Teaching Marching Squares with the Modern Web

Roger Ngo*

University of Illinois at Urbana-Champaign

ABSTRACT

Web frameworks in the modern day have made it accessible to create many appealing visualizations to communicate narratives to readers. With modern web development, we not only can tell narratives, but develop teaching tools to initiate engaging practical examples to convey theoretical concepts. This paper discusses an implementation of the Marching Squares algorithm at a conceptual level and shows a practical approach using HTML5 and ReactJS.

Keywords: Marching Squares, 2D HTML5 Canvas, React.js, Web Applications.

1 INTRODUCTION

Marching Squares is a graphics algorithm which generates contours within a two-dimensional scalar field. In this paper, I will describe the algorithm at the conceptual level, followed by a practical implementation in HTML5 and ReactJS which can be used as a tool to analyze the behavior of the algorithm itself when various input parameters are assigned. A reference implementation may be found: https://urbanspr1nter.github.io/marchingsquares/

Leveraging web technologies to teach, and understand various topics has become increasingly important, especially as of 2020, where the world is experiencing the need for more accessible materials for distance learning. By creating interactive web applications paired with content explaining the concept in a step-by-step manner, effective delivery of topics can be achieved.

I have chosen to implement the Marching Squares algorithm as a method to demonstrate this because

1. It is non-trivial enough that the algorithm does require substantial explanation and thought for the learner to understand

2. The implementation can yield aesthetic effects, thus allowing appreciation for the topic itself.

2 ALGORITHM

The following sections will explain how to implement Marching Squares.

2.1 Grid

The grid, G comprises of elements where each element is a cell value, G_{ij}. These can be floating point, or even integral. Marching squares is a divide-and-conquer algorithm where each iteration of the algorithm is applied within a $2x2$ sub-grid C of cells C_{ij}.

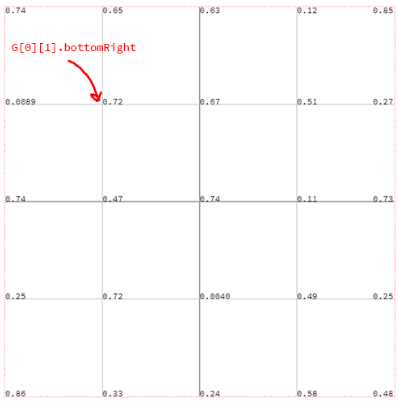

Figure 1: Abstracted grid with cell values.

2.2 Isovalue

Given a grid of cells, the isovalue v is a value which serves as a threshold for which we can determine a vertex state. In the case of Marching Squares, a cell value C_{ij} can be a vertex which may belong in two states:

1. ON (1) – Cell value is equal to, or greater than the isovalue
2. OFF (0) – Cell value is less than the isovalue

2.3 Edges

Edges connect two vertices. The edge endpoints lie between bipolar pairs of vertices. A bipolar pair is when two vertices are in opposite states. A bipolar pair can be visualized like the following:

```
(1) --- (0)
```

Figure 2: Bipolar pair visualized

We can determine the endpoint of an edge by placing the endpoint at the midpoint between two bipolar vertices. Once two endpoints have been found, they may be connected to form the edge. This can be easily found through a midpoint formula, assuming there are two points P_0 and P_1, it can be expressed by:

$$P_0 + (P1 - P0)/2$$

* rngo2@illinois.edu

2.4 Cases

There is a total of 16 configurations in which a $2x2$ sub-grid can represent edges.

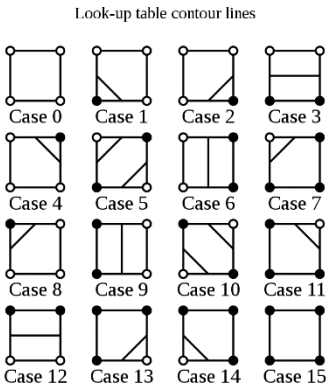

Figure 3: Look-up table of configurations. From Wikipedia.

There are two ambiguous cases which need to be considered: cases 5, and 10. However this is solved in the implementation.

