Implementation of the Easy Integral Surfaces Algorithm in C#

Augustus Loftin and Jonah Bosland

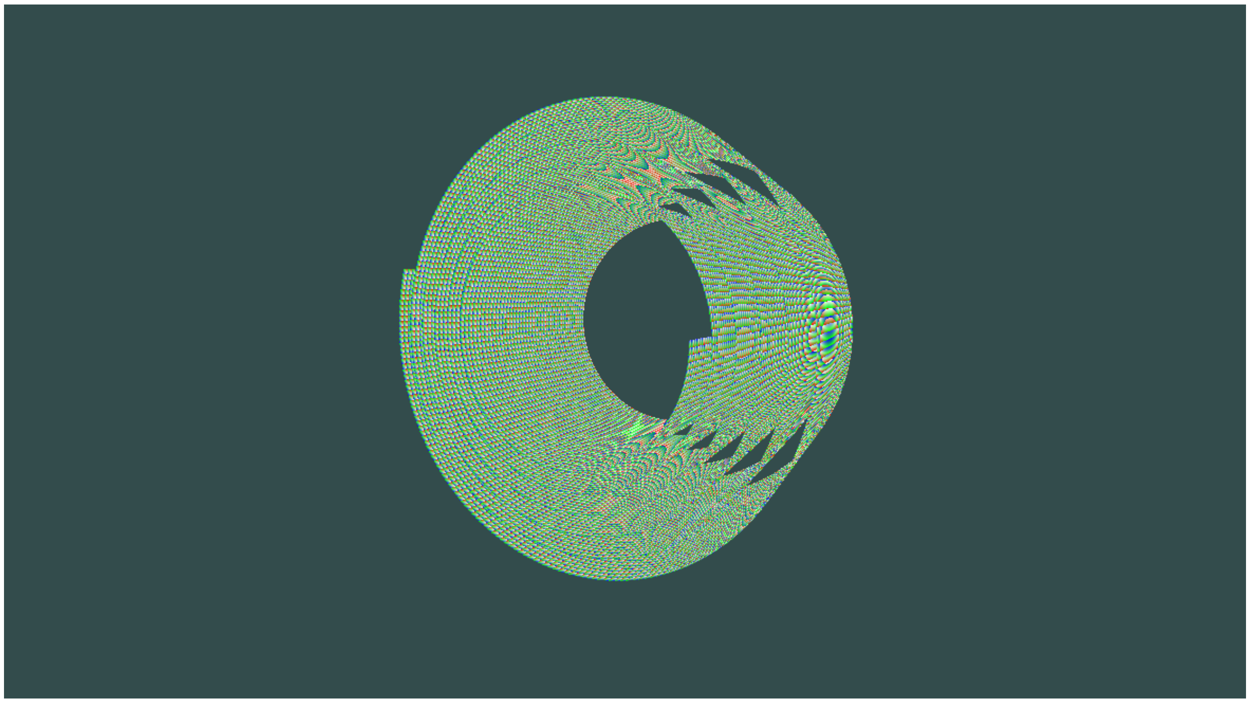

Fig. 1. An Integral surface representation of the vector field: $x = y, y = -x, z = 0$. With initial seeds of (1, 0, -1) (1, 0, 1), step size of 0.1.

Abstract—This paper is focused on our implementation of the easy integral surfaces algorithm as defined by Tony Mcloughlin, Robert S. Laramee, and Eugene Zhang in their paper *Easy Integral Surfaces: A fast, Quad-based Stream and Path Surface Algorithm.* These authors sought to layout an algorithm that would generate a quad mesh for stream path surfaces from a 3D vector field data set that was simple in its implementation. We then attempted to take this layout and implement the algorithm using C over C++ while still using OpenGL. This resulted in a fully functional algorithm that visualizes integral surfaces in a fast and efficient manner.

Index Terms—Integral Surfaces, C#, OpenGL

1 Introduction

The goal of our project was to implement the algorithm defined in the paper *Easy Integral Surfaces: A fast, Quad-based Stream and Path Surface Algorithm.* by Tony Mcloughlin, Robert S. Laramee, and Eugene Zhang [1]. The purpose the author's had for writing this paper was that integral surfaces were mostly avoided in the visualization community due to the inherent complexity in the algorithms used to create them. This problem was solved by a combination of multiple techniques, the most notable being the use of a simple 2D array as their data structure, which allowed for the easy implementation of a quad mesh. This paper shows a new implementation that is written in C# using the Open Toolkit library for access to OpenGL.

