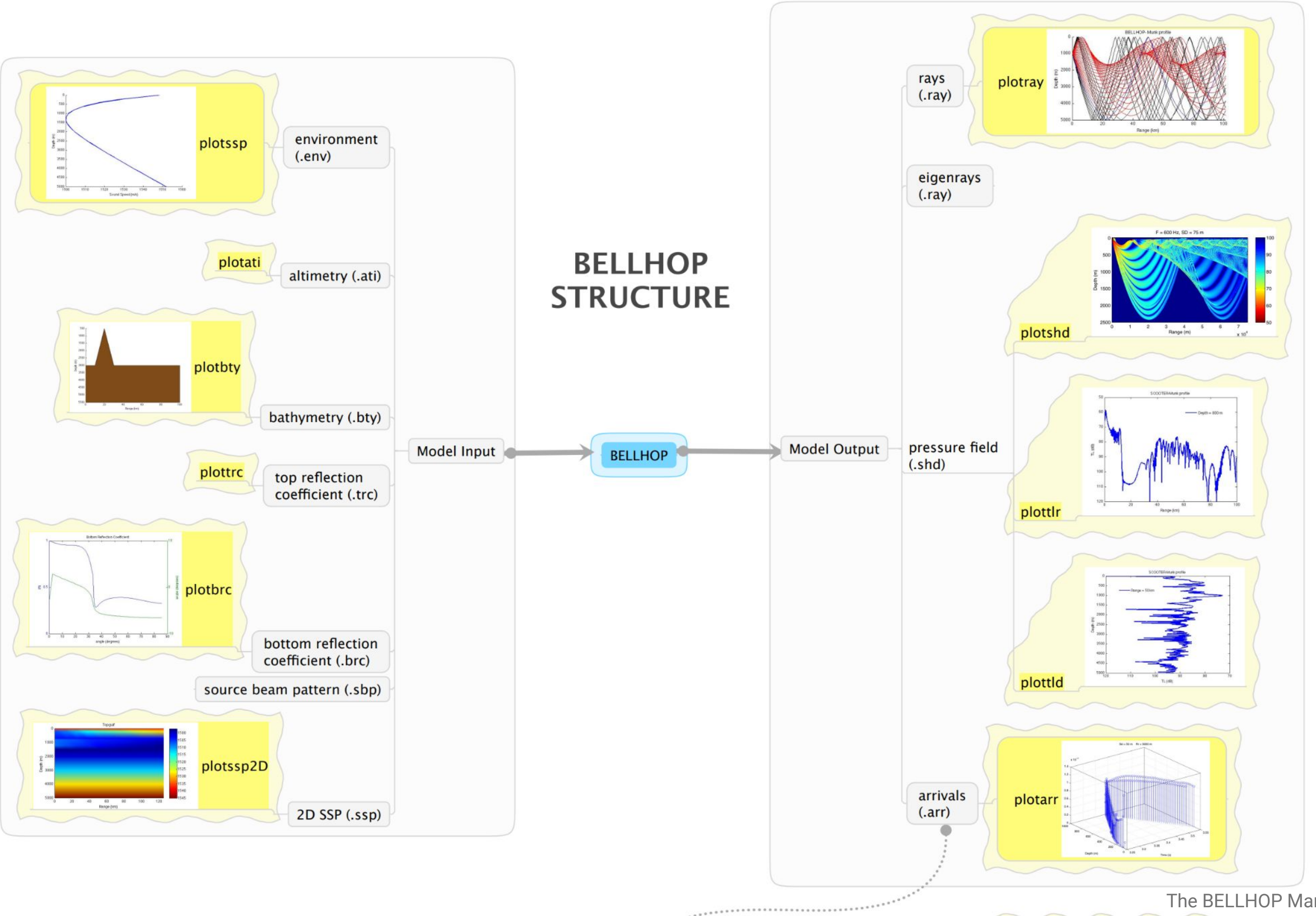
Interactive Underwater Acoustic Simulator with Ray Tracing

Ashton Palacios and Chris Kitras


BELLHOP STRUCTURE
plotssp
environment (.env)
plotati
altimetry (.ati)
plotbty
bathymetry (.bty)
Model Input
plottrc
top reflection coefficient (.trc)
plotbrc
bottom reflection coefficient (.brc)
source beam pattern (.sbp)
plotssp2D
2D SSP (.ssp)
BELLHOP
Model Output
rays (.ray)
plotray
eigenrays (.ray)
plotshd
pressure field (.shd)
plottlr
plottld
arrivals (.arr)
plotarr

BELLHOP Acoustic Toolbox

Underwater acoustic propagation modeling with arlpy and Bellhop

The underwater acoustic propagation modeling toolbox (`uwapm`) in `arlpy` is integrated with the popular Bellhop ray tracer distributed as part of the acoustics toolbox. In this notebook, we see how to use `arlpy.uwapm` to simplify the use of Bellhop for modeling.