2.5 Main Algorithm

I have now presented enough terminology to explain the algorithm in a step-by-step manner. The first step of the algorithm is to determine the vertex values:

1. Iterate through a grid, G, taking $2x2$ sub-grid samples
2. Obtain the `topLeft`, `topRight`, `bottomRight`, and `bottomLeft` coordinates along with the cell values associated with them.
3. We can now index into the grid G through i and j to find the cell values. The indices represent the row, and column, respectively.
4. Iterate through the grid G to determine which vertex value in comparison to the isovalue v. Set the vertex value at G to 1 if the cell value G_{ij} is at, or above the isovalue, and 0 if below.

Next, a $2x2$ sub-grid can be associated with a configuration.

1. Iterate through G. At each $2x2$ sub-grid, we will form a 4-bit binary code based on the vertex value by iterating clockwise from the top-left corner to the bottom-left corner. As we iterate, we perform a bitwise left-shift to make room for the next bit.
2. Once the binary code has been determined, we now have the configuration. Determine the endpoint between bipolar vertices, and construct the edge based on the case found using the 4-bit code.

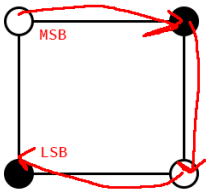

Figure 4: Case construction through clockwise vertex traversal

2.6 Linear Interpolation

With the traditional midpoint approach to building the edges for the Marching Squares algorithm, we may find that the final contoured image may not generate circular contours well. This was chosen that the edge endpoints were placed using the midpoint between the bipolar edges.

Instead, to generate finer contours, we can improve connection of vertices by using linear interpolation to place the endpoint of the edges at a better location based on the isovalue with respect to the original values of the cells.

$$P(t) = (1 - t)P_0 + tP_1$$

We can calculate t from the isovalue v and two vertex values f_1 and f_2 using

$$t = \frac{(v - f_1)}{f_2 - f_1}$$

Once t is found, we can find the position by using P_0 and P_1 position of the cells which contain f_1 and f_2 respectively.

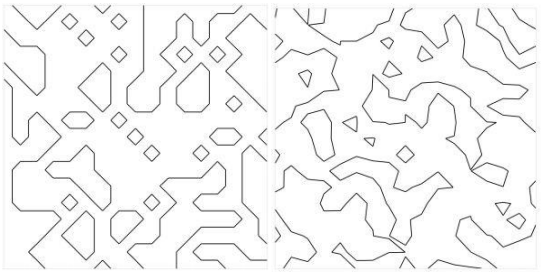

Figure 5: Connections using midpoint formula and using linear interpolation

3 Implementation

The following section is a high-level overview in how Marching Squares may be implemented in a simple web application. All code can be accessed within the GitHub repository: https://github.com/urbanspr1nter/marching-squares

3.1 Technologies

The web application implements an interactive HTML5 canvas which is abstracted a grid of NxN cells with each corner being a cell value. The Marching Squares algorithm is applied onto this grid.

The web application itself is completely run on the client side utilizing the ReactJS JavaScript user interface library. The scaffolding of the application itself is bootstrapped by `create-`

`react-app` to handle transpilation of JSX to standard ECMAScript 6, styling and asset bundling.

3.2 Library Implementation

All code relating to the main algorithm as discussed in Section 2 itself is implemented in the `./src/lib/utils.js` file. Within this file, the following functions are defined:

- `generateSamples` – Generate an NxN grid with cell values ranging from 0 to 1
- `getState` – Logic to determine the state of the cell based on the isovalue. (above or below the isovalue)
- `getCase` – Operate on a list of 4 state values to obtain the 4-bit code to obtain the configuration
- `getT` – Gets the t value from the isovalue, and two bipolar vertices
- `getEndpointByInterpolation` – Obtains the endpoint based on linearly interpolating through the locations of the vertices
- `getEndpointByMidpoint` – Obtains the endpoint based on the midpoint formula between two locations
- `getCoordsForCellAt` – Returns the pixel location for the cell column, and row. Additionally, the cell value is included within the returned result.
- `getEndpointsForCase` – Get all the endpoints to construct the line for a specific case. 16 cases are considered within this function

All functions in the `utils.js` library is accessed within the `Canvas.jsx` component found at the upper-level directory.