2 Background

In order to understand the workings of integral surfaces you first need the definition of streamlines. A streamline is a line that is tangent to the velocity of a vector field at each point. What this shows is the direction that an object would travel at that point at any time. We visualized 2D versions of these for project 3 and also use them to form quads in the algorithm implemented in this paper. An integral surface is the surface representation (or integration) of streamlines. It is very similar to if you just infinitely stacked streamlines together to the point where they appeared to be continuous.

• *Augustus Loftin is with Oregon State University. E-mail: loftina@oregonstate.edu.*
• *Jonah Bosland is with Oregon State University E-mail: boslandj@oregonstate.edu.*

3 Previous Work

The previous work that went into beginning this project were a paper that the project is based off of and Project 3 where we used similar techniques for visualizing streamlines on 2D vector fields.

3.1 Original Paper

As mentioned previously our algorithm is based on the framework laid out by Tony Mcloughlin, Robert S. Laramee, and Eugene Zhang in their paper *Easy Integral Surfaces: A fast, Quad-based Stream and Path Surface Algorithm [1].* This paper was published in 2009 and utilized openGL 1.2. The paper lays out 5 major steps that it takes in creating the mesh: seeding, advancing, updating, terminating, and rendering.

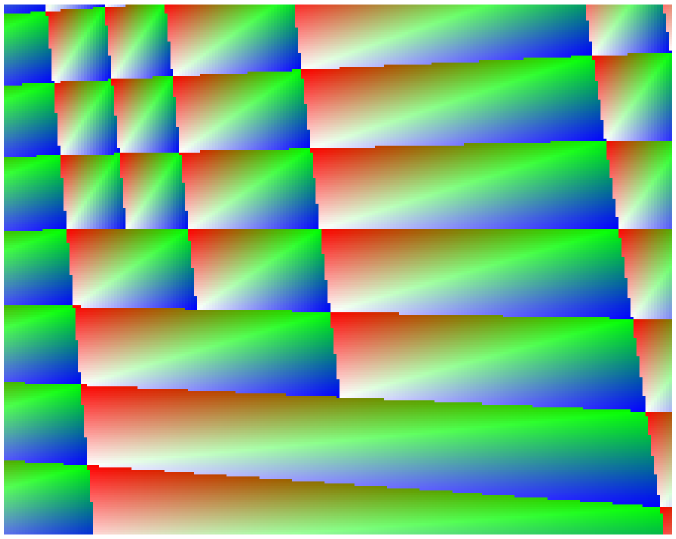
Fig. 2. An example of divergence.

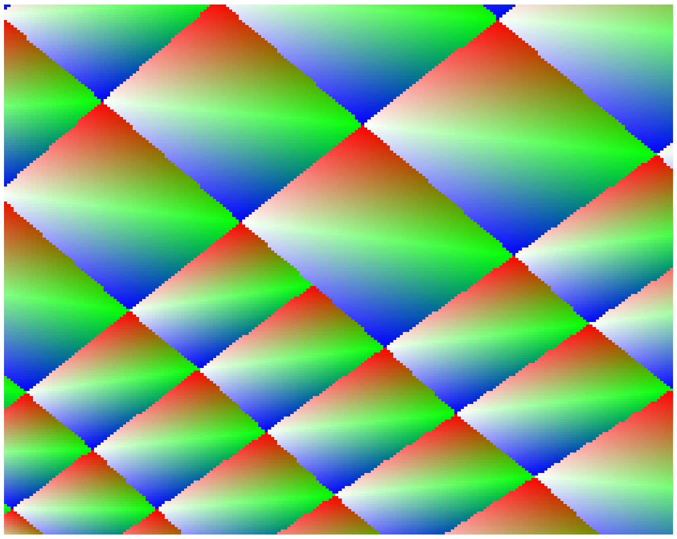
Fig. 3. An example of convergence.

3.1.1 Seeding

For the seeding step of the algorithm first we input two points and retrieve the distance between them creating the value for d_sample. We then insert a streamline seed per column that are separated by a distance that is at most one half of d_sample. With this we have set up our data structure with one vertex per column and are ready to begin to advance the surface.

3.1.2 Advancing

In order to advance the surface we take each seed and advance one step forwards in 'time' for each column. It is important for this part of the algorithm to preserve the empty flag of the vertices in order to keep consistent geological information. Specifically if advancing in a column where the last vertex was empty then your newly inserted vertex must also be flagged as empty. After you have inserted these new vertices into your data structure you have now created a new row of quads in your vertex and you must scan them for updates.