Prerequisites

- Install arlpy (v1.5 or higher)
- Install the acoustics toolbox (6 July 2018 version or later)

Getting started

Start off with checking that everything is working correctly:

In [1]:
```
import arlpy.uwapm as pm
import arlpy.plot as plt
import numpy as np
```

In [2]:
```
pm.models()
```

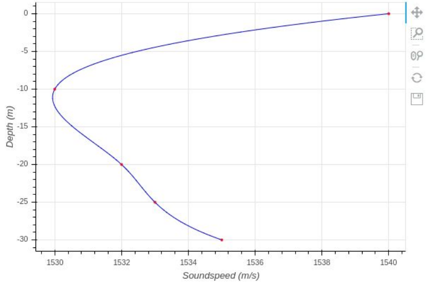
Out[2]:
```
['bellhop']
```

The `bellhop` model should be listed in the list of models above, if everything is good. If it isn't listed, it means that `bellhop.exe` is not available on the PATH, or it cannot be correctly executed. Ensure that `bellhop.exe` from the acoustics toolbox installation is on your PATH (updated `.profile` or equivalent, if necessary, to add it in).

From here on we assume that the `bellhop` model is available, and proceed...

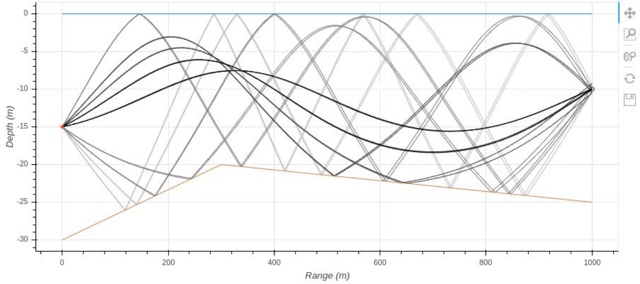
We next create an underwater 2D environment (with default settings) to model:

In [14]:
```
pm.plot_ssp(env)
```

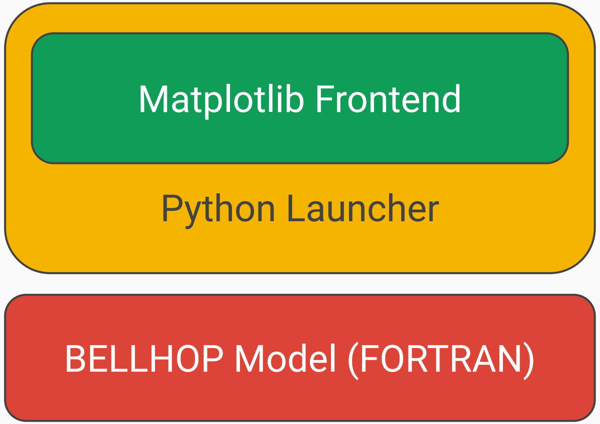


Looks more interesting! Let's see what the eigenrays look like, and also the arrival structure:

