FINITE ELEMENT ANALYSIS

ARDEN RASMUSSEN

Programming

Mesh Generation. Implement constrained Delaunay mesh generation given a specified PSLG. I have already implemented Delaunay mesh generation given a PSLG, but just need to implement the constraints enforcement. This *should* be doable.

Implement mesh refinement algorithm. Specifically Chew's Second algorithm. This will allow for finer control in an implementation of the area function, and allows for more analysis of how the mesh refinement will effect the accuracy of the resulting finite element approximation.

If I'm feeling bored, maybe create a PSLG creation system? Nope! I just decided that it is easy enough. I might make a Python system for assembling basic geometric shapes, and specifying their "resolution". So a user would be able to specify a point/rectangle/triangle/ellipse at position of size, and weather it is a hole or not. Then through a combination of these basic shapes, I think that most necessary meshes should be easily constructed.

Core. I've already implemented numerical integration and differentiation. So those should be good. I'm using a higher order numerical differentiation which gives errors of some small order. It should be sufficient for most reasons. And integration utilizing a very basic rectangular summation algorithm. I want to implement integration on triangular domain utilizing Gaussian Quadrature, on the barycentric coordinate system. There is a spread sheet online that provides all the weights and positions that I need.

Fully understand how to construct the elements of the matrix, though element wise composition. This is the key step, as I think that I have most of the other basic parts figured out.

I need to implement a system to construct the global basis functions. My best idea is to pass a vector of local basis functions, then by some if statement magic return the value of the appropriate local basis function.

Implement systems to enforce boundary conditions. Start with just Dirichlet, then try to implement more interesting conditions.

Implement time dependence! Nothing interesting happens without time dependence. So I really need to implement a method for using time dependence. The nice thing is that I can use the same mesh for each time step.

Or if I want to be really cool, I can try to refine the mesh based on the variation between the values at near by points, i.e. for larger variation between points, we would increase the mesh

Date: January 29, 2019.

refinement, and for very little variation, we could reduce the mesh. This would attempt to improve the mesh where it would be most necessary, and reduce it where there is not much interest happening.

Mathematics. Sparse matrix class, to implement large sparse matrix storage. Use the `row_ptr_` and `col_ind_` system. I think that this is probably the best method for storing the sparse matrix. Note that the best method does change for the sparsity of the matrix. So on initial tests, be sure to also print the sparsity of the matrix, so I can actually determine if this is the best method for storing the matrix. I have already done this, so I just need to addapt it.

Sparse matrix solving. I have implemented this using the Jacobi, GaussSeidel, ConjugateGradient, and Cholesky methods. I think that the Conjugategradient method is the best method for a numerical approximation of the system, but the Cholesky method provides the actual solution, but is a good method for getting the actual solution. When I get a matrix, be sure to print the sizes of the matrices, and the order of them. I'm thinking $\approx 10^4$ at most? Again, print this in order to get the actual order, then I can see if using the Cholesky method would prove to be too slow ($\approx 10s$).

My thinking is that if I implement time dependence, then I would probably want to do some iterations for $10^2 - 10^3$ steps, so if each step takes $10s$ then in total it would be more than $10^3 - 10^4 s \approx 16.6m - 2.7h$. Thus on the upper end, this would be annoying but possible. But if I don't implement time dependence, then the time of calculation doesn't matter at all, so I should just use the completely accurate method as long as it takes less than $10m$. Right?

Analysis.

Numerical Results. Ask Paul about the best method for result storage:

- **Function**: I would have to modify all of my functions to store the `lambda` expression as a string as well.
  - *C/C++*: I would need to construct a parser of the function string representation and convert it into valid C/C++ code. This would however be very nice, as it would allow for easy implementation into other system, and actually provides us with a usable function that gives us the result!
  - *Psudocode*: This is less useful, but could be made easier for my own parser to read and convert it into usable code. So I could create a python parser for my Psudocode and convert it into C or python code.
- **Values**: This limits the result to just the resolution that is provided from the initial execution, but I could make a very simple system to save the values for each "pixel" for all the different time steps. This would be much easier, but would not be as extendable. The "resolution" would be fixed from the run time, and if zoomed in too much would cause aliasing issues.

Visual Results. Reimplement a method for plotting the values of the output graph. I have done a version of this, but it is pretty bad, and I think that it is broken, so I need to rewrite it from scratch. Implement both 3D plotting and 2D color map based plotting. I would need to test every point in the domain mesh. I think this is best done by finding the bounds

of every triangle, then iteration over the points that are within the rectangular bounds of that triangle. Only calculate the value if that pixel is within that specific triangle. Then repeat this for all triangles, every time that we want to calculate for a pixel that we have already determined then we just skip it. But this should never happen, because we are only calculating for pixels within the current triangle, and so it should have never been calculated before. This could be paralleled without much difficulty.

2D is easy, 3D is simply using some $3D \to 2D$ projection, I do need to care about z-buffers. Then it would be nice to be able to set the camera position, and rotation. For 2D it would also be pretty nice to overlay the mesh. To overlay the mesh I could just use a simple line rasterizer, I am very good at those by now. In order to add text labels, I would need to import FreeType2, and we all know that's a pain in the ass. But It would allow for the generation of some very, very nice plots.

A different option for the graphical representation of the solutions, would be to generate a TIKZ picture of the image. Either as 2D with a bunch of colored squares, then overlaying the mesh is easy. Or as 3D with a bunch of connected lines/rectangles. Then all of the actual plotting would be done by latex.

I don't really care about the animation, so I will leave all of that to FFMPEG, but if I do leave it to FFMPEG, then I really need to figure out how to control the output frame rate, to make it appropriate.

Unlikely. If higher resolution on 2D is really desired, I could re-implement bi-cubic interpolation for inserting more values between the points. Or if that proves too much, then just to a bi-linear interpolation between points? It would not provide as accurate detail, but would be sufficient to prevent the hard edges.

Mathematical Analysis

Finite Element Factors. This is a list of things that I would find interesting to compare their effect on the accuracy of the finite element computations.

- Mesh Minimum Angle
- Mesh Minimum Area
- Mesh Edge Refinement
- Mesh "Visual appeal" from Chew's second algorithm.

Other. This is a list of interesting things that I could use Finite Element Analysis to look into further.

- Basic fluid flow simulation