Dhruv Warrier

dhruvwarrier.github.io | github.com/dhruvwarrier/ | linkedin.com/in/dhruvwarrier
dhruv.warrier89@gmail.com | (647) 928 7960 | 89 Chestnut St. Toronto ON M5G1R1

Education

University of Toronto St. George, BASc Computer Engineering, SEP '17 – APR '21
27 King's College Cir, Toronto, ON M5S 3H7, Canada, +1 416-978-2011
Courses taken: Algorithms and Data Structures, Operating Systems, Computer Hardware, Computer Graphics, Probability and Applications, Probabilistic Reasoning

Skills

Languages: C# 7.3, C++11, C89/ANSI C, C11, Verilog+Tcl, C# Mono
Libraries: STL, Boost, GTK, OpenGL, WPF, Winforms, DevExpress, .NET 4.7.0-4.7.2, Eyeshot
Environments: Microprocessor emulation, FPGA solution design, Desktop app development

Software Engineering Intern

MAY 2019 – AUGUST 2019

Rocscience Inc., Toronto

Developed 3D contouring and visualization tools, and migrated graphing and UI architecture to new product.

- Reimplemented underlying graphing architecture, for easier ports to future products.
- Redesigned underlying architecture for polyline editing tool across products. Extended it to allow instantiation of a "headless" editor (untied to a 3D object), reducing duplicate 3D objects by **50%.**
- Researched performance of the WPF UI binding system and developed a prototype to post-process ObservableCollection (UI) notifications. This was able to improve UI responsiveness by up to **70%** on certain large workflows with List<T> (array/vector) data collections.

Research Assistant

DEC 2017 – JAN 2019

Dr. Tamer Diraby, Dept. of Civil Engineering, University of Toronto — github.com/dhruvwarrier/city-builder

Worked on software tools for civil engineers using insights from Dr. Diraby's research. I co-developed "city-builder", a cross-platform 3D tool to help civil engineers design roads and cities.

- Developed a JSON-based file format (.city) to describe cities, roads, and lanes.
- Developed an API to create and manipulate lanes and roads without direct calls to the 3D renderer.

intel8080-emulator

github.com/dhruvwarrier/intel8080-emulator — C89/ANSI C

An emulator of the Intel8080 microprocessor, with asynchronous interrupts and partial emulation of the CP/M 2.2 BIOS. Written out of curiosity for how microprocessors work, and as a launchpad for more complicated emulators! After completing the CP/M 2.2 BIOS, I intend to use it to "natively" run a BASIC interpreter and the popular text based Zork game series, from their original source code.

- Implements all instructions, documented and undocumented.
- Tested with the Kelly Smith and Supersoft Diagnostic tests originally written for the 8080.
- Provides partial BIOS support for 8080 binaries targeting the CP/M 2.2 operating system.
- Aims for portability: Written in C89/ANSI C, builds on MSVC>=Windows XP, clang on OSX, POSIX environments>=199506, and WebAssembly.

raycast-3D-CycloneFPGA

github.com/dhruvwarrier/ raycast-3D-CycloneFPGA — Verilog, C++

A 3D ray-casting engine for Cyclone V FPGAs written in Verilog. Tested on a Terasic DE1_SoC prototype board.

- Designed a parallel architecture to compute 3D ray intersections simultaneously, with a lower bound output of **~600 fps** at **320 rays/frame** (160x120 pixels with 3-bit color palette on VGA monitors).
- Developed DSP modules for fixed point arithmetic, and precomputed lookup tables for trigonometric ratios. The lookup tables allowed for **1-cycle** lookup with adequate precision, instead of the longer **32-cycle** higher-precision calculation through IPs from the Quartus library.

shop2go-mapping-app

C++, GTK

A desktop mapping app to help distributors plan optimal routes to deliver packages in large cities, and help consumers conveniently locate offers/deals at shops along their daily route.

- Implemented the app's search bar with a modified prefix/radix-tree and Levenshtein distance matrix, taking up to **10µs** for autocompletion from **>300,000** entity names in a city, and under **11ms** with spell-checking on. A reference iterative implementation took up to **2.5s** without spell-checking, and a simpler prefix tree implementation took up to **200µs.**
- Developed multi-threaded path pre-computation algorithm to calculate all paths between **~400 nodes** (**159,600 paths**) in **3-5s** for the average city map, and under **13s** for all city maps. A previous implementation with the A* algorithm took on average **~120s** for 400 nodes.