAOKLLIHjz1Tzq>



First Look at: Golang

Programming Languages - First Impressions

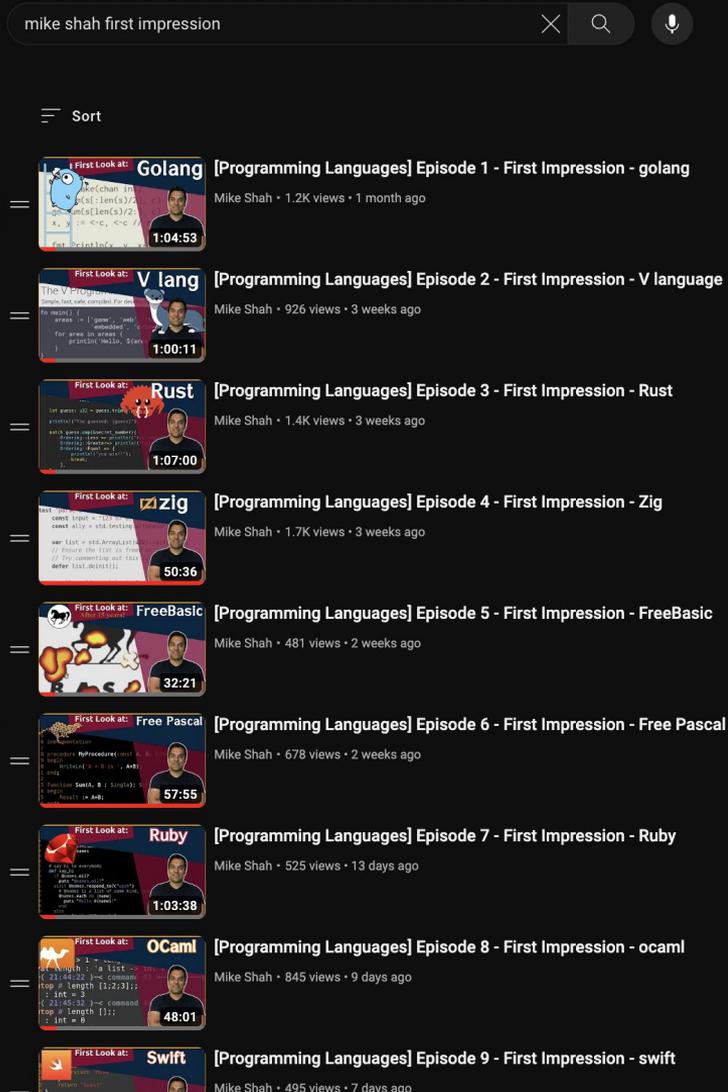
Mike Shah

Public

15 videos 476 views Updated 2 days ago

Play all Shuffle

This is a playlist where I download and try out programming languages in about an hour each. It's meant to provide a high level overview or 'first impression' (or in some case, revisit after a long duration). I encourage you to do the same thing - whether it's turning on the camera for an hour, or otherwise just exploring something outside of your normal software development sphere.



mike shah first impression

Sort

[Programming Languages] Episode 1 - First Impression - golang
Mike Shah • 1.2K views • 1 month ago

[Programming Languages] Episode 2 - First Impression - V language
Mike Shah • 926 views • 3 weeks ago

[Programming Languages] Episode 3 - First Impression - Rust
Mike Shah • 1.4K views • 3 weeks ago

[Programming Languages] Episode 4 - First Impression - Zig
Mike Shah • 1.7K views • 3 weeks ago

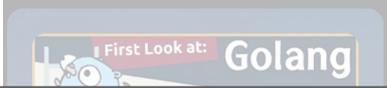
[Programming Languages] Episode 5 - First Impression - FreeBasic
Mike Shah • 481 views • 2 weeks ago

[Programming Languages] Episode 6 - First Impression - Free Pascal
Mike Shah • 678 views • 2 weeks ago

[Programming Languages] Episode 7 - First Impression - Ruby
Mike Shah • 525 views • 13 days ago

[Programming Languages] Episode 8 - First Impression - ocaml
Mike Shah • 845 views • 9 days ago

[Programming Languages] Episode 9 - First Impression - swift
Mike Shah • 495 views • 7 days ago



Sort

My goal today is not to convince you that any one programming language is better than another (I'm smarter than that...and we all have our favorites)

But--I do want to share my enthusiasm for D which stands out to me -- it's a language I have fun writing code in.

So in the same way that I've been exploring languages recently, I want to provide an **introduction to you of the D language** for about an hour.

...and maybe you will find some features in D you like and -- **maybe you'll try out Dlang!**

- First Impression - golang
- First Impression - V language
- First Impression - Rust
- First Impression - Zig
- First Impression - FreeBasic
- First Impression - Free Pascal
- First Impression - Ruby
- First Impression - ocaml

Playlist:
<https://www.youtube.com/playlist?list=PLv00cY6vfd-5hJ47DNAOKKLLIHjz1Tzq>

48:01



[Programming Languages] Episode 9 - First Impression - swift

Mike Shah • 495 views • 7 days ago

The past few months...



[Programming Languages] Episode 1 - First Impression - golang
Mike Shah · 1.2K views · 1 month ago

- I've tried...
- My...

Along our journey -- I'm going to show you some cool projects for inspiration.

Most of which are fully open source!

All of which you can learn something from.

And all of them in the D programming language.

- First Impression - V language
- First Impression - Rust
- First Impression - Zig
- First Impression - FreeBasic
- First Impression - Free Pascal
- First Impression - Ruby

Playlist:
<https://www.youtube.com/playlist?list=PLvv0ScY6vfd-5hJ47DNAOKKLLIHjz1Tzq>

1:03:38



[Programming Languages] Episode 8 - First Impression - ocaml
Mike Shah · 845 views · 9 days ago



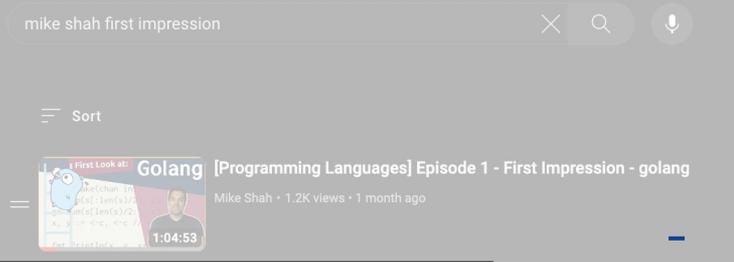
[Programming Languages] Episode 9 - First Impression - swift
Mike Shah · 495 views · 7 days ago

The past few months...

- I've been trying to learn a new language every month for the past few months.
- My goal was to learn a new language every month for the past few months.

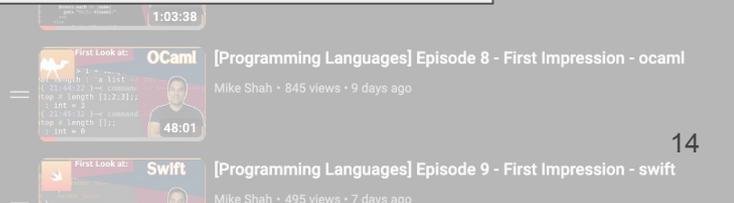
So -- let's begin!

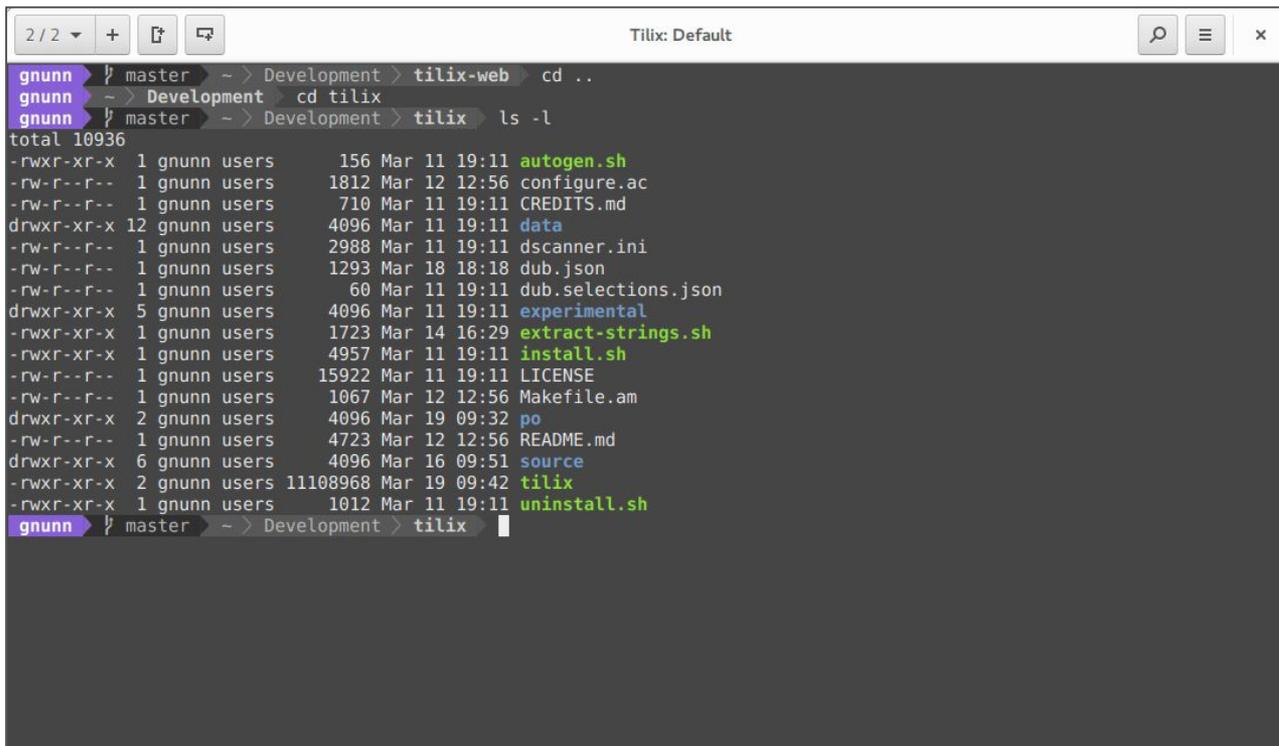
(...and start with something cool made in D)



