Visualization using Barnes-Hut algorithm

Jordan Force

March 20, 2017

1 Introduction

I've been working on the visualizations for a few months, and have tried many things that failed. I've learned a lot, and I'd like to document that here, in chronological order. I will explain the idea behind the visualizations, and the different techniques I've tried using to bring those ideas to fruition.

2 Visualization Idea

I'd like to color a map in a way that shows the intensity of storms (especially the economic damage caused by those storms) across the map, across some span of time. My initial plan was to split the map into a grid, and color each cell with a color that represents the amount of economic damage that occured within the cell. However, this has several problems associated with it. First, it creates arbitrary divisions. Second, an extreme storm event may only affect a few cells, even though the results of that storm were catastrophic.

Instead, I thought of coloring the storm damage in a way analogous to coloring a surface with the gravitational field on it. For example, this map shows the variation in the gravitational field on the surface of the earth:

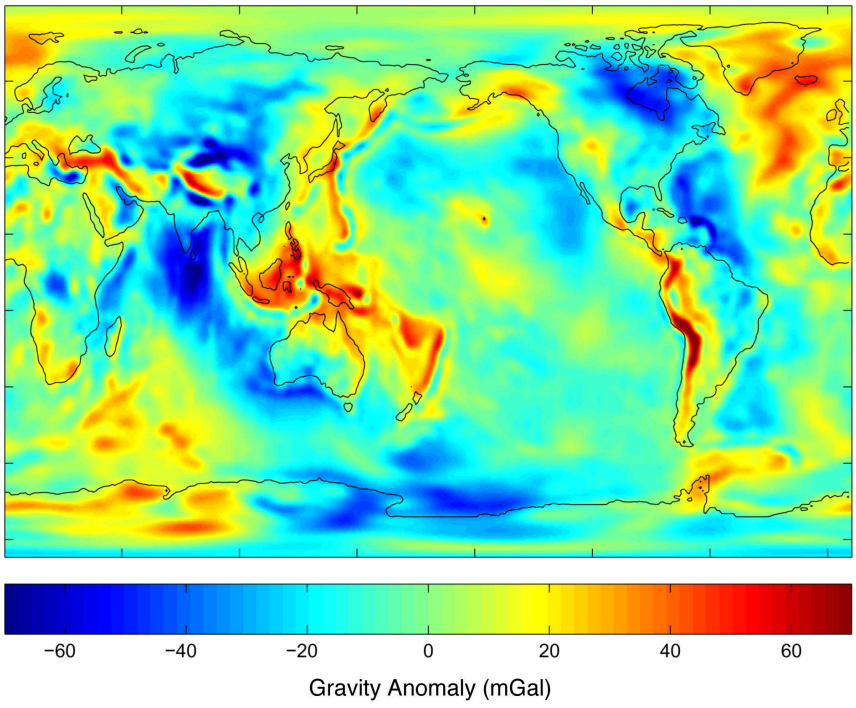

Figure 1: Gravitational field. Courtesy of NASA [3]

There are limitations to this analogy. While a gravitational field is a vector field, our "field" would be a scalar field. Suppose you have two point masses on a plane. Each point has the same mass. If you place one point mass at location $(3, 0)$, and the other at location $(-3, 0)$, then the gravitational field at the origin is zero, since a test particle placed at that location would be pulled toward both point masses with the same force, in opposite directions, thus cancelling each other out. Our visualizations should not have this cancelling effect. That is, the "field" at a point (x, y) due to n storm events, each with weight m_i and at point (x_i, y_i) is:

$$f(x,y) = \sum_{i=1}^{n} \frac{m_i}{\sqrt{(x_i - x)^2 + (y_i - y)^2}^2} = \sum_{i=1}^{n} \frac{m_i}{(x_i - x)^2 + (y_i - y)^2} \tag{1}$$

Of course, this model isn't completely accurate, since earth (despite what some people on the internet might believe) isn't flat. My original plan to deal with this was to project the earth onto a plane, in a very simple manner. Given a set of n points, I calculate the minimum latitude (ϕ_{min}), maximum latitude (ϕ_{max}), minimum longitude (λ_{min}), and maximum longitude (λ_{max}). I use the two points $(\phi_{min}, \lambda_{min})$ and $(\phi_{min}, \lambda_{max})$ as reference. Let d be the distance between these two points on the earth (I will show how to compute this later). The first point has cartesian location $(0, 0)$, and the second one has cartesian location $(d, 0)$. Given a point (ϕ, λ), such that $\phi_{min} \leq \phi \leq \phi_{max}$ and $\lambda_{min} \leq \lambda \leq \lambda_{max}$, I would compute the cartesion location using the distance between this point and the reference point at $(0, 0)$, and the distance between this point and the reference point at $(d, 0)$. I won't go through all of the math, but from these distances, I was able to deduce the x and y coordinates of the point. The following figure shows this:

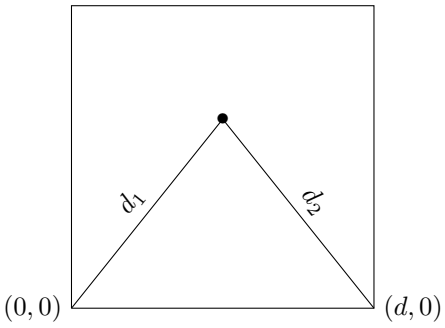

Figure 2: My original projection

To calculate the distance between two points, we use the great circle distance. I originally relied on the definition from Movable Type (known as the Haversine formula)[1], but I am in the process of switching to the definition used in Wolfram Mathematica (as far as I know, the two are equivalent) [4]. We first define the functions:

$$\text{hav}(z) = \sin^2(\frac{z}{2}) \tag{2}$$

$$\text{hav}^{-1}(z) = 2\sin^{-1}(\sqrt{z}) \tag{3}$$

Given the Earth's radius r, and two points $(\phi_1, \lambda_1)(\phi_2, \lambda_2)$, the distance between them is:

$$\text{distance}((\phi_1, \lambda_1), (\phi_2, \lambda_2), r) = r\text{hav}^{-1}(\text{hav}(\phi_1 - \phi_2) + \cos\phi_1 \cos\phi_2 \text{hav}(\lambda_1 - \lambda_2)) \tag{4}$$

3 Barnes-Hut

Coloring in a map would require evaluating the "field" at hundreds, thousands, or even hundreds of thousands of points, depending on the resolution. Our database contains millions of storm events. To speed up these computations, I decided to use an algorithm called the barnes-hut algorithm, originally

developed for speeding up astrophysics simulations. The basic premise is that clusters of objects, that are sufficiently far away, can be treated as a single object. The barnes-hut algorithm creates a quad-tree by recursively dividing up $\mathbb{R}^2$ into quadrants (or $\mathbb{R}^3$ into octants), and forming a tree, where the leafs are the actual objects in the tree. The following two images are courtesy of Samzidat Drafting Company:

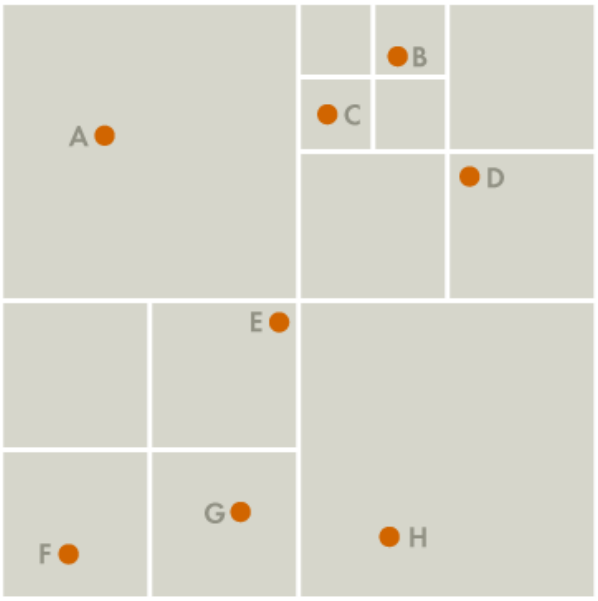

Figure 3: Quadrant being subdivided [5]

