# Overview

I believe the challenge is complete. The executable is LaserEstim.exe. It is built from 2 projects:

* Geometry : builds a library and includes the bulk of the programming
* LaserEstim: builds the executable. This is just a dos command line program for calling into the library.

The user-experience is via command line. Any argument error should show a help string. Other error handling is minimal per the following notes:

* Many Json errors just throw “unknown exception”
* Open loops of edges are unmanaged. A trivial example of an open loop would be a single line segment.

Two json files were added for testing:

* IceCreamCone.json tests non-orthogonal profiles
* Google.json tests a simple circle, presumed to be the only case having just one vertex.

## Disclaimer

I’m going to discuss this project like it’s real. The one exception is that I’m not going to discuss nested shapes, because that would invalidate most of the scope.

# Unresolved

## Open Loops

The problem with open loops is serious. Imagine a profile consisting of just two crossed line segments. Such a part would be quoted with no errors reported. The application cannot be considered ready for release until the problem with open loops is resolved. Two measures are recommended:

* Present the profile graphically for a user to review.
* Add automation to check for cases like open loops and intersecting edges. The result of the check would be: Ok, No-Way, or Review

One might argue that closed loops are expected, but I don’t think one can argue that closed loops are guaranteed.

## Interface

The command-line interface will probably not be used for production. It may have some diagnostic use. That leaves open the question of how this solution will interface with business processes. The likely scenario is as follows:

* The existing automated quoting system is extended to link with the geometry library provided here to automatically process profiles (the Json files) and respond when all is ok.
* Non-ok profiles are sent to a person/user for review. A software application would provide a user interface for viewing the profile and either declining the RFQ (request for quote) or responding with a quote. Once the user has decided, the application would communicate back to the automated quoting system.

# Challenges

## Json

When I saw the Json file I recognized the format. I have written manual parsers for that format in the past but they were not identified to me a json files. A home-written parser for a published format is rarely a good idea so I downloaded json-cpp from: <https://sourceforge.net/projects/jsoncpp/>. There were problems linking with an MFC build and I chose MFC to assure a future option to view the profile. I solved the problem by adding the json-cpp files to the geometry project (5 cpp files and 12 header files) . The only change to those files were the #include lines.

The vertex identifiers were coming in as strings and the edges were referencing those identifiers as integers. That is handled in the application with a try/catch where the catch attempts to convert/parse the string as an integer before rethrowing the exception.

Schema.json was only used as a reference. I could not find a way to have it help json-cpp.

## Git

I expect I will be able to submit the finished project with a Git-push but I was far from successful with Git. The first attempt had the following problems:

1. A large number of files that don’t belong in the repository. – my error  
   I haven’t been able to learn how to clean-up such problems.
2. Checkin comments are present, but not very good. I haven’t been able to learn how to go back and review those comments.
3. The json files should have first been checked-in as exactly downloaded before the modified versions were checked in. I wasn’t able to do that.
4. I haven’t learned yet how to assure that a repository is sufficiently complete. Typically, I would get all the files to an empty directory and then assure a proper build. With Git, the process will be similar but Git must be present in the directory before it can be run, so a ‘catch-22’.

The clean Git file will be pushed as Bobk\_Plethora\_Challange. The poor first attempt will be pushed as Bobk\_Plethora\_Challange\_NG.

# Future enhancements

## Json errors

Errors from the parsing need to be much more descriptive and should report line numbers from the json file. Granted, the json format considers line feeds to be whitespace, but line numbers remain the best way to find any offending data. This work will require changes to the json source code.

## Test harness

Before release there should be a test harness that automatically runs the application through a series of files and forces various known error conditions.

## Machine time estimates

Generally, the reason machine tool paths slow down for a tight radius is the acceleration limits on the machine servos. If that is the case, perhaps there are acceleration issues for corners between straight lines. Remember: I said I would write about this as if it were real.

Rapid traverse time to bring the laser into position and move between edges is not trivial and should be considered. An example of a move between edges would be a flange that has an OD, and an ID.

## Recyclable material

Material costing 0.75 $/in for thicknesses that can be laser cut is fairly expensive like copper, so the quoting application should consider the recycle value of the cut-away material.

## Scalability testing

The application should scale quite well. The ‘test’ argument in the command line application attempts a quick-n-dirty scalability test, but a proper test will require some much larger profile files.

## Gotchas

Perhaps add handling for “gotchas”. My laser cut experience is many years out of date, but one example of a gotcha might be that very small pieces fall out and are tough to gather. Hmm, maybe I’m recalling a plasma cutter.