- First Impression - V language
- First Impression - Rust
- First Impression - Zig
- First Impression - FreeBasic
- First Impression - Free Pascal
- First Impression - Ruby

Playlist:
<https://www.youtube.com/playlist?list=PLvv0ScY6vfd-5hJ47DNAOKKLLIHjz1Tzq>





```
gnunn | master ~ > Development > tilix-web > cd ..
gnunn | ~ > Development > cd tilix
gnunn | master ~ > Development > tilix > ls -l
total 10936
-rwxr-xr-x 1 gnunn users 156 Mar 11 19:11 autogen.sh
-rw-r--r-- 1 gnunn users 1812 Mar 12 12:56 configure.ac
-rw-r--r-- 1 gnunn users 710 Mar 11 19:11 CREDITS.md
drwxr-xr-x 12 gnunn users 4096 Mar 11 19:11 data
-rw-r--r-- 1 gnunn users 2988 Mar 11 19:11 dscanner.ini
-rw-r--r-- 1 gnunn users 1293 Mar 18 18:18 dub.json
-rw-r--r-- 1 gnunn users 60 Mar 11 19:11 dub.selections.json
drwxr-xr-x 5 gnunn users 4096 Mar 11 19:11 experimental
-rwxr-xr-x 1 gnunn users 1723 Mar 14 16:29 extract-strings.sh
-rwxr-xr-x 1 gnunn users 4957 Mar 11 19:11 install.sh
-rw-r--r-- 1 gnunn users 15922 Mar 11 19:11 LICENSE
-rw-r--r-- 1 gnunn users 1067 Mar 12 12:56 Makefile.am
drwxr-xr-x 2 gnunn users 4096 Mar 19 09:32 po
-rw-r--r-- 1 gnunn users 4723 Mar 12 12:56 README.md
drwxr-xr-x 6 gnunn users 4096 Mar 16 09:51 source
-rwxr-xr-x 2 gnunn users 11108968 Mar 19 09:42 tilix
-rwxr-xr-x 1 gnunn users 1012 Mar 11 19:11 uninstall.sh
gnunn | master ~ > Development > tilix |
```

- Blog on Development: <https://dlang.org/blog/2017/08/11/on-tilix-and-d-an-interview-with-gerald-nunn/>
- Github or Dub Repository: <https://github.com/gnunn1/tilix/>

```

2 / 2 + [ ] [ ]
Tilix: Default
gnunn ~ master ~ Development > tilix-web > cd ..
gnunn ~ Development > cd tilix
gnunn ~ master ~ Development > tilix > ls -l
total 10936
-rwxr-xr-x 1 gnunn users 156 Mar 11 19:11 autogen.sh
-rw-r--r-- 1 gnunn users 1812 Mar 12 12:56 configure.ac
-rw-r--r-- 1 gnunn users 710 Mar 11 19:11 CREDITS.md
drwxr-xr-x 12 gnunn users 4096 Mar 11 19:11 data
-rw-r--r-- 1 gnunn users 2988 Mar 11 19:11 dscanner.ini
-rw-r--r-- 1 gnunn users 1293 Mar 18 18:18 dub.json
-rw-r--r-- 1 gnunn users 60 Mar 11 19:11 dub.selections.json
drwxr-xr-x 5 gnunn users 4096 Mar 11 19:11 experimental
-rwxr-xr-x 1 gnunn users 1723 Mar 14 16:29 extract-strings.sh
-rwxr-xr-x 1 gnunn users 4957 Mar 11 19:11 install.sh
-rw-r--r-- 1 gnunn users 15922 Mar 11 19:11 LICENSE
-rw-r--r-- 1 gnunn users 1067 Mar 12 12:56 Makefile.am
drwxr-xr-x 2 gnunn users 4096 Mar 19 09:32 po
-rw-r--r-- 1 gnunn users 4723 Mar 12 12:56 README.md
drwxr-xr-x 6 gnunn users 4096 Mar 16 09:51 source
-rwxr-xr-x 2 gnunn users 11108968 Mar 19 09:42 tilix
-rwxr-xr-x 1 gnunn users 1012 Mar 11 19:11 uninstall.sh
gnunn ~ master ~ Development > tilix

```

Why you might care to look?

- Nice look at how to do GUI development with libraries like gtk.
- D can simply import C code with **ImportC**
 - **A full C compiler built into D**
- Many bindings to C libraries that you get for free with the D language.

<https://github.com/gnunn1/tilix/blob/master/source/app.d>

```

import gtk.Main;
import gtk.Version;
import gtk.MessageDialog;

import gx.i18n.l10n;
import gx.gtk.util;
import gx.gtk.vte;

import gx.tilix.application;
import gx.tilix.cmdparams;
import gx.tilix.constants;

int main(string[] args) {
    static if (USE_FILE_LOGGING) {
        sharedLog = new FileLogger("/tmp/tilix.log");
    }

    bool newProcess = false;
    string group;

    string cwd = Util.getCurrentDir();
    string pwd;
    string de;
    trace("CWD = " ~ cwd);

```

- Blog on Development: <https://dlang.org/blog/2017/08/11/on-tilix-and-d-an-interview-with-gerald-nunn/>
- Github or Dub Repository: <https://github.com/gnunn1/tilix/>



DLang a First Impression

(La premiere impression)

Pop Quiz: (l'examen surprise!) (1/3)

- Let's take a look at an example of D code
 - I'll give everyone a minute to think about it
 - Try to think about what is being done
 - So... what does this program do?

```
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
9     static immutable b = sort(a);
10
11
12     pragma(msg, "Finished compilation: ", b);
13 }
14
15
```

Pop Quiz: (l'examen surprise!) (2/3)

- One of the first examples on the www.dlang.org webpage
 - An example of sorting an array!
 - Line 3:
 - There's a built-in standard library (named 'Phobos')
 - Line 5:
 - Function call using universal function call syntax (UFCS)
 - Line 7:
 - enum constant
 - Line 9:
 - immutable static data stored in b
 - Line 12:
 - pragma outputs value after compilation
- This program does most of its work (the working) at compile-time!

Sort an Array at Compile-Time

[your code here](#)

```
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
```

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](#)

```
d.stdout;
```

```
n;
```

```
ng_compilation  
mpilation: ", b);
```

- pragma outputs value after compilation

```
14
```

```
15
```

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



The D Programming Language

(Le langage de programmation D)

D Language History - Created by Walter Bright [[wiki](#)]

- Walter Bright
 - Wrote a C Compiler (Datalight C compiler)
 - Famously created the Zortech C++ compiler
 - Designed the game Empire
 - (There is even a translation of Empire to D!)
 - Between 1999-2006 worked alone on D version 1 programming language.
 - (Originally it was the Digital Mars Compiler, but everyone colleagues and friends insisted on calling it the next evolution to C++, thus the name 'D')
- Around 2006 or 2007 -- D2 would start being developed with Andrei Alexandrescu and others.
 - Full history here - Origins of the D Programming Language
 - <https://dl.acm.org/doi/pdf/10.1145/3386323>



Dconf 2022 in London

D hosts an online and in-person conference every year: <https://dconf.org/>

So what is the D Programming Language? (1/2)

So what is the D Programming Language? (2/2)

D is a general-purpose programming language with static typing, systems-level access, and C-like syntax. With the **D Programming Language**, write fast, read fast, and run fast.

So, over the last 25 years -- now three D Compilers!

- DMD is the official reference compiler
 - The compiler is **open-source** and you can fork a copy of it today
 - DMD is a **very fast compiler** (in part because of D's module system)
- GDC
 - GCC-based D Compiler Frontend
 - Good GDB support
- LDC - LLVM based D Compiler
 - Allows you to get LLVM optimizations and target many architectures

Note: Common for D programmers to develop in DMD for quick edit-compile-run cycles, and then deploy using GDC or LDC

Downloads

Choose a compiler

[\(more information\)](#)



DMD

- Official reference compiler
- Latest D version
- Simple installation
- Very fast compilation speeds
- Architectures: i386, amd64

[About](#) · [Download](#)



GDC

- [GCC](#)-based D compiler
- Strong optimization
- Great [GDB](#) support
- Architectures: i386, amd64, x32, armel, armhf, [others](#)

[About](#) · [Download](#)



LDC

- [LLVM](#)-based D compiler
- Strong optimization
- [Android support](#)
- Architectures: i386, amd64, armel, armhf, [others](#)

[About](#) · [Download](#)

<https://dlang.org/download.html>

Downloading the Tools

- The download of any of the compilers is relatively simple and available for many architectures from the homepage
 - Along with the download, you also get:
 - **Dub** - the package manager for managing dependencies and as a lightweight build tool.
 - Other useful tools like **dfmt** (a code formatter) and **dscanner** (a linter) exist
 - A VSCode extension (**code-d**) is available, as well as some support in IntelliJ for D.