3.1.3 Updating

The reason that you must scan your new row for updates is to preserve visual quality by preventing effects such as shearing. To do this you must check for flow divergence, flow convergence, and flow curvature.

- When checking for flow divergence you must have a quad where the bottom two angles are greater than 90 degrees with a separation distance of greater than one half of d_sample. To correct this we insert a new column of empty vertices into our data structure and populate two new vertices that are used to split the current quad in two. An example of this can be seen in figure 2.

- When checking for flow convergence you must have two quads where the bottom most left and bottom most right two angles are less than 90 degrees with a separation distance of less than one half of d_sample. To correct this you set the two vertices that are separating the two quads as empty and no longer visualize them conjoining the two quads together. An example of this can be seen in figure 3.

- Flow curvature occurs when one angle at the bottom of the quad is greater than 90 degrees and the other is less than 90 degrees. In the paper they discuss how this requirement might be too strict and that these values produce better results at greater than 93 degrees and less than 87 degrees. When updating quads that meet the requirements for flow curvature you process them in an order based on whether they are experiencing right or left flow. For right flow you process the row from left to right and vice versa. The action that you commit on the quad is that you rotate it by shortening the side running along the column of the data structure that is adjacent to the larger angle. When you shorten the side you lower the upper vertex setting the new distance between them to be the length of the opposing edge of the quad times the smaller angle divided by 90. An example of this can be seen in figure 4.

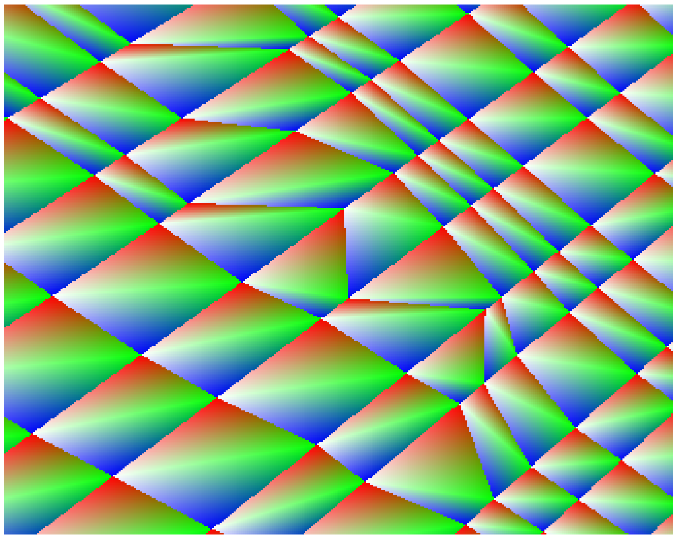
Fig. 4. An example of curvature.

After you have properly updated all of your quads for the most recently advanced row you must check your terminating conditions.

3.1.4 terminating

There are four termination cases for an integral stream surface.

- The advancement function for the program has a limited number of times that it can be called. If the surface exceeds this limit then the advancement will stop. This is to insure the surface does not get stuck in an infinite loop.

- The advancement will also stop is part of the surface leaves the vector field. This is to insure that the surface does not go outside the memory bounds.

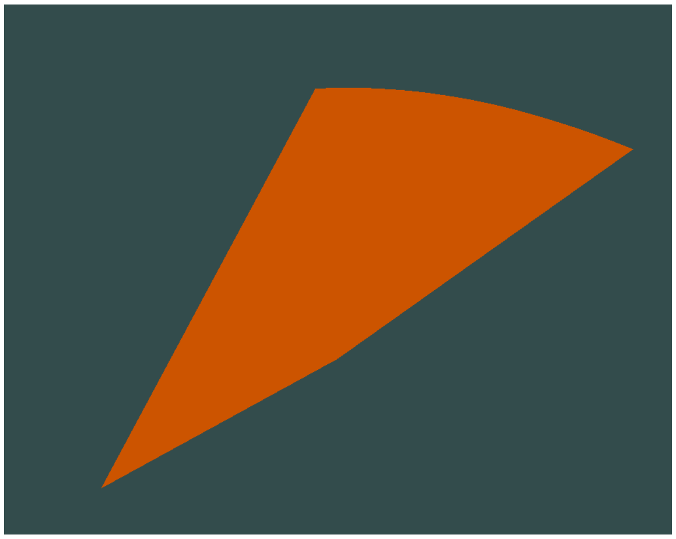
Fig. 5. An integral surface representation of the vector field: $x = x, y = y, z = z$. It progresses 500 steps.