3.3 Canvas Implementation

The entry point for the application is at `./src/App.js` and is responsible rendering the page content along with the demo application. The demo application is contained within the `./src/components/Canvas.jsx` component. The point of interesting discussion lies in the latter mentioned component.

`Canvas.jsx` is a React functional component utilizing the library functions to deliver a functional demonstration of the Marching Squares experience to the user. The canvas itself is of 800 pixels in width and 800 pixels in height. There are several controls which make the canvas interactive:

- **Randomize New Set** – Generates a new data set of random cell values given the current grid size for rendering
- **Toggle Grid Overlay** – Toggles the visibility of the grid lines and cell values
- **Interpolation Method** – Specifies the interpolation method in connecting endpoints of the edge lines. Two are available: linear interpolation, and midpoint.
- **Isovalue Slider** – Slider to decrease or increase the isovalue to be used as a threshold. The isovalue can be set from 0 to 1.
- **Grid Size Slider** – Resize the number of cells within the grid from $2x2$ to $128x128$.

For rendering the following flow is used:

1. Generate the sample values based off the default grid size
2. Prepare the canvas by drawing the grid cells, and the values.
3. Render the values by calling the library functions to obtain the state of each cell value and calculate the endpoints.

Upon any user interaction of the UI, event handlers are then triggered to update the internal state managed by React to invoke the initialization flow once more, but with the new parameter values.

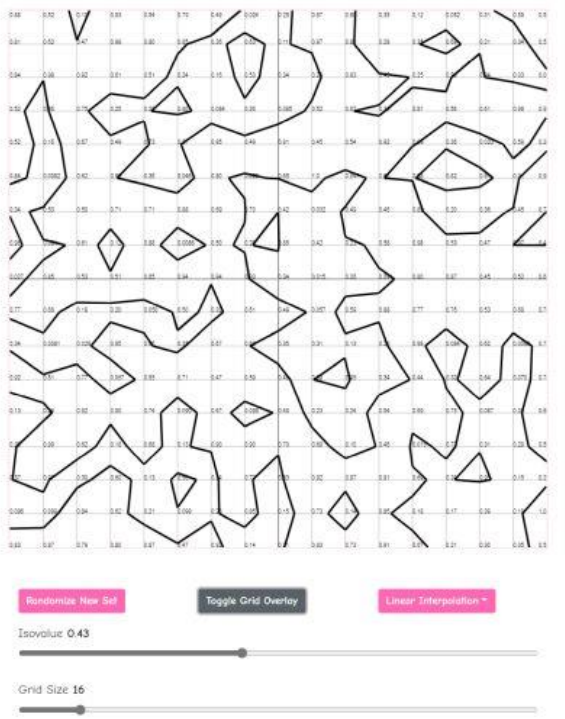

Figure 6: Demonstration of Marching Squares as a web application

3.4 Delivery

The web application is deployed as a static website in GitHub pages. Using the yarn package manager, and create-react-app toolchain, creating the build using the command `yarn build` will create an optimized version of the application.

The application files are then uploaded to a GitHub repository which then is specified to be a static GitHub Pages deployment. This allows economical publishing of web applications intended to serve as teaching tools.

4 Conclusion

With the Marching Squares implementation demonstrated, it is my hope that the method in utilizing web technologies for teaching can be fulfilling as it allows the presenter to deliver material to be understood at high quality. Additionally, modern web tools which can be used for construction such as ReactJS are highly accessible, and low, or even no-cost. It then allows a low-barrier of entry by all who have ability to write client-side code.

References

[1] Wikipedia. Marching Squares. https://en.wikipedia.org/wiki/Marching_squares

[2] J. Wong. Metaballs and Marching Squares. 2014. http://jamie-wong.com/2014/08/19/metaballs-and-marching-squares/