DMD 2.107.0

[Changelog](#)

	Windows ⓘ
	Installer 7z
	macOS ⓘ
	dmg tar.xz
	Ubuntu/Debian ⓘ
	i386 x86_64 tar.xz
	Fedora/CentOS ⓘ
	i386 x86_64 tar.xz
	openSUSE ⓘ
	i386 x86_64 tar.xz
	FreeBSD ⓘ
	x86_64



Language Server

To start coding effectively, we recommend using an editor supporting the Language Server Protocol. For VSCode you can immediately install the [VSCode extension code-d \[openvsx\]](#)

[About](#) · [Download](#)

<https://dlang.org/download.html>

DLang Domains

- It's a general purpose-language systems language, so D can be used in any domain.
- Dlang has found **some niches in performance-based** domains:
 - e.g. image processing, gaming, streaming, finance, and simulation

 eBay <i>One of the world's largest marketplaces</i> Large scale data mining tools. Command line tools in D GitHub	 eCratum <i>Supplier Management tool for SME</i> Core applications (Public API, Support app) use D. Hiring	 Emsi <i>Data-driven modelling</i> N-dimensional dataset processing, in-memory data manipulation DConf talk GitHub
 Facebook <i>Online social networking service</i> C Preprocessor warp and more infrastructure tools . DConf talk	 Funatics <i>MMO Game Developer</i> “D, with its elegance, simplicity and performance, turned out to be the perfect replacement for Node.js” DConf talk	 Funkwerk AG <i>Passenger information systems</i> “We use D to achieve perfect, well-readable code.” DConf talk Hiring
 Infognition <i>Video processing</i> “D shines from low-level control to high-level abstractions.” Testimonial	 JumiaFood <i>Instant delivery platform</i> “D tremendously helps us to monitor our entire Kubernetes infrastructure.” GitHub	 Magikcraft <i>Interactive teaching platform</i> “We use D language for microservices and other awesome things!” GitHub

<https://dlang.org/orgs-using-d.html>

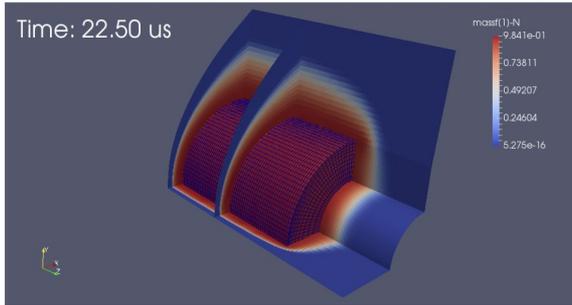
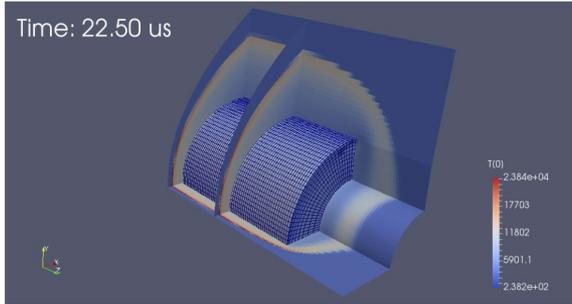
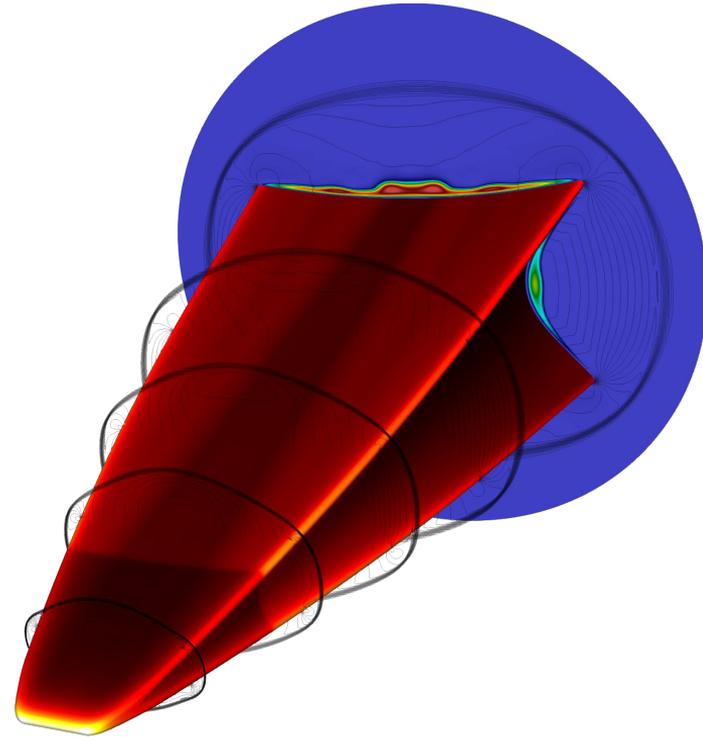


Figure 5.7: Static temperature and mass fraction of nitrogen atoms in the flow field from the chemical nonequilibrium simulation.



- 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>

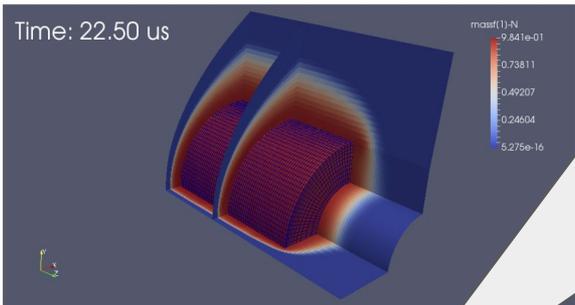
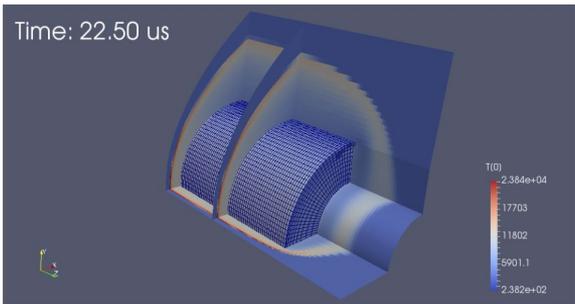


Figure 5.7: Static temperature and mass fraction of nitrogen in the flow field from the chemical nonequilibrium simulation.

Why you might care to look?

- Project with 10+ years in development
 - High performance!
- From the github page: *“Our focus is on open source development to give a simple access point for doing gas dynamics in research and teaching.”*

<https://gdtk.uqcloud.net/docs/eilmer/about/>

Install prerequisites

The main requirement is a D language compiler. We recommend using the latest stable release of the LLVM D compiler.

To build Eilmer and other programs in the toolkit, you will require:

- D compiler
 - Binary releases for the latest stable release of the LLVM D compiler (`ldc2` and `ldmd2`) may be found at: <https://github.com/ldc-developers/ldc/releases> .

- 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>

DLang Features

- We've seen **compile-time function execution** (ctfe) as one modern feature of the D language compiler
- The language itself supports many nice quality of life features for safety and productivity -- for example:
 - Built-in dynamic arrays
 - Built-in Associative arrays (i.e. map/hashtable/dictionary)
 - Bounds checked arrays
 - (With ability to disable if needed)
 - lambda's and delegates
 - Uniform Function Call Syntax (UFCS)
 - Object-Oriented Programming Paradigm
 - Functional paradigms (lazy evaluation, pure functions)
 - Concurrency
 - Garbage Collection or manual memory management options
 - i.e. You can just use malloc/free if you really want!
 - and more!

Features Overview

Navigate D's implementation of a few key programming language concepts.

- [Garbage Collection](#)
- Functions
 - [Function Delegates](#)
 - [Function Overloading](#)
 - [out parameters for functions](#)
 - [Nested functions](#)
 - [Function literals](#)
 - [Closures](#)
 - [Typesafe variadic arguments](#)
 - [Lazy function argument evaluation](#)
 - [Compile time function evaluation](#)
 - [Uniform Function Call Syntax](#)
 - [User-Defined Attributes](#)
- Arrays
 - [Lightweight arrays](#)
 - [Resizeable arrays](#)
 - [Built-in strings](#)
 - [Array slicing](#)
 - [Array bounds checking](#)
 - [Array literals](#)
 - [Associative arrays](#)
 - [String switches](#)
 - [Aliases](#)
- OOP
 - Object Orientation
 - [Interfaces](#)
 - Single inheritance of implementation/multiple inheritance of interfaces

Phobos The Standard Runtime Library

- Phobos is the **standard runtime library** that comes with D.
 - Thus, I like to think of D as a ‘batteries included’ language
 - You can get started immediately and be productive and writing software to solve problems.
 - Phobos comes ready with a rich set of algorithms, containers (data structures), and other common libraries for solving problems.
 - “Containers” are the standard libraries **data structures** (beyond the built-in types) that describe how we access and store data.
 - And the “**algorithms**” and “**ranges**” and are building blocks for computation
- The Standard Library (std) has common data structures and ability to work with data (json, csv, xml), compression (zip), networking (sockets, curl), etc.

Phobos Runtime Library

Phobos is the standard runtime library that comes with the D language compiler.