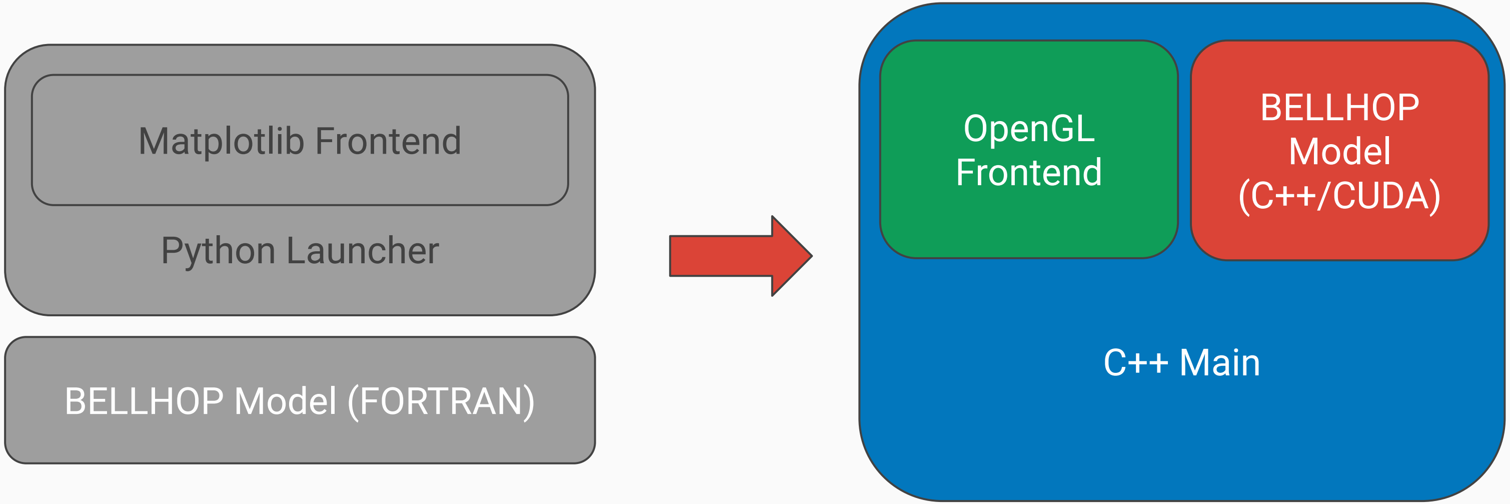
In [15]:
```
rays = pm.compute_eigenrays(env)
pm.plot_rays(rays, env=env, width=900)
```



