Morse-Smale Visualizations

Shayla Rao and Josie O'Harrow

2021
December

1 Abstract

The purpose of this paper is to outline our Scientific Visualization final project on Mathematical Topographical Mapping. In this paper we will be discussing the Morse-Smale complex algorithm and how it applies to topographical data and critical points. In this project we created a data set, applied algorithms to it and finally outputted a visualization. The final visualization shows, height, critical points such as maximums, minimums and saddle points, their classifications and Morse-Smale lines.

2 Introduction

The domain expert is Josie, who studies mathematics. That said, our data set is not strictly mathematical; we will be visualizing topographical maps of mountains and applying mathematical properties to them. We created this data set in a way that it mimics mountains and contains the maximums, minimums and saddle points necessary to make our calculations. Some of the features of this project include, the creation of a data set in the form of a ply file, applying algorithms to the data set and outputting a final visualization. Our visualization contains two modes, the first being a flat visualization of the data set that contains all of the critical points which include: maximums, minimums and saddle points and their classifications using colors. The second mode contains the a visualization of the data set with height included. This mode shows the mountains and classified critical points through the use of colors. The maximum critical points are colored green, the minimum critical points are colored red and the saddle points are colored blue.

3 Background

The Morse-Smale complex is a tool for smooth 2−manifolds, which we are applying for the purposes of this class to regular surfaces which are the graph of 2D scalar fields in Euclidean 3−space. The Morse-Smale complex partitions the domain of a scalar field into regions where the gradient of the scalar field has similar behavior. This would be particularly applicable for a topography visualization, which was why we were originally interested in learning this tool.

3.1 Definitions

Let M be a smooth 2−manifold, and let $F : M \to \mathbb{R}$ be a function.

Definition 3.1. If $x \in M$ and $df(x) = 0$, then x is a singular point.

Definition 3.2. Let M be a smooth 2−manifold. An integral line for a smooth function $F : M \to \mathbb{R}$ is a flow along the gradient of F that connects two singular points, and goes through no other singular points of F.

Definition 3.3. Let x be a singular point of M. The ascending manifold of x, A_x is the set of points belonging to integral lines whose origin is x, and the descending manifold of x, D_x is the set of points belonging to integral lines whose destination is x. Note that these may be empty.

Definition 3.4. The Morse-Smale complex can be defined to be:

$$\bigcup_{x,p \in M \text{ are singular } x \neq p} A_x \cap D_p$$

Applying this definition to the data set that we wrote, the Morse-Smale complex would look like this:

.

If we focus our attention to the minimum point, in red (p_1), and the top right maximum point in green (p_2), then $A_{p_2} = D_{p_1} = \emptyset$, and $D_{p_2} \cap A_{p_1}$ can be seen in the figure below highlighted in orange:

Here, the orange region along with the edges and vertices bounding it would make up a 2−cell in our Morse-Smale complex.