Generally, the `std` namespace is used for the main modules in the Phobos standard library. The `etc` namespace is used for external C/C++ library bindings. The `core` namespace is used for low-level D runtime functions.

The following table is a quick reference guide for which Phobos modules to use for a given category of functionality. Note that some modules may appear in more than one category, as some Phobos modules are quite generic and can be applied in a variety of situations.

Modules	Description
<i>Algorithms & ranges</i>	
<code>std.algorithm</code> <code>std.range</code> <code>std.range.primitives</code> <code>std.range.interfaces</code>	Generic algorithms that work with <code>ranges</code> of any type, including strings, arrays, and other kinds of sequentially-accessed data. Algorithms include searching, comparison, iteration, sorting, set operations, and mutation.
<i>Array manipulation</i>	
<code>std.array</code> <code>std.algorithm</code>	Convenient operations commonly used with built-in arrays. Note that many common array operations are subsets of more generic algorithms that work with arbitrary ranges, so they are found in <code>std.algorithm</code> .
<i>Containers</i>	
<code>std.container.array</code> <code>std.container.binaryheap</code> <code>std.container.dlist</code> <code>std.container.rbtrees</code> <code>std.container.slist</code>	See <code>std.container.*</code> for an overview.

<https://dlang.org/phobos/index.html>

```
23 // Retrieves all of the playlists from the channel.
24 void GetPlaylists(){
25     // Query all the playlists for the channel
26     string query = "https://youtube.googleapis.com/youtube/v3/playlists?part=snippet%2CcontentDetails&channelId=~gChannelID~"&maxResults=50&key=~gYouTubeAPIKey~";
27
28     // Perform the query
29     auto content = get(query);
30     // Now we parse the content into json "j"
31     auto j = parseJSON(content);
32
33     foreach(key; j["items"].array){
34         writeln("    <tr>");
35         string id = strip(j["items"][counter]["id"].toString, "\\");
```

The following is a capture of one of my command-line scripts

- I take advantage of **std.net.curl** to make YouTube API calls
 - See line 29 (Note: Consider using a Builder to create a Query String)
- **std.json** is then used to retrieve data
 - 'auto' at line 29 infers the type, and then we parse the JSONObject
- Then I use a **range-based** loop (line 33) to iterate through the keys of my json object.

```
41 /// Perform a Request on the github api
42 auto GetRequest(DefaultUser user, Course course, string request){
43     // Setup an HTTP Request
44     auto http = HTTP();
45     http.url = "https://api.github.com/orgs/~course.coursename~/~request;
46     http.method = HTTP.Method.get;
47     http.setAuthentication(user.uname,user.OAUTH);
48
49     // Store the result of the data that we retrieve
50     char[]    resultString;
51
52     // Retrieve the header data
53     http.onReceiveHeader = (in char[] key, in char[] value) {
54         writeln("onRecieveHeader: ",key, ": ", value);
55     };
```

Yet another tool -- again -- same pattern but with calls to GitHub API

- Observe line 53 we set the event handler using a lambda function
 - Attributes **'in'** function effectively as 'transitive const' data.

Set the event handler that receives incoming headers.

```
{null} onReceiveHeader();
```

DLang for Scripts (1/2)

- As an interesting anecdote -- most of these scripts (and I have dozens of them...) use to be written in Python.
 - The translation was relatively simple -- and I've found D to be writeable like the Python language
- **But -- I still execute my source files like scripts in Python**
 - (I'll explain on the next slide)

Curl + YouTube API

```
23 // Retrieves all of the playlists from the channel.
24 void GetPlaylists(){
25     // Query all the playlists for the channel
26     string query = "https://youtube.googleapis.com/youtube/v3/playlists?part=snippet%2CcontentDetails&channelId=~gChannelID~&maxResults=50&key=~gYouTubeAPIKey~";
27
28     // Perform the query
29     auto content = get(query);
30     // Now we parse the content into json "j"
31     auto j = parseJSON(content);
32
33     foreach(key; j["items"].array){
34         writeln("  <tr>");
35         string id = strip(j["items"][counter]["id"].toString, "\\");
```

Curl + Github API

```
41 /// Perform a Request on the github api
42 auto GetRequest(DefaultUser user, Course course, string request){
43     // Setup an HTTP Request
44     auto http = HTTP();
45     http.url = "https://api.github.com/orgs/~course.coursename~/~request;
46     http.method = HTTP.Method.get;
47     http.setAuthentication(user.uname,user.OAUTH);
48
49     // Store the result of the data that we retrieve
50     char[] resultString;
51
52     // Retrieve the header data
53     http.onReceiveHeader = (in char[] key, in char[] value) {
54         writeln("onRecieveHeader: ",key, ": ", value);
55     };
```

DLang for Scripts (2/2)

- A little helper tool called **rdmd** will compile (and cache) on the fly.
 - Great -- now I get a **statically typed, systems language** that I can write my scripts in.
- (Note: ldmd2 is the equivalent for the [LDC](#) compiler of rdmd)

On-the-fly compilation with `rdmd`

The helper tool `rdmd`, distributed with the DMD compiler, will make sure to compile all dependencies and automatically runs the resulting application:

```
rdmd hello.d
```

On UNIX systems the shebang line `#!/usr/bin/env rdmd` can be put on the first line of an executable D file to allow a script-like usage.

Browse the [online documentation](#) or run `rdmd --help` for an overview of available flags.

(Aside) DLang for Scripts Performance

- *Generally speaking*, compiled languages (as you may know) often achieve more performance versus interpreted languages
 - That is the case with my 'D' versus 'Python' performance case
 - (Yes, Python numpy, or calling into pyCuda speeds things up)
 - But the point is, I get a language that's easy to write in, but boosts great performance.
 - **DMD is a fast compiler -- rdmd allows me to use dmd almost like a scripting language**
- **Yet -- there's more to the performance story!**



[Dlang Series Teaser] Dlang versus Python speed comparison (Matrix Multiply)

<https://www.youtube.com/watch?v=HS7X9ERdjM4&list=PLvv0ScY6vfd9Fso-3cB4CGnSIW0E4btJV>

```
134     foreach(student ; students.parallel){
135         if(student.reponame.indexOf(reponame_prefix)>=0){
136             auto pid = spawnShell("git clone https://"~us
se.coursename~/~student.reponame~");
137         }
138     }
```

I can get thread-based parallelism relatively cheaply!

- Observe line 134, I can simply call `.parallel` on an array, and within a range-based loop this create multiple threads.



- So here was a Raytracer that I built-in the D programming language
 - An obvious candidate for parallelism from the [std.parallelism](#) module

- Talk/Website: <https://www.youtube.com/watch?v=nCIB8df7q2g>
- Github or Dub Repository: https://github.com/MikeShah/Talks/tree/main/2022/2022_dconf_London

```
74     foreach(y ; cam.GetScreenHeight.iota.parallel){
75         foreach(x; cam.GetScreenHeight().iota.parallel){
76 //     for(int y=cam.GetScreenHeight()-1; y >=0; --y){
77 //         for(int x= 0; x < cam.GetScreenWidth(); ++x){
78
79         // Cast ray into scene
80         // Accumulate the pixel color from multiple samples
81         Vec3 pixelColor = Vec3(0.0,0.0,0.0);
```

- Again observe that I'm able to parallelize this loop
- There's also something interesting going on here syntactically to talk about with D
- The function calls take advantage of **Universal Function Call Syntax (UFCS)** ([UFCS](#)) -- a great feature for readability
 - `cam.GetScreenHeight.iota.parallel`
 - **as opposed to**
 - `parallel(iota(cam.GetScreenHeight())));`

(Aside)

- The D Compiler has a **built-in profiler and gc (memory) profiler**
- You can watch my previous talk below to learn more about how .parallel improved performance

- **DConf Online '22 - Engineering a Ray Tracer on the Next Weekend with DLang**
- <https://www.youtube.com/watch?v=MFhTRiobWfU>

-profile [switches see -profile]

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

- So highlighted above is the '-profile' flag being used.
- Below is the summary of the profile (trace.log)
 - Note the summary is found at the bottom of trace.log

```
614 ===== Timer frequency unknown, Times are in Megaticks =====
615
616 Num      Tree      Func      Per
617 Calls   Time      Time      Call
618
619 4888100  51585     51369     0    double utility.GenerateRandomDouble()
620 13419031 12011     10287     0    vec3.Vec3 vec3.Vec3.opBinary!("{}",).opB
621 12866509 9584      6947      0    double vec3.DotProduct(const(vec3.Vec3
622 10279720 34363    6823      0    bool sphere.Sphere.Hit(ray.Ray, double
623 6814276 5462     4708      0    vec3.Vec3 vec3.Vec3.opBinary!("{}",).opBi
624 35995879 4336     3747      0    const bool vec3.Vec3.IsZero()
625 6498806 3946     3466      0    vec3.Vec3 vec3.Vec3.opBinaryRight!("{}",)
626 2570181 73278    2032      0    vec3.Vec3 main.CastRay(ray.Ray, sphere.
627 20559440 4289     1867      0    const double vec3.Vec3.LengthSquared()
628 84971600 1543     1543      0    pure nothrow @nogc @trusted bool core.
```