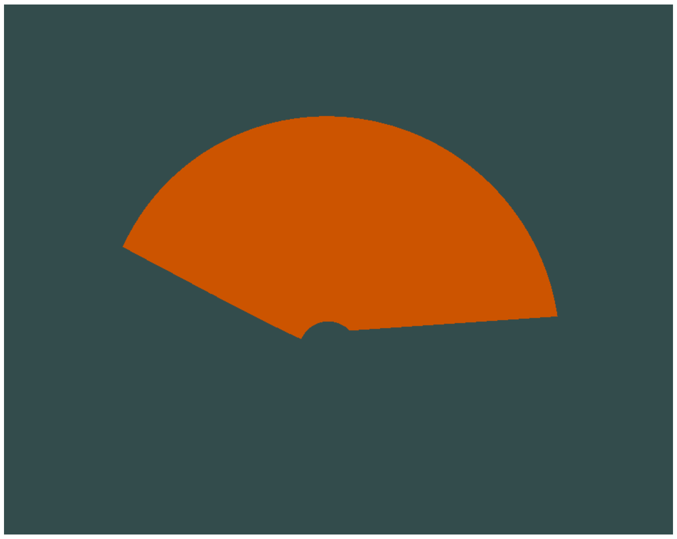
Fig. 6. An integral surface representation of the vector field: $x = \frac{-y}{x^2+y^2}, y = \frac{x}{x^2+y^2}, z = 0$. It progresses 500 steps.

- The advancement can additionally be stopped by a separation point. This will stop the surface construction and introduce two new surfaces that can be computed independently.
- Finally the advancement will be halted if the surface encounters a point of zero velocity. This is to avoid drawing unnecessary data in a sink.

3.1.5 Rendering

The integral surfaces are rendered using OpenGL using a quad based mesh. The quad based mesh helps make the integral surfaces much easier to understand and to properly texture.

3.2 Project 3

The other piece of previous work that went into our algorithm was the streamline step function from project 3 of this term. Since we did not have access to the integrator that the original authors used we had to create our own function for producing a single step forwards in a streamline. The streamline step function with some slight changes worked perfectly for this.

4 Results

Since we were unable to produce naturally occurring or simulated flow field data in order to test our algorithm we used vector field equations. The reason that we could not produce flow data is that we planned to use the John Hopkins turbulence flow data set, but when we went to generate the data we learned that we would have to get an access key that would take 3-4 days to obtain. At this point that would not work with our timeline and pivoted to using equation representations of vector fields. Some examples of the integral surfaces we generated are provided in figures 5 and 6. Another area in which we had to deviate from the original algorithm was that we used a2D list instead of an array. the reasoning for this is that we decided to code our algorithm in C# which does not support dynamic 2D arrays the same way that C++ does.

5 Evaluation

To evaluate our results we used two methods. The first method was using a 3D vector field visualizer to make sure that our surface was properly visualizing the slice of vector field. The second method was coloring each vertex of a quad a different color. This method helped us orient ourselves to the beginning and end of our surface as well as allow us to evaluate whether we were properly updating the quads. This color scheme can be seen in figures 2, 3, and 4.

6 Division of Tasks

For this project we eventually decided that pair programming would be the best method to ensure a high quality of work. Despite this there are still some functionalities that we both worked on separately. We also both contributed to the writing of this paper.

6.1 Augustus

The tasks that Augustus completed in this project were:

- The Divergence algorithm.
- The Curvature algorithm.
- OpenTK initialization.

6.2 Jonah

The tasks that Jonah completed in this project were:

- Adapting the streamline step function.
- The Convergence algorithm.
- Producing screenshots for the paper.

7 Conclusion

Overall we managed to meet our goal that was proposed when we presented this project to the class. Our algorithm is fully functional and succeeds in visualizing vector fields in an efficient and timely manner. We also decided to pivot and code our algorithm in C# differing from the original. As a result we generated multiple integral surfaces in openGL and have a created a GitHub of this project for potential further use if necessary.

8 Acknowledgements

We used OpenTK which is a c# wrapper for OpenGL. Much of the code that just gets OpenTK working was copied from the tutorial for OpenTK [2, 3].

References

[1] T. McLoughlin, R. S. Laramee, and E. Zhang. Easy integral surfaces: A fast, quad-based stream and path surface algorithm. In *Proceedings of the 2009 Computer Graphics International Conference*, CGI '09, p. 73–82. Association for Computing Machinery, New York, NY, USA, 2009. doi: 10.1145/1629739.1629748

[2] D. Perks. Learn opentk.

[3] Varon, J. Häger, P.-J. Briers, and F. JA. Opentk github. Dec 2021.