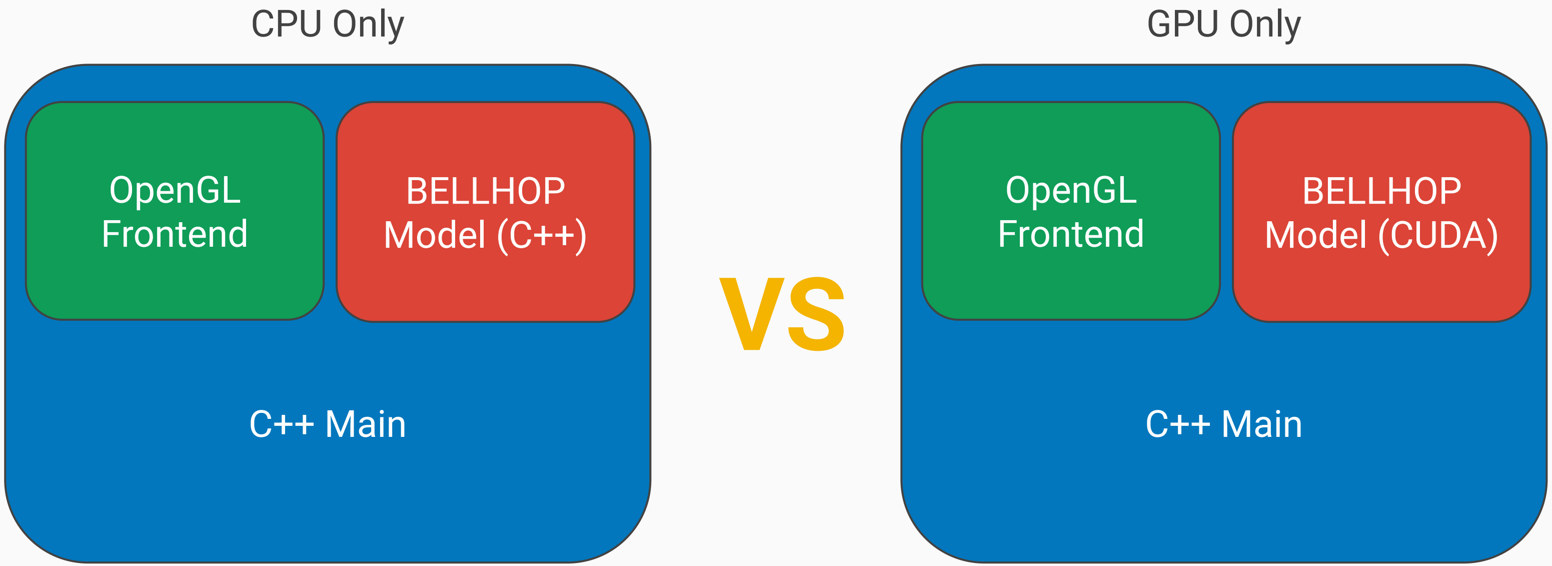
Our Implementation



Our Implementation



Our Implementation



BELLHOP Hacking

README GPL-3.0 license

bellhopcxx / bellhopcuda

C++/CUDA port of `BELLHOP` / `BELLHOP3D` underwater acoustics simulator.

Impressum

Copyright (C) 2021-2023 The Regents of the University of California
Marine Physical Lab at Scripps Oceanography, c/o Jules Jaffe, jjaffe@ucsd.edu
Based on BELLHOP / BELLHOP3D, which is Copyright (C) 1983-2022 Michael B. Porter

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see https://www.gnu.org/licenses/.

FAQs

What is C++/CUDA?

This is a single codebase which can be built as multithreaded C++ code for your CPU, or as CUDA code for your NVIDIA GPU. **You can use the CPU version (bellhopcxx) even if you don't have an NVIDIA GPU.**

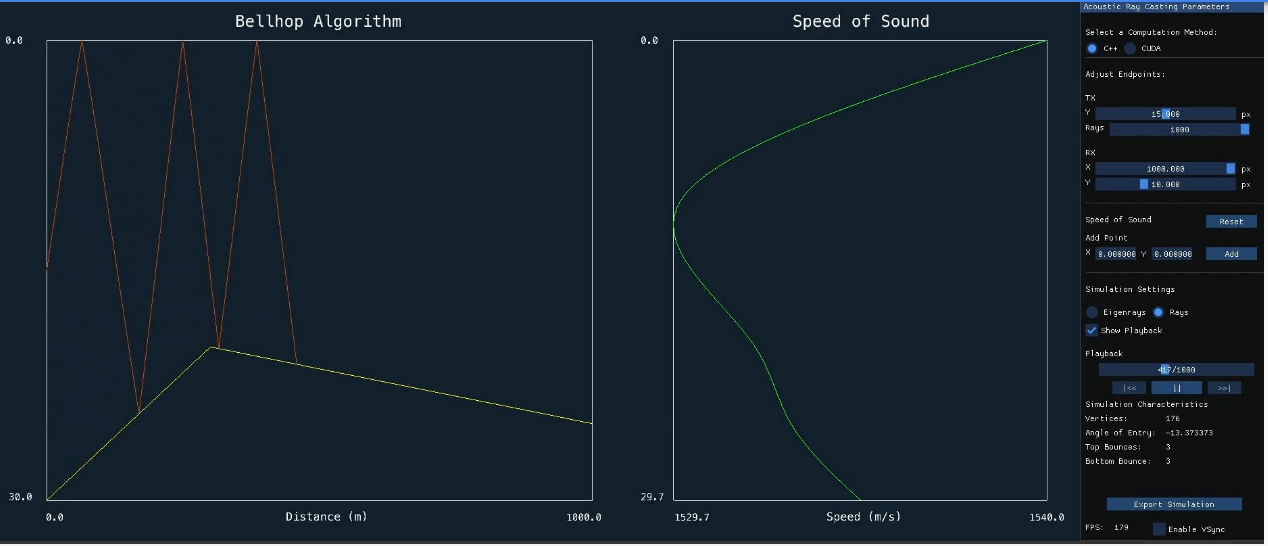
What platforms does this run on?

`bellhopcxx` is compatible with all platforms (Linux, Windows, Mac), and `bellhopcuda` is compatible with all platforms which support CUDA (Linux and Windows).

Why should I use `bellhopcxx` / `bellhopcuda` instead of `BELLHOP`?