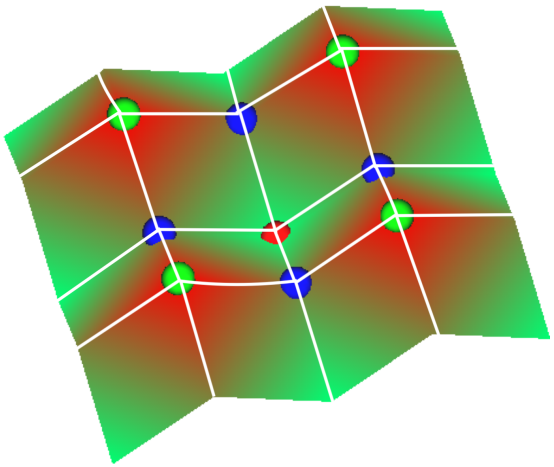


Figure 1: Morse-Smale complex

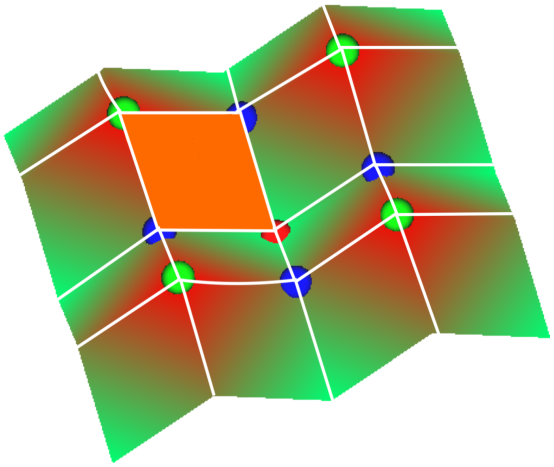
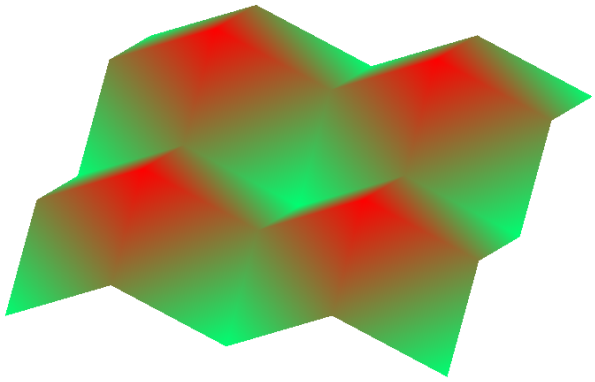


Figure 2: 2-Cell in Morse-Smale complex

4 Data Set



The image above shows the data set used in the project. We created this data set using the format of a ply file and applied it to the data we created. We wrote the actual data by hand so that we could test our features using a toy model for the type of surface that they would actually show up on. We changed the vertices and the faces which also changed the number of vertices and faces. This is the visualization of the data set without any of the algorithms for critical points and Morse-Smale applied to it.

5 Division of Tasks

The tasks of this project are such that they need to be done somewhat simultaneously in order to work well. So, the way we completed this project was by pair programming using Live Share tool in Visual Studio 2019. While programming, we both worked on the functions together and when issues arose we debugged together as well using the Local Windows debugger and using breakpoints when necessary. In order to understand the background ideas of this project such as the Morse-Smale complex and other algorithms, Jo would understand them and explain them to me in a way that I would understand them.

6 Algorithms

Below we list the type, name, and inputs for all the functions we added, including a description for each algorithm.

1. icVector2 normalized_grad_at_point(Quad, x, y)

   Given a quad and a point x, y lying inside the bounds of the quad, this function returns the normalized gradient at that point based off of the bilinear interpolate function presented in class. Using our bilinear interpolate function, $F : A \subset \mathbb{R}^2 \to \mathbb{R}$ we defined the gradient to be $\nabla F(x,y) = \left(\frac{\partial F}{\partial x}(x), \frac{\partial F}{\partial y}(y)\right)$, where we computed these partial derivatives by hand using the bilinear interpolate formula given in class. Then for a point (x,y), we returned $\frac{\nabla F(x,y)}{||\nabla F(x,y)||}$ using the normalize function for the icVector2 class.

2. float bilinear_interpolate_at_point(Quad* quad, float x, float y)

   This algorithm was re-used from our bilinear interpolation in class. Given a point (x,y) within a quad with x_1, x_2 bounding x values in the quad, and y_1, y_2 bounding y values within the quad, we return:

$$\begin{aligned} f(x,y) = &\frac{x_2 - x}{x_2 - x_1}\frac{y_2 - y}{y_2 - y_1} f(x_1, y_1) \\ &+ \frac{x - x_1}{x_2 - x_1}\frac{y_2 - y}{y_2 - y_1} f(x_2, y_1) \\ &+ \frac{x_2 - x}{x_2 - x_1}\frac{y - y_1}{y_2 - y_1} f(x_1, y_2) \\ &+ \frac{x - x_1}{x_2 - x_1}\frac{y - y_1}{y_2 - y_1} f(x_2, y_2). \end{aligned}$$

3. Quad* get_quad_by_position(float x, float y)

   Given a (x,y) position that falls in our data set, we loop through all the quads and return the one that contains our point. If the point happens to fall on a vertex, this will return the first quad found.

4. Vertex* find_with_coords(double x, double y, int vert_count, Vertex** vlist)

   Given coordinates (x,y) and a list of vertices, this returns either the vertex at point (x,y) in that list, or NULL.

5. void find_critical_points() This method found all of the singularities occurring within quads or at vertices, and added them to the appropriate singularity lists.