17

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

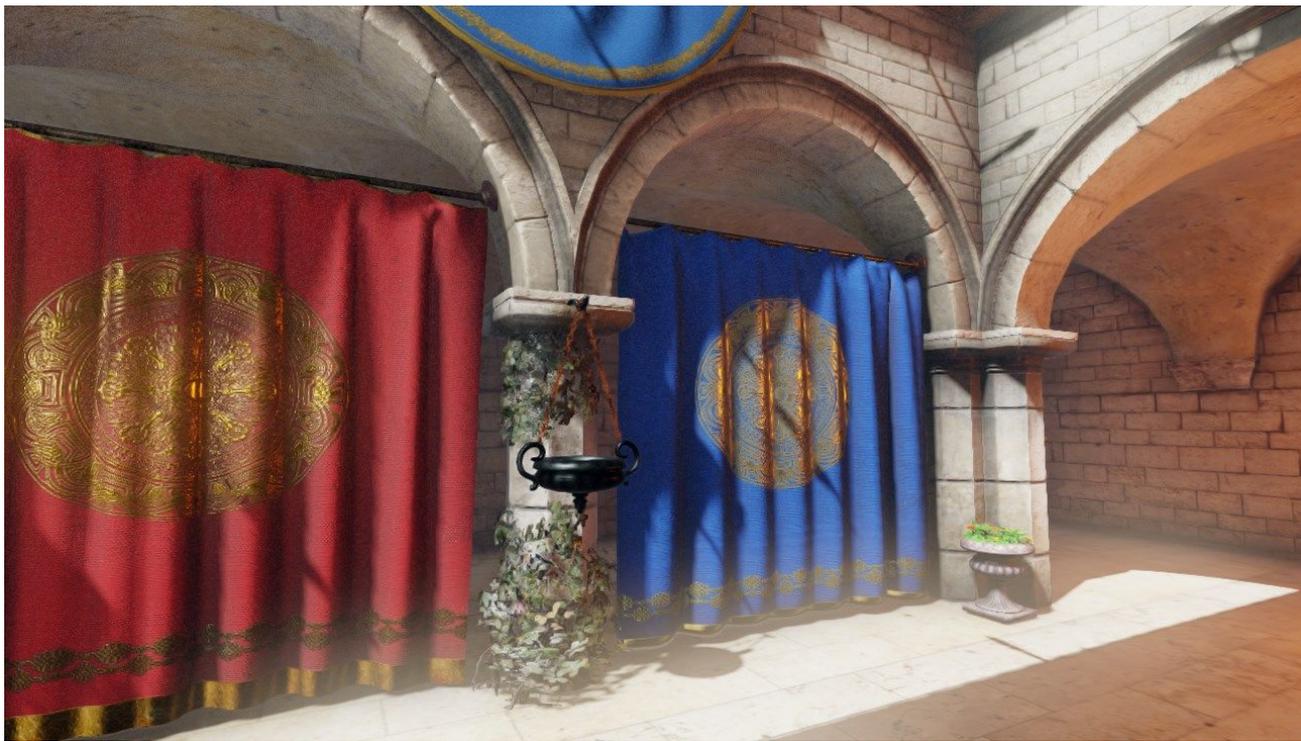
```
dmd -g -profile=gc ./src/*.d -of=prog
```

- Now notice there are no allocations for Vec3!
 - They're all done on the stack -- so let's do another speed test!

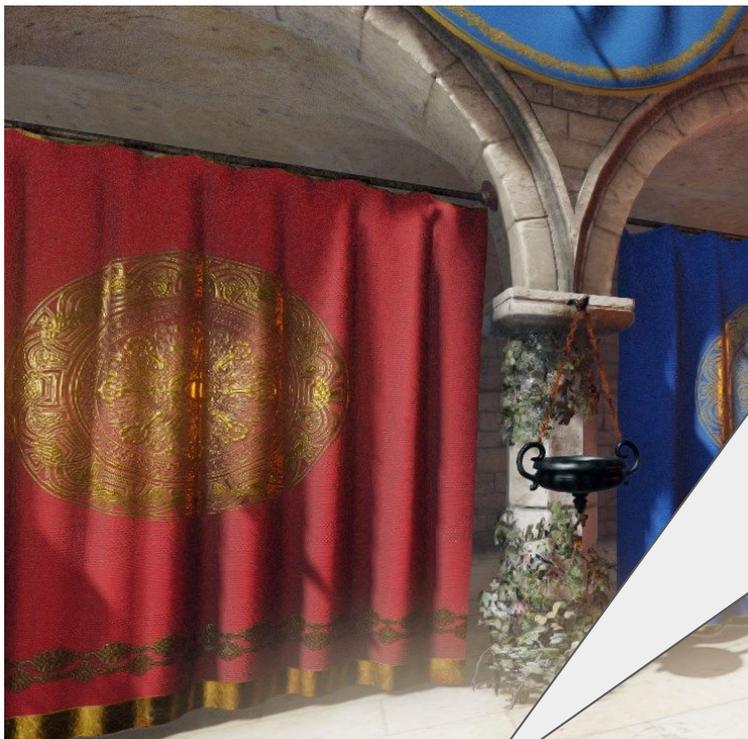
```
1 bytes allocated, allocations, type, function, file:line
2 993941664 10353559 sphere.HitRecord main.CastRay ./src/main.d:23
3 993839232 10352492 sphere.HitRecord sphere.HittableList.Hit ./src/sphere.d:44
4 288000000 4500000 ray.Ray camera.Camera.GetCameraRay ./src/camera.d:33
5 227915392 3561178 ray.Ray material.Lambertian.Scatter ./src/material.d:27
6 146712384 2292381 ray.Ray material.Metal.Scatter ./src/material.d:46
```

32

And more graphics open-source projects...



- 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>



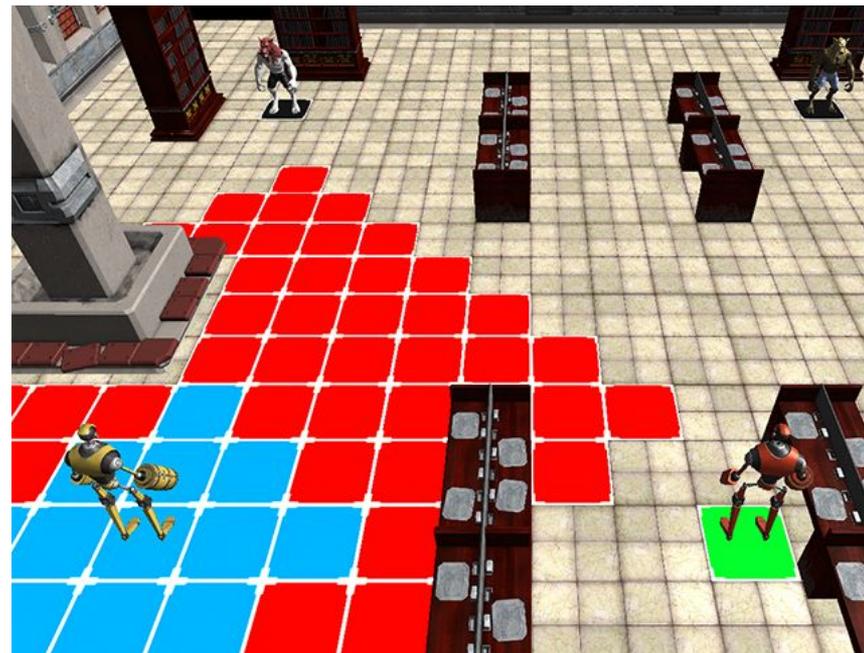
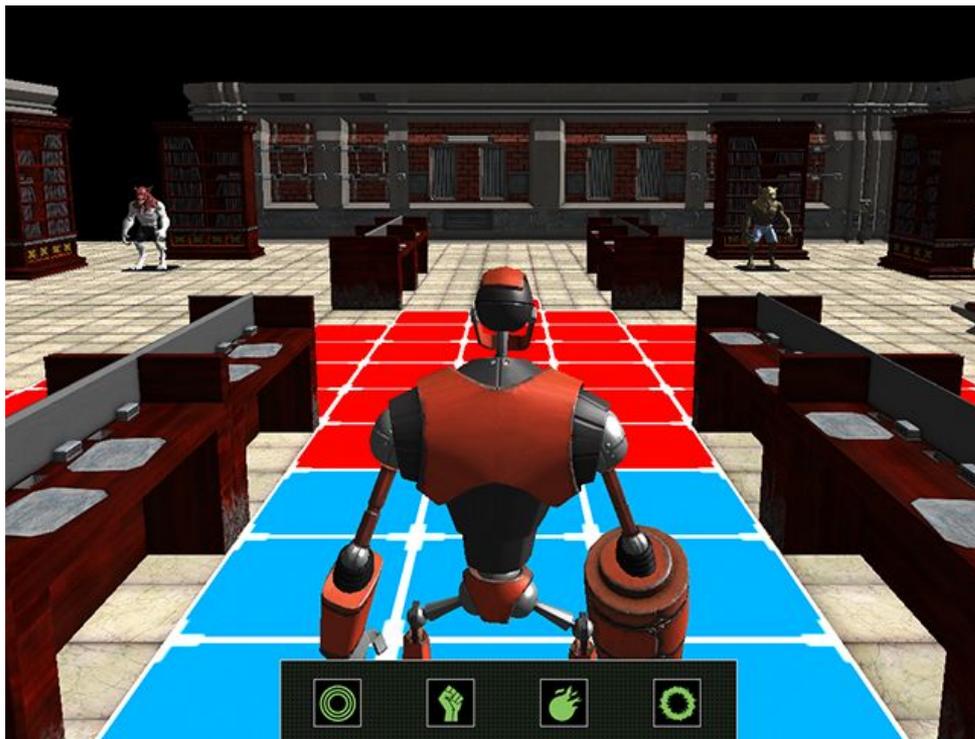
Why you might care to look?