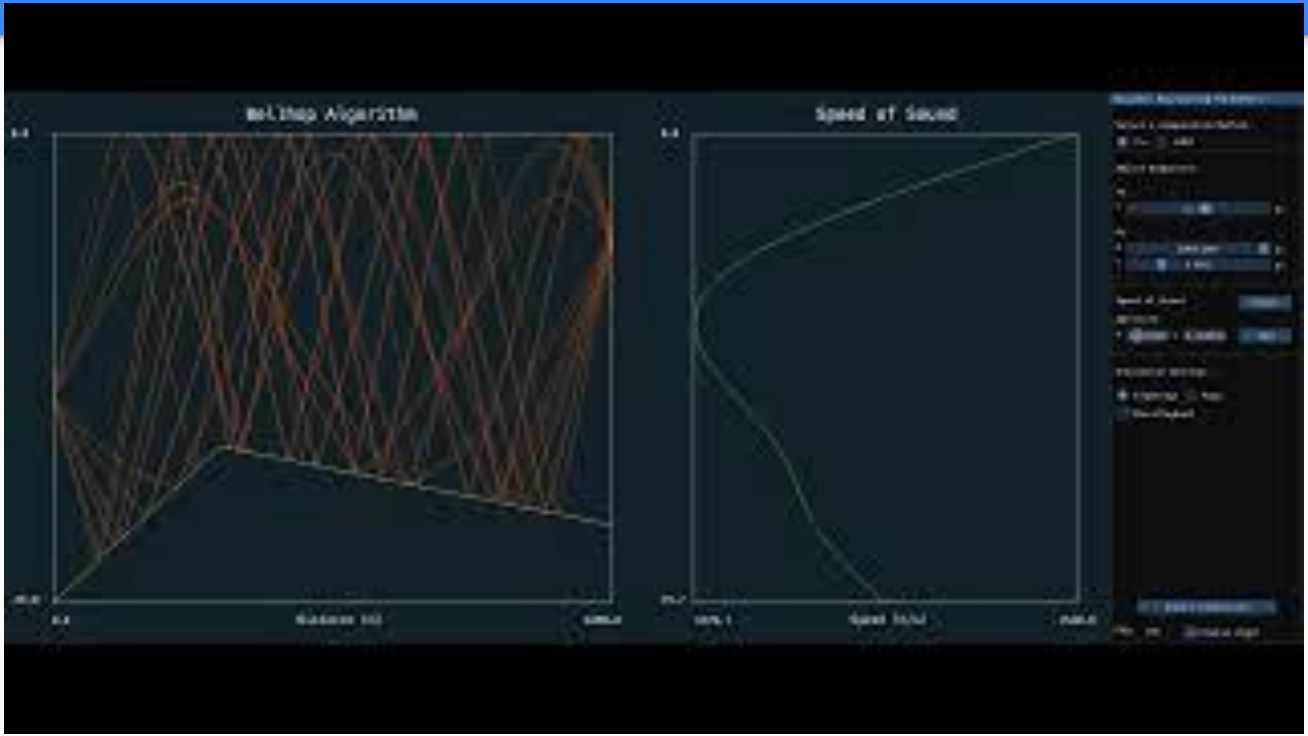
```
template<bool O3D> struct bhcParams {
    char Title[80]; // Size determined by WriteHeader for TL
    real fT;
    BdryType *Bdry;
    BdryInfo<O3D> *bdinfo;
    ReflectionInfo *refl;
    SSPStructure *ssp;
    AttenInfo *atten;
    Position *Pos;
    AnglesStructure *Angles;
    FreqInfo *freqinfo;
    BeamStructure<O3D> *Beam;
    SBPInfo *sbp;
    /// Pointer to internal data structure for program (non-marine-related) state.
    void *internal;
};

template<bool O3D, bool R3D> struct bhcOutputs {
    RayInfo<O3D, R3D> *rayinfo;
    cpxf *uAllSources;
    EigenInfo *eigen;
    ArrInfo *arrinfo;
};
```

Graphical Frontend



- OpenGL
  - Uses CPU or GPU based on system
- Vectors of Vertices
- Shaders apply color & transformation
- Buffer swapping
- ImGUI widgets

