# CS 217 Final Project
# Progress Report

Daniel Vyenielo
dvyen001@ucr.edu
https://github.com/DVNLO

November 18, 2021

**Abstract**

For my final project I will implement solutions to the point in polygon problem on CPU and GPU and compare the performance of each implementation. A solution to the point-in-polygon problem [1] determines if a point in a two-dimensional plane falls within the boundary of a two-dimensional polygon in the same plane.

## 1 Project Description and Sequential Timeline

The project will be implemented sequentially in 4 phrases.

In the first phase I will implement a sequential classical ray casting algorithm on CPU.

In the second phase I will parallelize the sequential implementation using the c plus plus algorithms library [2] and specify a parallel execution policy [3]. This phase should be fairly trivial since the classical ray casting algorithm exhibits the necessary independence to be embarrassingly parallel [4].

In the third phase I will implement the classical ray casting algorithm on GPU. This phase may expand into successive implementations with the intent to further optimize the implementation, successively, through different techniques we learned this quarter. However, I am only willing to commit to one implementation, but, time allowing, I may try additional optimization techniques.

In the last phase, phase 4, I will benchmark the non-parallelized sequential implementation on CPU, the parallelized implementation on CPU, and the implementation(s) on GPU.

[1] https://en.wikipedia.org/wiki/Point_in_polygon
[2] https://en.cppreference.com/w/cpp/algorithm
[3] https://en.cppreference.com/w/cpp/algorithm/execution_policy_tag_t
[4] https://en.wikipedia.org/wiki/Embarrassingly_parallel

## 2 Outcome

The products of this project will be a sequential classical ray casting implementation, a parallel classical ray casting implementation, and benchmarks showing the relative and absolute performance of the various implementations running on randomly generated sets of points and polygons. I will compile the results of my benchmarks and present them in the final report.

## 3 Tools/Libraries

**I need access to a c plus plus compiler on Bender with at least cpp-17** to use the parallel execution policy [5], specifically;

```
class parallel_unsequenced_policy { ... }; (3) (since C++17)
```

. Otherwise, I can use the existing toolchain on Bender for the project.

## 4 Team

I will work individually on the project.

[5] `https://en.cppreference.com/w/cpp/algorithm/execution_policy_tag_t`