- It's a substantial project that would be of interest to graphics developers
- You can take a look at the project hierarchy to see how a D project is organized.
- Fun comparison of C++ and D renderers [\[here\]](#)

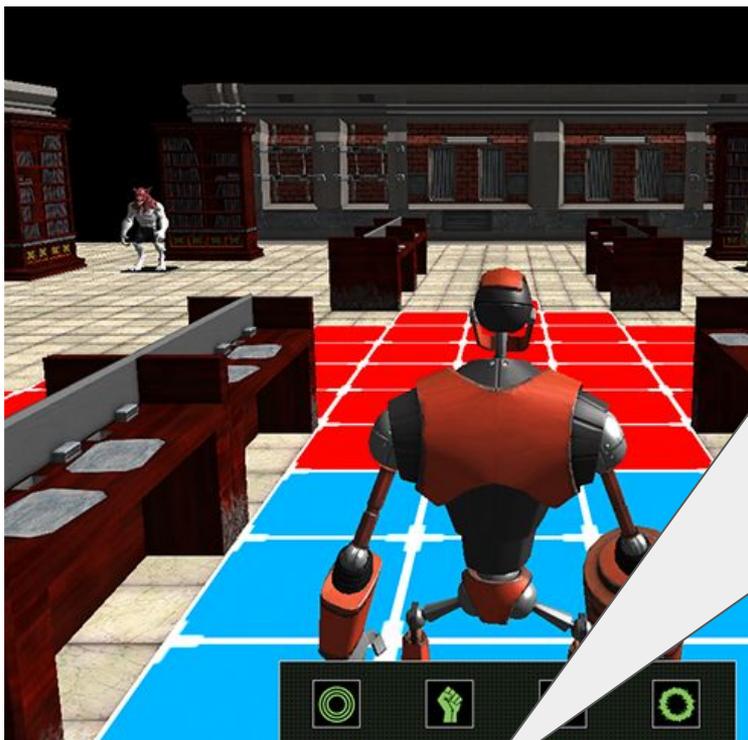
<https://github.com/gecko0307/dagon/blob/master/src/dagon/graphics/mesh.d>

```
28 module dagon.graphics.mesh;
29
30 import std.math;
31 import std.algorithm;
32
33 import dlib.core.memory;
34 import dlib.core.ownership;
35 import dlib.geometry.triangle;
36 import dlib.math.vector;
37 import dlib.geometry.aabb;
38
39 import dagon.core.bindings;
40 import dagon.graphics.drawable;
41
42 enum VertexAttrib
43 {
44     Vertices = 0,
45     Normals = 1,
46     Texcoords = 2
47 }
```

- 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>



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



Why you might care to look?

- Just to show another game engine that had been done in D!
- The code shows embedding shaders as strings -- there's also nice examples of '[mixins](#)' in the codebase.

<https://github.com/Circular-Studios/Dash/blob/develop/source/dash/graphics/shaders/glsl/ambientlight.d>

```
module dash.graphics.shaders.glsl.ambientlight;
import dash.graphics.shaders.glsl;

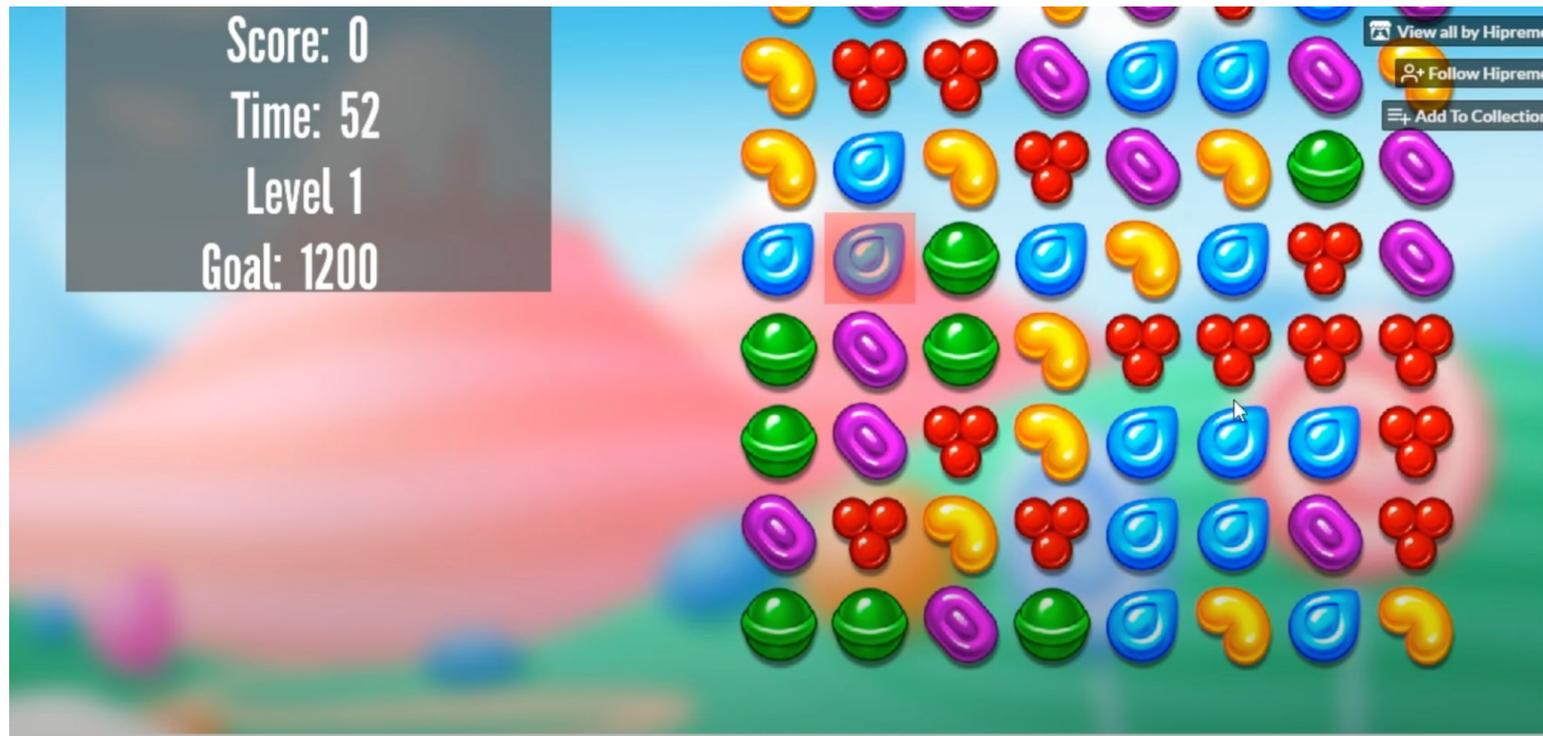
package;

// Takes in a clip-space quad and interpolates the UVs
immutable string ambientlightVS = glslVersion ~ q{
    layout(location = 0) in vec3 vPosition_s;
    layout(location = 1) in vec2 vUV;

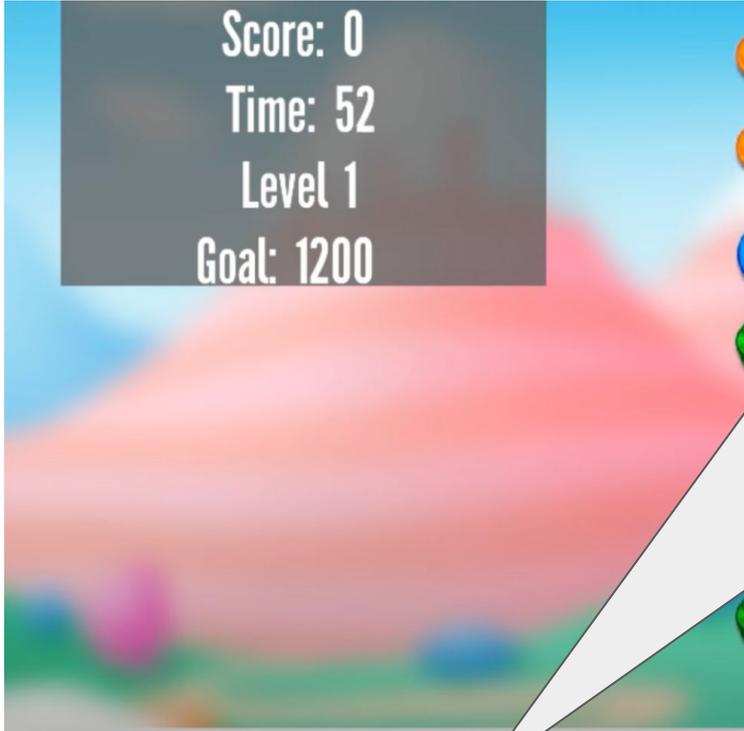
    out vec4 fPosition_s;
    out vec2 fUV;

    void main( void )
    {
        fPosition_s = vec4( vPosition_s, 1.0f );
        gl_Position = fPosition_s;
        fUV = vUV;
    }
};
```

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



- Github or Dub Repository: <https://github.com/MrcSnm/HipremeEngine>
- DConf 2023 Talk: [DConf '23 -- Hipreme Engine: Bringing D Everywhere -- Marcelo Mancini](#)



Score: 0
Time: 52
Level 1
Goal: 1200

Why you might care to look?