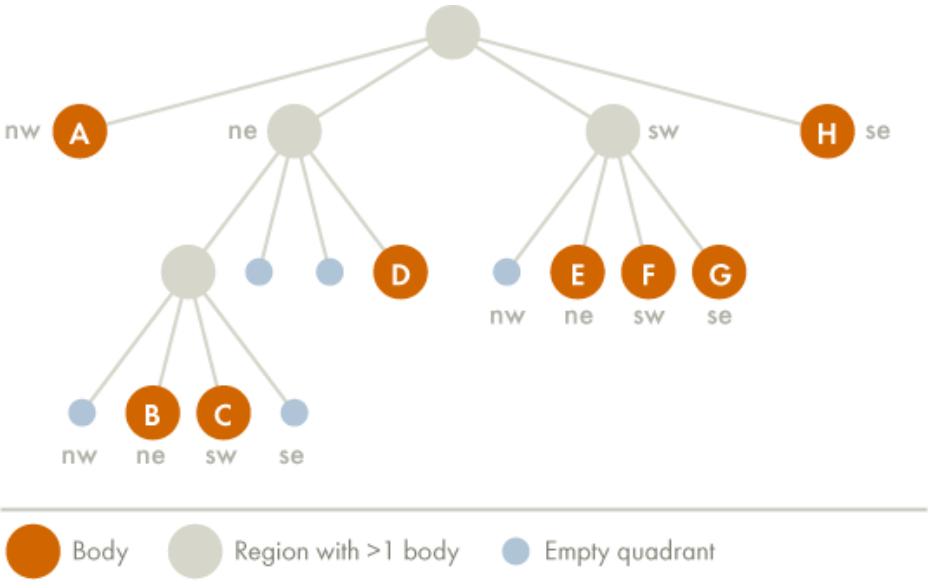

Figure 4: Tree from quadrant subdivision [5]

Each of the internal nodes (colored grey) holds the center of mass of the objects that are children of that internal node. To compute the gravitational field at a point, we traverse the tree, starting from the root, and at each internal node, we let d be the distance from the point to the node's center of mass, and let s be the width of the quadrant that the node represents. If $\frac{s}{d} < \theta$, where θ is some constant (usually θ is 0.5), then we use the total mass of objects in the quadrant, and the center of mass, to approximate the field contribution of the child objects. We do not traverse to the children of this internal node.

My original plan was to take our storm events, project their locations onto a plane, build the barnes-hut tree, and compute the "field" at many points, and then project those sampling points back to latitude and longitude coordinates. However, my projection doesn't preserve distances. That is, given two points with distance d between them, it isn't generally the case that the distance between them, when projected onto the cartesian plane, will remain d. Actually, there are no isometries (transformations that preserve distance) between the surface of a sphere and a plane [2]. Therefore, I need a different plan.

My current plan is to use the Quadrilaterized Spherical Cube (QSC) projection to project the points onto a plane, and then subdivide that plane into quadrants to create the barnes-hut tree. The QSC projection has the property that a region on the sphere maintains its area when projected using the QSC projection [6]. I am using the Proj4 library for doing this projection. Again, since the QSC projection isn't isometric, I need to still work in the latitude and longitude domain when calculating the center of mass for the quadrant. Typically, the center of mass is R such that:

$$\sum_{i=1}^{n} m_i(\vec{r_i} - \vec{R}) = 0 \tag{5}$$

If I was working in normal cartesian coordinates, I would simply re-arrange the equation to calculate $\vec{R}$. However, there isn't an obvious definition for center of mass on the surface of a sphere. Therefore, I let $\vec{r_i} - \vec{R}$ be defined by the initial bearing of the shortest path between $\vec{R}$ and $\vec{r_i}$ and the distance of this path. More precisely, we let:

$$\vec{r_i} - \vec{R} = \text{distance}(\vec{r_i}, \vec{R}) \begin{bmatrix} \cos(\phi_1)\sin(\phi_2) - \sin(\phi_1)\cos(\phi_2)\cos(\lambda_2 - \lambda_1) \\ \sin(\lambda_2 - \lambda_1)\cos(\phi_2) \end{bmatrix} \tag{6}$$

Where $\vec{R} = (\phi_1, \lambda_1)$ and $\vec{r_i} = (\phi_2, \lambda_2)$, and the distance function is defined in equation 4.

Unfortunately, there doesn't seem to be a closed form way to compute $\vec{R}$ from this formulation. But, we can use numerical root finding methods to find $\vec{R}$, that get us very close to an exact solution. I've used the Numpy library in Python to achieve this, and I plan on using these methods in my C code using the GNU Scientific Library.

References

[1] Movable Type Ltd. Calculate distance and bearing between two latitude/longitude points using haversine forumla in javascript. [Online, http://www.movable-type.co.uk/scripts/latlong.html].

[2] Augustin-Liviu Mare. Differential geometry of curves and surfaces: Gauss' theorema egregium. [Online, http://uregina.ca/~mareal/cs6.pdf].

[3] NASA. Earths gravity field : Image of the day. [Online, https://earthobservatory.nasa.gov/IOTD/view.php?id=3666].

[4] Wolfram Technologies. Haversine wolfram language documentation. [Online, http://reference.wolfram.com/language/ref/Haversine.html, see SphereDistance under applications for the function I'm using].

[5] Tom Ventimiglia and Kevin Wayne. The barnes-hut algorithm. [Online, http://arborjs.org/docs/barnes-hut].

[6] Frank Warmerdam and Gerald Evenden. Quadrilateralized spherical cube. [Online, http://proj4.org/projections/qsc.html].