We used two modes to search for and classify singularities. The first mode used our search for singularities method from project 2. We classified these singularities by checking the determinant of the Hessian matrix, but in this case that could only definitely tell us that a point was a saddle point. Thus, to figure out if a point was a min, max, or saddle, we sampled many points in a circle around our point and checked if they were all of larger scalar value, smaller scalar value, or mixed larger and smaller scalar value, than the scalar value of our point of interest. In the first case, it was a minimum, in the second a max, and in the third a saddle point.

The second mode looped through all vertices and manually verified which ones were minima, maxima, and saddle points by comparing them to their neighbors. We called a point a minima if it has a lower scalar value than all of its neighbors. We defined it to be a maxima if it had a higher scalar value than all of its neighbors. Finally, if two of its neighbors were higher and two were lower scalar values and the lower scalar valued neighbors were not themselves adjacent, we defined the vertex to be a saddle point.

6. double det_hessian_matrix(Quad* quad)

   This returned, given a quad, the determinant of the Hessian matrix for the bilinear interpolate function for that quad using second derivatives that we computed by hand.

7. void init_verts() This function, run at the very beginning, normalized the vertices to have scalar values in a [0, 1] range, and updated their z-values according to scaling by the maximum bounds value we computed using the minimum radius needed to bound the mesh completely in a square based at the origin, and a custom set factor for additional scaling as needed.

8. void morse_smale()

   The idea for this algorithm is to approximate the actual boundary lines for the 2-cells in the Morse-Smale complex. This is based on the fact that a cell in a Morse-Smale complex will be bounded by the lines joining the minima to the two saddle points, and the maxima to the two saddle points. Then if we find, or closely approximate, all such lines we will have a close approximation for all the boundaries of 2-cells in our Morse-Smale complex.

   We assume we do not know an actual generating function for the data set, and instead draw (approximate) integral lines from each minimum and maxima in many directions using points on a circle based at the minimum/maximum

point. Once we have computed many of the integral lines from our singularity of interest, we look for any that have a point on them sufficiently close to a saddle point. Here, sufficiently close is defined using a constant in our code. If any are close, we choose the integral line that has a point falling the closest to that saddle point and add it to the set of lines drawn in our morse-smale visualization at the end.

The halting criteria for drawing our integral lines works in the following way: we continue to go in the direction of an integral line until we are no longer increasing or decreasing, depending on if we are coming from a minima or maxima.

To draw the integral lines, given a starting point near our singularity, we compute the gradient at that point and take a step of size STEP_SIZE in the direction of the gradient at that point. This step is added as a line segment in a PolyLine object that is approximating the integral line containing our starting point. We continue to reevaluate the gradient at our new point and "step" in that direction until we reach the halting criteria described above.

7 Previous Work

Some of the previous work that applies to this final project are bilinear interpolation, finding critical points, height and color visualizations and scalars. In past homework assignments executed all of these visualization techniques and that past knowledge assisted with this project. Homework 2 specifically asked for critical points such as saddle points and also asked for scalars, which was really useful for our classifying and finding of critical points. We also used a good amount of the ideas from homework 3, but the difference between this project and Homework 3 is that we are visualizing scalars instead of vector flow. Thus, instead of drawing separatrices in the phase plane using the vector assigned at each point, we wanted to draw integral lines using our gradient vector.

8 Results

Our final outputs are shown in the figures below. Figure 1 shows Mode 1 which contains the data set with no height and classified critical points.The maximums are colored green, the minimums are colored red and saddle points are colored blue. Figure 2 shows the data set with height added and the critical points with their

respective colors. Figure 3 shows the Morse-Smale expected outcome on the data set with height and classified critical points.