- Example of how to build a tool that builds on several platforms
- Some example of how to replace Druntime

<https://github.com/MrcSnm/HipremeEngine>

- Xbox Series (UWP): [build/uwp](#)
- Android: [build/android/](#)
- Browser (WebAssembly): [build/wasm](#)
- PS Vita: [build/vita](#)
- MacOS : [build/appleos](#)
- iOS: [build/appleos](#)
- Windows: [bin/desktop](#)
- Linux: [bin/desktop](#)

- Github or Dub Repository: <https://github.com/MrcSnm/HipremeEngine>
- DConf 2023 Talk: [DConf '23 -- Hipreme Engine: Bringing D Everywhere -- Marcelo Mancini](#)

(Aside) Other Graphics Resources

- The bind-bc libraries by Michael (Mike) Parker provide access to libraries like Simple Directmedia layer (SDL) and other graphical libraries to enable much of this game work.

bindbc-sdl **1.4.5**

Static & dynamic bindings to SDL2 & the SDL_* libraries, compatible with BetterC, @nogc, and nothrow.



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

```
dub add bindbc-sdl
```



<https://code.dlang.org/packages/bindbc-sdl>

(Aside) Commercial Games with D Language

- D has also been used in AAA commercial games
 - Full presentations here:
 - Using D Alongside a Game Engine
 - https://dconf.org/2013/talks/evans_1.html
 - Quantum Break: AAA Gaming With Some D Code
 - <https://dconf.org/2016/talks/watson.html>
 - D: Using an Emerging Language in Quantum Break
 - <https://www.gdcvault.com/play/1023843/D-Using-an-Emerging-Language>

DLang Paradigms

- Expressiveness
 - You can write in a procedural style, oop style, functional style, generic code, parallel code using threads, fibers, simd, etc.

D supports five main programming paradigms:

- concurrent (actor model)
- object-oriented.
- imperative.
- functional.
- metaprogramming.



Wikipedia

[https://en.wikipedia.org/wiki/D_\(programming_lang...](https://en.wikipedia.org/wiki/D_(programming_language))

D (programming language) - Wikipedia

Functional Style -- no raw loops

```
1 // @ file functional_filter.d
2 import std.stdio;
3 import std.algorithm; // map
4 import std.string;
5
6 void main(){
7
8     // Loop style
9     // A little better with foreach loop
10    auto words = ["hello", "world", "dlang", "c++", "java"];
11    int coolLangauges = 0;
12    foreach(element ; words){
13        if(element=="dlang"){
14            coolLangauges++;
15        }
16    }
17    writeln("Cool langauges found: ",coolLangauges);
18
19    // Functional-style
20
21    auto words2 = ["hello", "world", "dlang", "c++", "java"];
22    import std.array;
23    auto result = words.filter!(a=> a.indexOf("dlang") >=0).array;
24    writeln("Cool langauges found: ",result);
25
26 }
```

Object-Oriented Style

```
1 // @ inheritance.d
2 import std.stdio;
3
4 interface Dog{
5     void Bark();
6     void Walk();
7 }
8
9 class Husky : Dog{
10     void Bark(){ writeln("Husky Bark!"); }
11     void Walk(){ writeln("Husky Walk!"); }
12 }
13
14 class GoldenRetriever : Dog{
15     void Bark(){ writeln("GoldenRetriever Bark!"); }
16     void Walk(){ writeln("GoldenRetriever Walk!"); }
17 }
18
19 void main(){
20
21     Dog dog1 = new Husky;
22     Dog dog2 = new GoldenRetriever;
23
24     Dog[] collection;
25     collection ~= dog1;
26     collection ~= dog2;
27     foreach(doggy ; collection){
28         doggy.Bark();
29     }
30
31 }
```

“Hello world” of meta programming/introspection

Template Constraints and introspection

```
36 // NEW: Introspection capabilities at compile-time
37 //     to ensure class has memory and elements fields.
38 void printData(T)(T theStruct)
39     if(hasMember!(T,"memory") &&
40         hasMember!(T,"elements"))
41 {
42     foreach(i ; 0 .. theStruct.elements){
43         write(theStruct.memory[i],",");
44     }
45     writeln();
46 }
```

```
> dmd -betterC -unittest -run test.d
```

40.3 Retained Features

1. Nearly the full language remains available. Highlights include:

1. Unrestricted use of compile-time features
2. Full metaprogramming facilities
3. Nested functions, nested structs, delegates and [lambdas](#)
4. Member functions, constructors, destructors, operating overloading, etc.
5. The full module system
6. Array slicing, and array bounds checking
7. RAII (yes, it can work without exceptions)
8. `scope(exit)`
9. Memory safety protections
10. [Interfacing to C++](#)
11. COM classes and C++ classes
12. `assert` failures are directed to the C runtime library
13. `switch` with strings
14. `final switch`
15. `unittest`
16. [printf format validation](#)

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

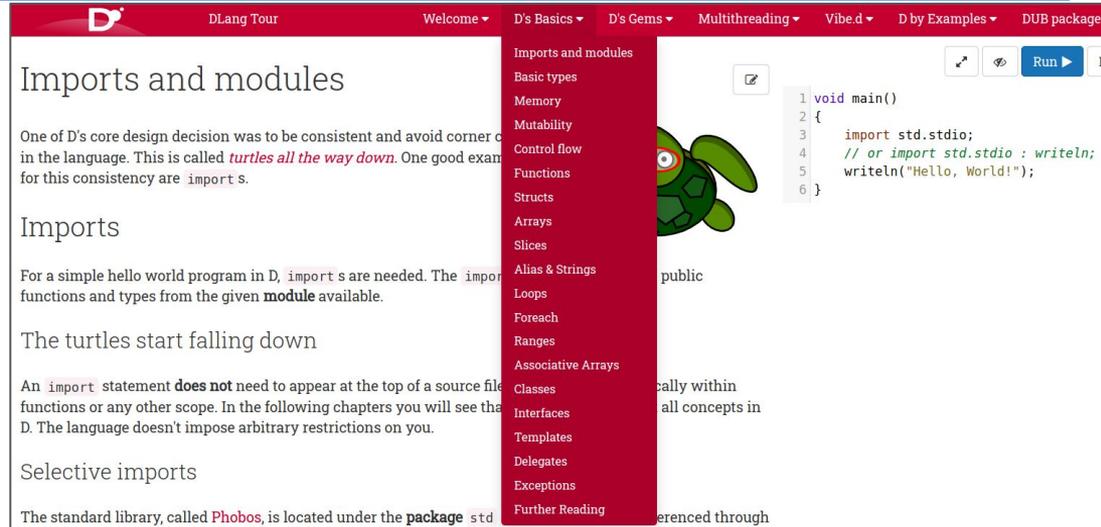
Not ready to try D?

- Use it as a 'betterC'
 - Useful for bare-metal programming or enhancing a C-codebase.
 - disables D language run-time, so reduces dependencies
 - Get other features of the D language I have not talked about
 - e.g. `unittest` support
 - e.g. RAII support
 - e.g. Excellent metaprogramming support
 - compile-time functionality remains
- Nice talk on bare metal programming on kernels here:
 - [DConf '23--Multiplex: Using D for Kernel Development--Zachary Yedidia](#)

Learning More About the D Language

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 cases in the language. This is called *turtles all the way down*. One good example for this consistency are `import` s.

Imports

For a simple hello world program in D, `import` s are needed. The `import` 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, functions or any other scope. In the following chapters you will see that in D. The language doesn't impose arbitrary restrictions on you.

Selective imports

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

```
1 void main()
2 {
3     import std.stdio;
4     // or import std.stdio : writeln;
5     writeln("Hello, World!");
6 }
```

<https://tour.dlang.org/>

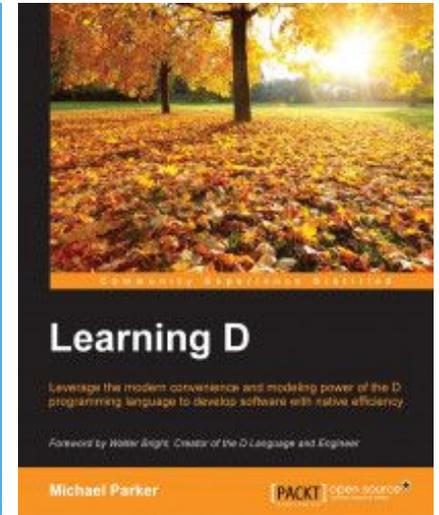
More Resources for Learning D

I would start with these two books

1. Programming in D by Ali Çehreli
 - a. Freely available <http://ddili.org/>
2. Learning D by Michael Parker

Any other books you find on D are also very good -- folks in the D community write books out of passion!