Demo



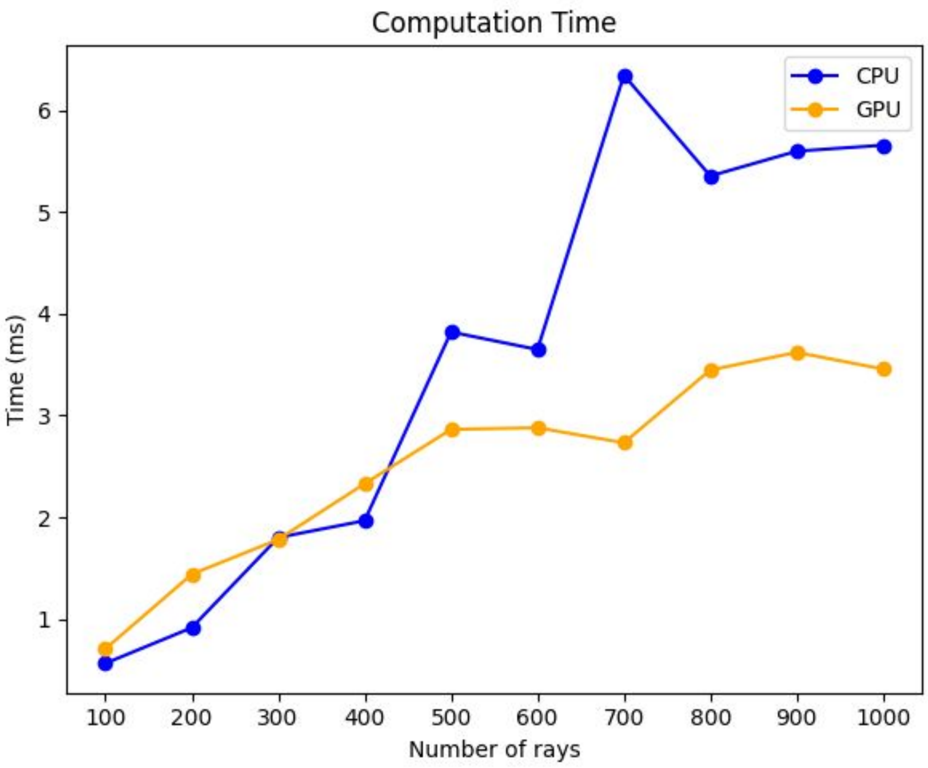
Ray Test

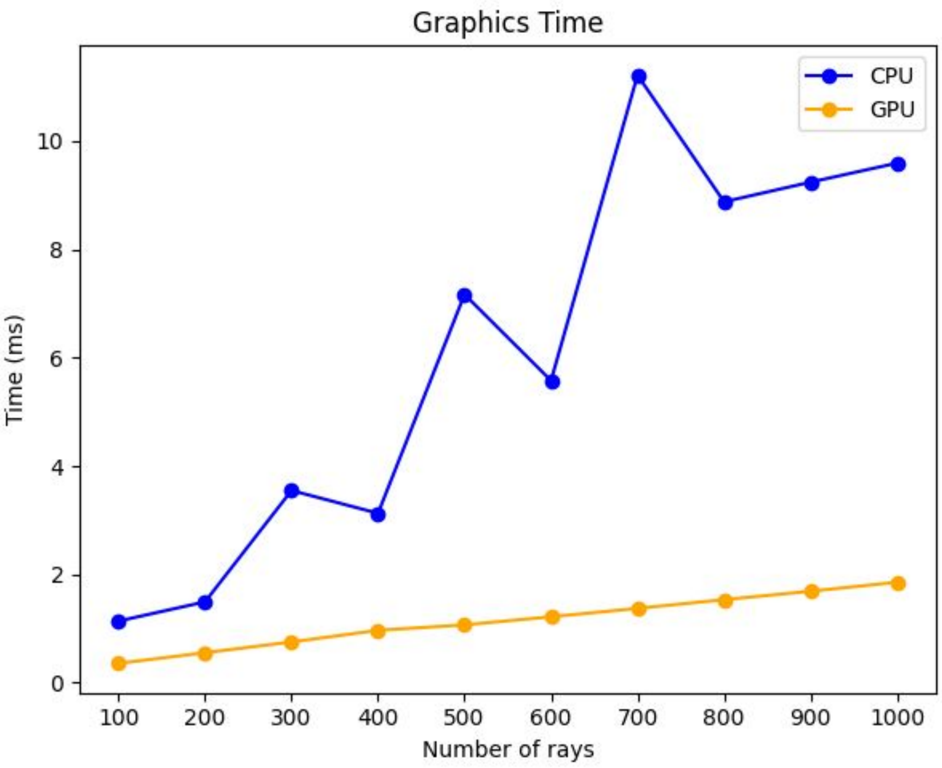
- Measure time it takes to simulate 100-1000 rays on CPU vs. GPU
- Perform a breakdown of frame rendering aspects
    - Memory Management
    - BELLHOP calculations
    - OpenGL drawing