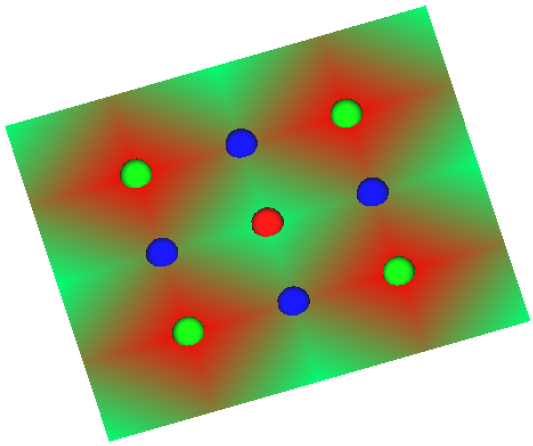


Figure 3: Mode One

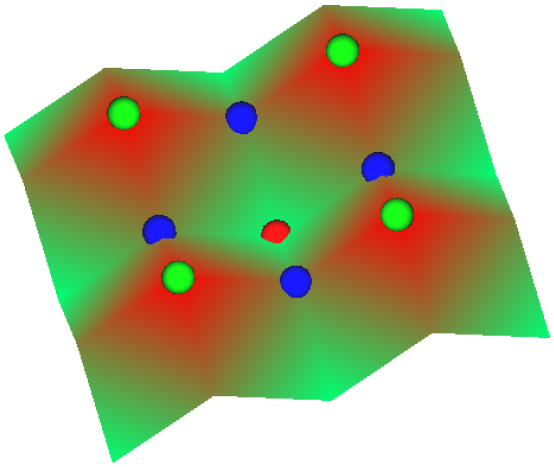


Figure 4: Mode Two

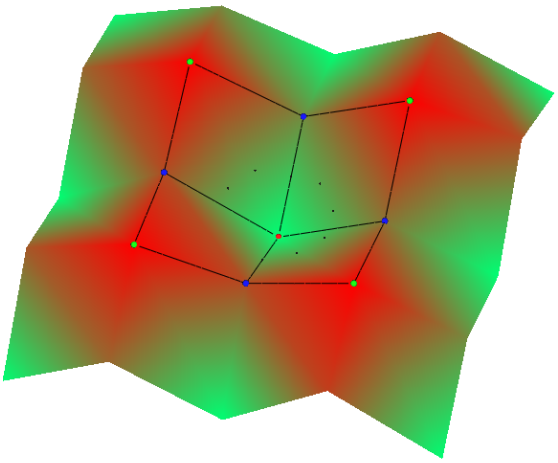


Figure 5: Morse-Smale Visualization

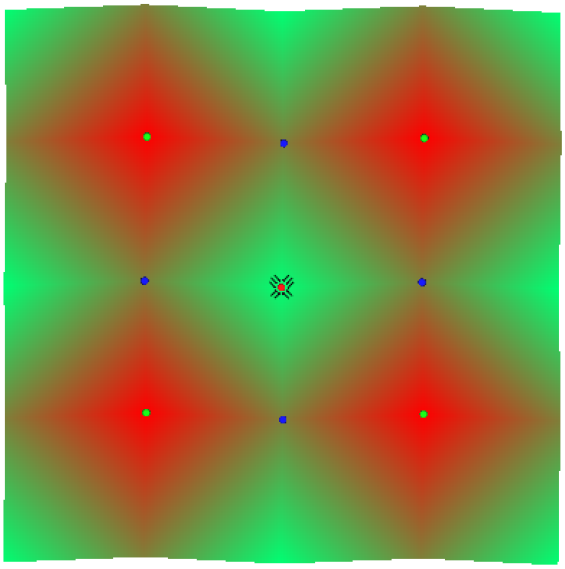
9 Interpretation of Visualization

The figures above show the variety of visualizations outputted in this project. As mentioned previously, Figure 3 shows Mode One, Figure 4 shows Mode Two with only critical points and Figure 5 shows the Morse-Smale complex applied to our data set. As seen above, the different types of critical points are clearly visualized. Our data set mimics mountains so wherever the mountains have the greatest height, there is a maximum critical point there. So, with our data set there are four maximums. The local minimums of the mountains contain the minimum critical points. The global minimum of the mountains contains the saddle point. A saddle point is the point where all of the derivatives in orthogonal directions are zero. As seen above there is only one saddle point in our

10 Conclusion

In the end, our visualization was successfully implemented for our data set, and would likely extend to other data sets. We ended up having to make several updates after a lot of debugging, including verifying that our circle points algorithm was working, checking the gradient function at several random places, make changes to our parameters, and lastly showing the circle points with their gradient vectors, where our code was going awry for a long time. Because of the very flat structure of

our data set, we kept having straight lines up to the maximum points that didn't go near the saddle points at all (see below).



We added a workaround for this by starting at points near our minima or maxima that aimed at all of the saddle points and drew an integral line from there. This allowed us to get the necessary closeness to saddle points to meet our criteria and end up in the visualization.

Ultimately, we developed the tools and designed the algorithms we would need to visualize the Morse-Smale complex, and ended with a Morse-Smale visualization for our data set.

11 Bibliography

Chen, Guoning. "Morse-Smale Complex." University of Houston, Spring 2013.