The online forums and discord are otherwise very active



YouTube

- I am actively adding more lessons about the D programming language
 - <https://www.youtube.com/c/MikeShah>

[Episode 0] Series Teaser

matrix.py
matrix.d
DLang

D Language (DLang) Programming

Mike Shah

Public

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

Play all Shuffle

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

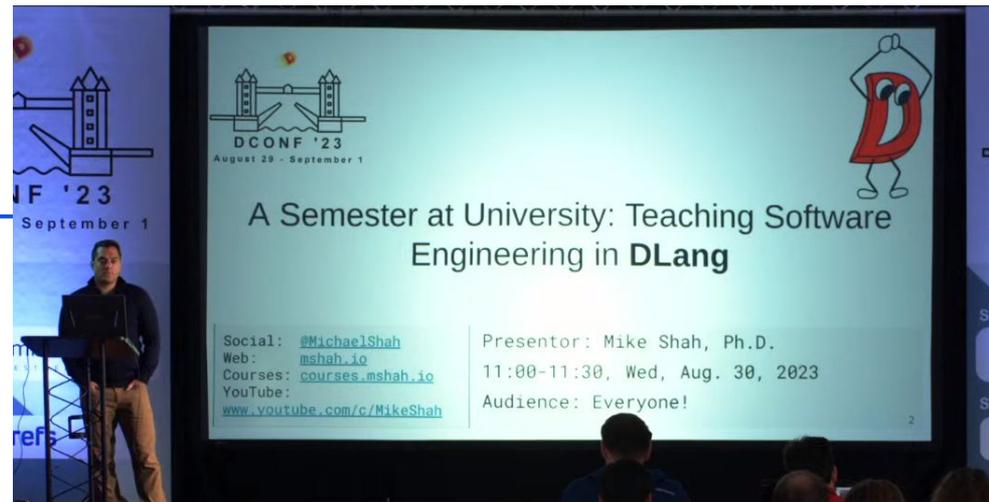
Sort All Videos Shorts

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

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

Teaching D Language

- You can hear my perspective
- **Even better** -- you can hear the students perspective
 - They built a networked collaborative paint program that is also available.
- D Conf 2023:
 - YouTube:
<https://www.youtube.com/live/wXTlafzIJVY?si=Xpy6g5h4wtIUrt2E&t=7711>
 - Link to Conference Talk Description:
<https://dconf.org/2023/index.html>



The Case for D [[link](#)] [[archived link](#)] (1/2)

- Andrei Alexandrescu [[wiki](#)] one of the main contributors to has a wonderful article on “The Case for D” written in 2009.
 - In short, D is a ‘high-level systems language’ where you can be productive, and enjoy coding....*Of course, I'm not deluding myself that it's an easy task to convince you.*

D Fundamentals

D could be best described as a high-level systems programming language. It encompasses features that are normally found in higher-level and even scripting languages -- such as a rapid edit-run cycle, garbage collection, built-in hashtables, or a permission to omit many type declarations -- but also low-level features such as pointers, storage management in a manual (' la C's **malloc/free**) or semi-automatic (using constructors, destructors, and a unique scope statement) manner, and generally the same direct relationship with memory that C and C++ programmers know and love. In fact, D can link and call C functions directly with no intervening translation layer. The entire C standard library is directly available to D programs. However, you'd very rarely feel compelled to go that low because D's own facilities are often more powerful, safer, and just as efficient. By and large, D makes a strong statement that convenience and efficiency are not necessarily at odds. Aside from the higher-level topics that we'll discuss soon, no description of D would be complete without mentioning its attention to detail: all variables are initialized, unless you initialize them with **void**; arrays and associative arrays are intuitive and easy on the eyes; iteration is clean; NaN is actually used; overloading rules can be understood; support for documentation and unit testing is built-in. D is multi-paradigm, meaning that it fosters writing code in object-oriented, generic, functional, and procedural

The Case for D [[link](#)] [[archived link](#)] (2/2)

- Andrei Alexandrescu [[wiki](#)] one of the main contributors to has a wonderful article on “The Case for D” written in 2009.

- In short, D is a ‘high-level systems language’ where you can be productive, and enjoy coding.

- Again, you’ll decide yourself after trying if D is your new language of choice.
- My hope -- In this talk, I can at the least show you some great features of D, and where to look for inspiration for D in the open source world.

D Fundamentals

D could be best described as a high-level systems programming language. It encompasses features that are normally found in higher-level and even scripting languages -- such as a rapid edit-run cycle, garbage collection, built-in hashtables, or a permission to omit many type declarations -- but also low-level features such as pointers, storage management in a manual (‘la C’s **malloc/free**) or semi-automatic (using constructors, destructors, and a unique scope statement) manner, and generally the same direct relationship with memory that C and C++ programmers know and love. In fact, D can link and call C functions directly with no intervening translation layer. The entire C standard library is directly available to D programs. However, you’d very rarely feel compelled to go that low because D’s own facilities are often more powerful, safer, and just as efficient. By and large, D makes a strong statement that convenience and efficiency are not necessarily at odds. Aside from the higher-level topics that we’ll discuss soon, no description of D would be complete without mentioning its attention to detail: all variables are initialized, unless you initialize them with **void**; arrays and associative arrays are intuitive and easy on the eyes; iteration is clean; NaN is actually used; overloading rules can be understood; support for documentation and unit testing is built-in. D is multi-paradigm, meaning that it fosters writing code in object-oriented, generic, functional, and procedural

So why care as an open source developer?

- I've found D to be:
 - Readable
 - Writeable
 - Performant
 - Allow fast iteration times
 - This combination of attributes provides a competitive advantage
 - I **believe** based on working with students, that D-based projects are very easy to have contributors at different skill levels participate at scale.
- The ecosystem of D Compilers is very open, so no worry about D disappearing
- Overall:
 - A friendly language, allowing you to work at many different levels and paradigms, could be a wonderful way to build software and collaborate with others

What's next for me?

- Converting my website to use the vibe framework
 - See: <https://vibed.org/>
- Yet another open-source tool in the ecosystem for building scalable websites and web applications.

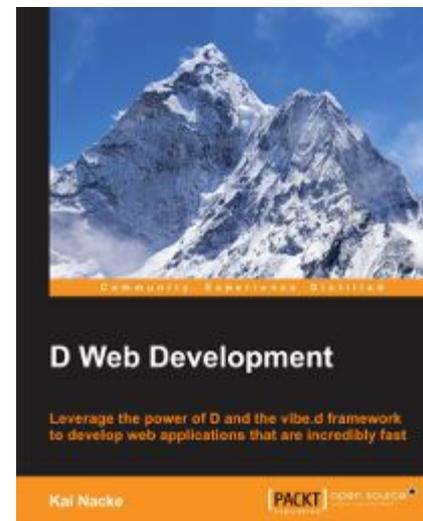


get **vibe.d** 0.9.7

Asynchronous I/O that doesn't get in your way, written in D

Fork me on GitHub

Productive	Fast	Simple
High-level declarative REST and web application framework	Asynchronous I/O for maximum speed and minimum memory usage	Fiber based blocking programming model for concise and intuitive development
Full HTTP(S) stack with client, server and proxy implementations	Compile-time "Diet" templates for unparalleled dynamic page speed	Compact API with sensible default choices
Shipped with native database drivers for MongoDB and Redis	Compiled to native machine code	Full support for exception based error handling
Complete concurrency toolkit and support for low level I/O operations	Multi-threading and integrated load-balancing*	Simple access to third-party extension libraries using the DUB package system
Read more...	Read more...	Read more...



Some Summary of D Topics Today

- It is a compiled language
 - (i.e. machine code is executed as opposed to interpreting code)
- The compilers (DMD, LDC2, GDC) have years of optimization built into them
- D does lots of compile-time function evaluation (CTFE)
 - Run code at compile-time, so you don't need to evaluate at run-time
- The language allows you to control system resources
 - i.e. You can turn on and off garbage collection for example.
- Parallelization can often be trivially enabled (e.g. `std.parallel`)
- Universal Function Call Syntax (UFCS) for writing readable code
- `rdmd` gives you a 'script like' feel to the language when you need
 - Keep all of your code and cognitive load in one programming language



Thank you Fosdem 2024!

The **D** Programming Language for Modern Open Source Development

-- Programming in DLang
with Mike Shah

16:00 - 16:50 Sat, Feb 3, 2024
Location: k.1.105 (La Fontaine)
50 minutes | Introductory Audience

Social: [@MichaelShah](https://twitter.com/MichaelShah)
Web: mshah.io
Courses: courses.mshah.io
 **YouTube**
www.youtube.com/c/MikeShah
<http://tinyurl.com/mike-talks>

Thank you!

Errata/Questions

Questions and notes after the talk

- **Questions during the talk**

- Rust vs D
 - I have not used Rust professionally to comment on a large code base, but here are some thoughts.
 - Probably each have their own domains
 - I've found D code very 'malleable' (i.e. high plasticity) which may be an advantage
 - Anyone with a C, C++, Java background I suspect will have an easy transition to D
 - For game/graphics (my domain) or other highly stateful applications I've found D great!
 - For systems programming both are good languages with memory safety features
 - For experts in either 'Rust' or 'D', the old advice probably applies where you pick the language you are most comfortable in, and that's the language you'll like best.

- **Questions after the talk**

- **pure** is available in D, so you can define pure functions
 - Useful for concurrency, minimizing state, improving chance of compile-time function
- Regarding the 'template constraints' here's the page
 - <https://dlang.org/articles/constraints.html>
 - I *believe* no need to write 'static if' because constraint is evaluated at compile-time, but you could put in a static if
 - I also did not discuss 'pre' and 'post' contracts used when developing software -- which is another nice feature
-

Extras and Notes

More Useful Links

- <https://github.com/dlang-community/awesome-d>
- Another list of projects and companies (here: <https://github.com/dlang-community/awesome-d?tab=readme-ov-file#organizations>) using D now or in the past.
- D repositories
 - <https://github.com/topics/dlang> and/or <https://github.com/topics/d>