Heightmap generation algorithm?

Asked 15 years, 10 months ago Modified 14 years, 10 months ago

Viewed 9k times

17

I was looking around the internet and couldn't find a perfect algorithm for this particular problem:

Our customer have a set of points and weight data along with each point as can be demonstrated by this image:

[weighted points http://chakrit.net/files/stackoverflow/so_heightmap_points.png]

Of which, we have a GIS program that could generate a "heightmap" or a sort of terrain data from these points and their weight values but as we have near a thousand points of data and that these will change over time, we would like to create our own tools to auto-generate these heightmaps.

So far, I've tried to calculate the weight for each pixel from its distance to the closest data point with `Sqrt((x1 - x2) ^ 2 + (y1 - y2) ^ 2)` and applying weight and distance factor to the data point's color to produce the resulting gradient color for that particular pixel:

[heightmap result http://chakrit.net/files/stackoverflow/so_heightmap_result.png]

You can see that there are still problems with certain configuration of data points and the algorithm sometimes produce a rather polygonal image when there is a lot of data points. The ideal result should looks more like an ellipsis and less like a polygon.

Here is one example image from wikipedia article on gradient ascent which demonstrates the result I want:

[mountains](http://chakrit.net/files/stackoverflow/so_gradient_descent.png)
http://chakrit.net/files/stackoverflow/so_gradient_descent.png

The gradient ascent algorithm is not of my interest. What I'm interested in; is the algorithm to calculate the original function in that picture in the first place, provided data points with weights.

I've not taken any class in topological maths, but I can do some calculus. I think I may be missing something and am rather lost at what should I type in that Google search box.

I need some pointers.

Thanks!

algorithm gis geospatial arcgis heatmap

Share

Improve this question

edited Feb 11, 2010 at 17:45

MarkJ

Follow 30.4k ● 5 ● 71 ● 113

asked Feb 17, 2009 at 15:56
chakrit
61.5k ● 25 ● 136 ● 163

How smooth do you want the terrain generated? Do you have negative as well as positive displacement. I assume the weight refers no the are a 'hill' directly occupies but instead it's maximum point. – ShuggyCoUk Feb 17, 2009 at 16:08

@ShuggyCoUk 1. As smooth as possible without too much complications. 2. All values are positive. 3. Yes, it could interpreted as a maximum point of hills and still be correct. – chakrit Feb 17, 2009 at 16:12

1 @chakrit the image links are broken, plz fix them if possible – Aditya Jul 12, 2013 at 4:57

8 Answers

Sorted by: Highest score (default)

7

What you are looking for is Surface Interpolation.

Some products exist to do this (here's one)

The resulting function/spline/other mathematical construct can then be interrogated at the required resolution to supply the height map.

Your interpolation function

```
Sqrt((x1 - x2) ^ 2 + (y1 - y2) ^ 2)
```

Is similar to Inverse Distance Weighted methods except you are applying an arbitrary filter and discarding many of the other data points.

Most of these techniques rely on a reasonable number of samples and 'terrain-like' behaviour underpinning the values.

I suggest using the weight as the height sample and trying the simple Shepard's Method in the second link (do not filter any pixels to start with) by taking the proportion of a sample points contribution to the overall height value at an interpolation point you can blend the colours of the samples in those ratios to also colour the point. Use the intensity (roughly speaking the grayscale in simple RGB space) to display the height or add contour lines in black like the example image does.

Share Improve this answer Follow

edited Feb 17, 2009 at 16:20

answered Feb 17, 2009 at 16:14

ShuggyCoUk

36.4k ● 6 ● 82 ● 102

Also note that you can render at a lower resolution and do a simple (and quick) bilinear interpolation to produce your final height map, however your tool might do this itself if you supply a low res height map anyway. – ShuggyCoUk Feb 17, 2009 at 17:00

5

This problem is not as easy as it looks on the surface. Your problem is that both sides of the border of two regions need to have the same height, which is to say, the height at a given pixel is determined by more than just one nearest neighbor.

