Automated Solving of Jigsaw Puzzles – A Lake Monsters Social Distancing Project

Like me, many of you or your family members have spent time during quarantine solving jigsaw puzzles. This got me to thinking: can a computer solve jigsaw puzzles? So, I did some research and found several papers and research projects by folks looking at the problem. One pair of papers by two academics used a rigorous mathematical approach that looked interesting. So, I spent some time noodling around with the problem and have concluded that this might make an interesting group project for team members looking for some intellectual stimulation while the lab is closed.

I propose that interested students work with me (and any other mentors who would like to join in) to read and understand the two papers and, based on their concepts, we implement a jigsaw solver.

One of the reasons I liked the approach of these two researchers is that I believe the math involved, while college-level, is none-the-less within reach for any of our students who are willing to put in a little effort under the guidance of one or more mentors.

# Reading the Papers

We’ll cover a number of math topics. In all cases, mastery of the topics is NOT a requirement. Instead, think of this project as introducing you to these topics and their underlying concepts. For example, there is a little calculus involved. You might think, “I can’t do that. I haven’t even taken pre-calc yet.” But, if you understand, for example, that speed is distance divided by time, you already know almost enough calculus to understand these papers!

Some of the math topics we’d cover:

* Calculus – derivatives. Not how to compute them. Just what they are.
* Linear algebra. Used in these papers as a way of compactly representing multiple equations as a single equation.
* Computational Geometry. A fancy term for using the computer to solve geometry problems such as: do these two pieces fit together?
* Math notation. The papers express relatively simple concepts using what will be, to most of you, unfamiliar notation. For example, the list of points that represent the boundary of a puzzle piece is called a curve and is represented by . At one point, the paper says: “we will consider simple plane curves A plane curve, as should be obvious, is a curve lying in the x/y plane. The math expression is a formal way of saying **exactly** the same thing. Let’s parse it out:

That’s our curve.

Means “is contained in”

The set of real numbers. In other words, the number line.

The x/y plane. Think of the x-axis as one number line and the y-axis as another.

So, the expression literally means “ lies in the x/y plane” and thus is repeating exactly what the words said.

# Programming Concepts and Tools

## C++

Based on what I’ve looked at, I would recommend we implement this in C++. The main reason is the availability of several useful open-source libraries. All these libraries are available for C++, but not universally for Java or Python. If you already know Java, you’ll be able to pick up the necessary C++ as we go along. We’ll using Microsoft Visual Studio Community Edition, which is free.

## Eigen

Eigen is a library for linear algebra. It greatly simplifies the programming of repetitive calculations such as performing the same operation on all points on a curve.

## OpenCV

A real-time computer vision library.

## Intel TBB

Intel’s Threaded Building Blocks. A library for parallel programming on multi-core systems.

# Logistics

Here’s how I think this would work. There are several phases:

1. Reading the papers. We’d meet as a group and read the papers together, with me available to answer questions, explain novel concepts encountered, etc.
2. Planning the solver. As a group, we’ll
   1. Figure out what data structures will be needed to represent the puzzle and solution.
   2. Divide the solution into subsystems. The papers describe several phases to the solution.
   3. Create a test plan. It is important to be able to test each of the subsystems independently. This allows teams to each work on their subsystems without having to wait for an earlier subsystem and also allows teams to use consistent data for testing.
   4. Assign implementors. We’ll divide into subteams for the subsystem implementation.
3. Writing the solver.

# References

1. [Hoff, D., Olver, P.J., Automatic Solution of Jigsaw Puzzles, preprint, University of Minnesota, 2012](http://www-users.math.umn.edu/~olver/vi_/puzzles.pdf).

2. [Hoff, D., Olver, P.J.: Extensions of Invariant Signatures for Object Recognition. J. Math. Imaging Vis., Online First(TM), 7 June 2012](http://math.umn.edu/~olver/vi_/hoffo.pdf).