Results: Ray Test

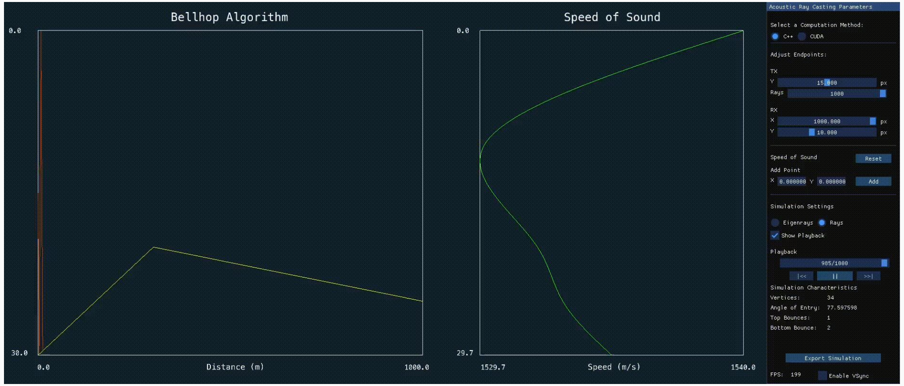
# of Rays	100	200	300	400	500	600	700	800	900	1000
Computation Speed Up	0.80	0.63	1.01	0.84	1.33	1.27	2.32	1.55	1.55	1.63
Graphics Speed Up	3.17	2.69	4.7	3.23	6.7	4.58	8.17	5.79	5.47	5.16

Results: Ray Test





Frame Breakdown Test

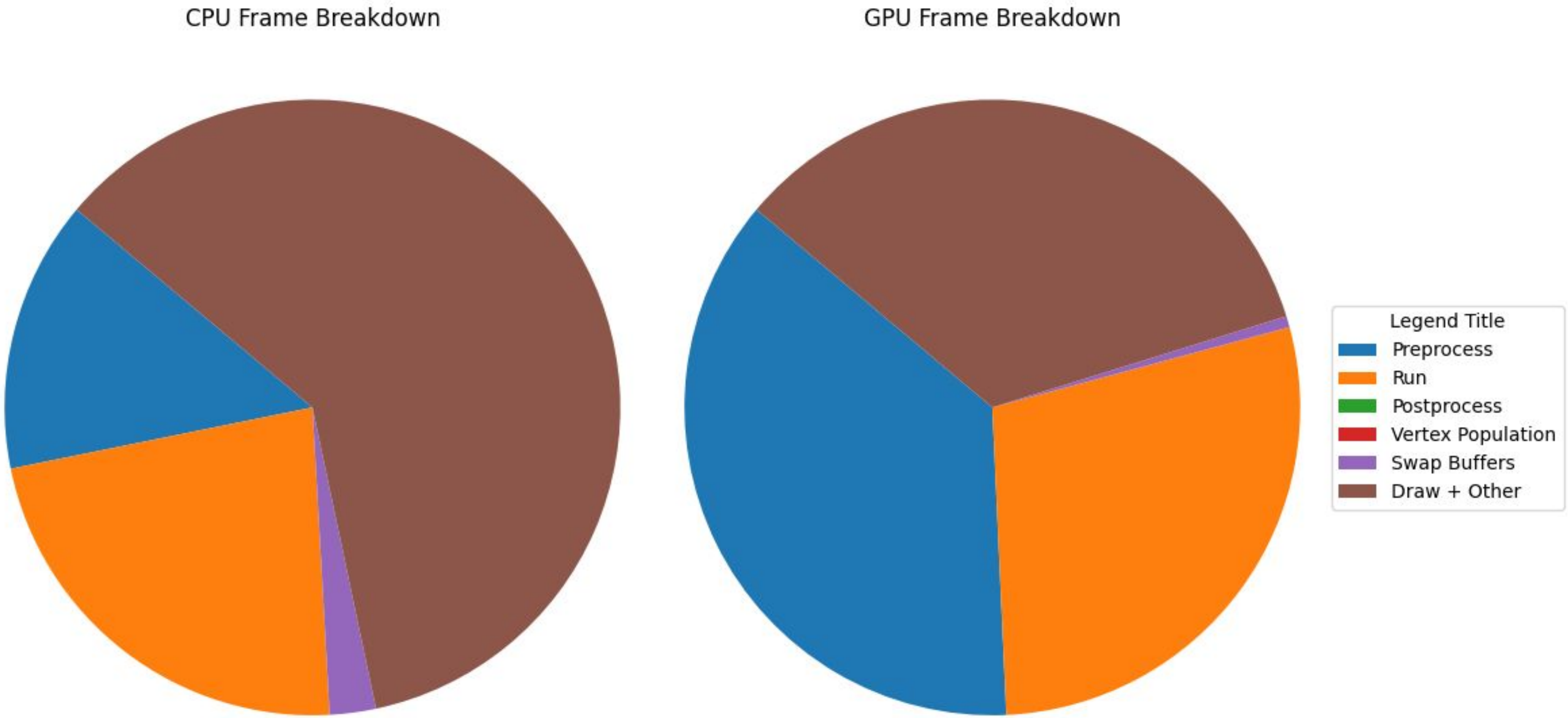


- Run program simulating 1000 rays in a preloaded environment on CPU and GPU
- Perform a breakdown of frame rendering aspects
  - Memory Management
  - BELLHOP calculations
  - Asset Preparation
  - Buffer update/swap

Results: Frame Breakdown

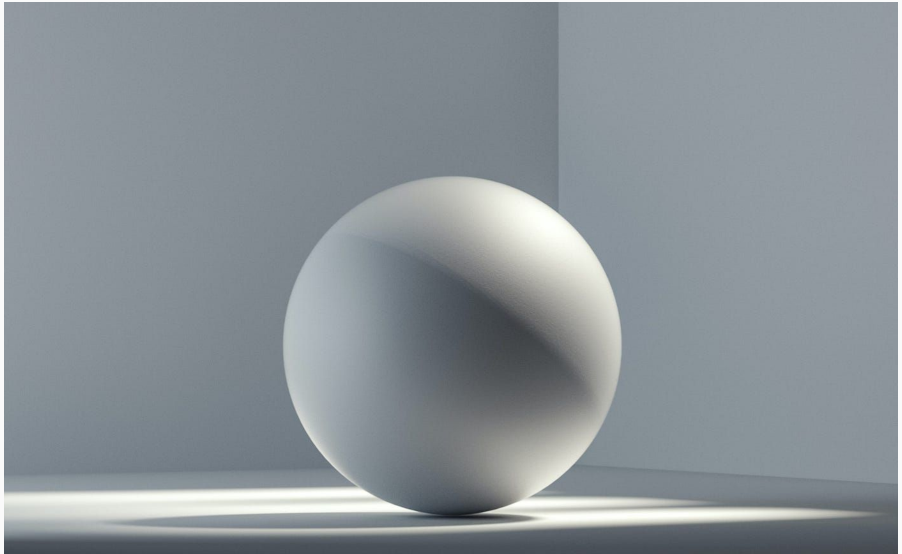
	Preprocessing	Run	Post-processing	Vertex Population	Swap Buffers	Draw + Misc.
CPU time (ms)	2.401 **14.3%**	3.81 **22.69%**	0.0004 **0.0023%**	0.0014 **.0083%**	0.41 **2.44%**	10.17 **60.56%**
GPU time (ms)	1.96 **39.74%**	1.13 **22.91%**	0.00015 **.003%**	0.00052 **.01%**	0.031 **.63%**	1.81 **36.7%**

Results: Frame Breakdown



Future Work

- Polish and submit PR to BELLHOP Acoustic Toolbox
- Use the 3D BELLHOP model and create corresponding OpenGL frontend
- Leave vertex data in GPU memory when drawing


3D Generative Modeling with DeepSDF - C. Wolfe

Conclusion

- Significant speed ups in both computational and graphical components
- Great starting point for a real time underwater communications simulator

Sources

- http://oalib.hlsresearch.com/Rays/HLS-2010-1.pdf
- https://arlpy.readthedocs.io/en/latest/_static/bellhop.html
- https://github.com/A-New-Bellhope/bellhopcuda
- https://towardsdatascience.com/3d-generative-modeling-with-deepsdf-2cd06f1ec9b3