If I understand it correctly, you need at least two algorithms (and a third piece of jargon).

To do this correctly, you need to break the plane into a Voronoi tesselation.

You are probably going to want to use a kd-tree to help you find the nearest neighbor. Instead of taking O(n^2), this will bring it down to O(n log(n)) (the added benefit is that your Voronoi region generation phase will be fast enough in development to work on the height calculation phase).

Now that you have a 2-D map indexing each point to its nearest neighbor i, you need to walk across every x,y point on the map and calculate its height.

To do that for a given point x,y, first grab its nearest neighbor i and stick that into a list, then collect all contiguous regions on the Voronoi diagram. An easy way is to use flood fill to find all the points in the region, then look around the border and collect up the other identities.

Using this list of all the nearest neighbors, you now have a shot at interpolating correctly! (See other answers for interpolation schemes).

Share Improve this answer Follow

answered Feb 17, 2009 at 16:17

Jared Updike
7,268 ● 8 ● 48 ● 73

@Jared ah... If I remove the gradient calculation and just set the color of each pixel to the nearest point's color, I'd get a Voronoi diagram right? – chakrit Feb 17, 2009 at 16:46

@chakrit: Exactly. Just a fancy name for an intuitive concept. In practice I would use a two-dimensional array of ints to track the nearest neighbor and not the color per se, but it's conceptually the same thing and your colors make it possible to actually visualize the thing. – Jared Updike Feb 17, 2009 at 20:19

1 For KDtree + inverse-distance weighting in Python, see stackoverflow.com/questions/3104781/… – denis Jul 21, 2010 at 15:38

4

You've asked for information on algorithms for 2-D interpolation of irregular data, which is quite a complex area. Since you say you have ArcGIS, I **strongly advise** you to interpolate automatically in ArcGIS using its built-in features for automatic calculations. I am sure that will be **far easier** than writing your own interpolation algorithm. I've done some automation of ArcGIS, it's fairly straightforward.

If you do write your own interpolation code - I advise you not to - the first thing is to choose the appropriate algorithm as there are several each with their own pluses and minuses. Here is some advice cribbed from the help for the excellent interpolation tool Surfer (which BTW can also be automated quite easily). There are more algorithms, these are just the ones I've tried.

- **Kriging** is one of the more flexible methods and is useful for gridding almost any type of data set. With most data sets, Kriging with the default linear variogram is quite effective. In general, we would most often recommend this method. Kriging is the default gridding method because it generates a good map for most data sets. For larger data sets, Kriging can be rather slow. Kriging can extrapolate grid values beyond your data's Z range.
- **Inverse distance weighting** is fast but has the tendency to generate "bull's-eye" patterns of concentric contours around the data points. Inverse Distance to a Power does not extrapolate Z values beyond the range of data. A simple inverse distance weighting algorithm is easy to implement but will be slowish.
- **Triangulation with linear interpolation** is fast. When you use small data sets, Triangulation with Linear Interpolation generates distinct triangular faces between data points. Triangulation with Linear Interpolation does not extrapolate Z values beyond the range of data.
- **Shephard's Method** is similar to Inverse Distance to a Power but does not tend to generate "bull's eye" patterns, especially when a smoothing factor is used.

> **Shepard's Method** can extrapolate values beyond your data's Z range.

To implement the algorithms: you can try Googling, or follow the links in some of the other answers. There are some open-source GIS packages which include interpolation, so maybe you can extract the algorithms from them if you like potholing through C++. Or this book by David Watson is apparently considered a classic, although it is a tricky read and the sample code is spaghetti Basic!! But, from what I hear, it's the best available. If anyone else on Stack Overflow knows better, please do correct me as I can't believe it either.

Share Improve this answer Follow

answered Feb 21, 2009 at 20:42

10 Code
20 Fix
Goto 10

MarkJ
30.4k ● 5 ● 71 ● 113

3

Kriging is one of the heavyweight methods for doing this, particularly within the field of GIS. It has several nice mathematical properties - the downside is that it can be slow depending on your variogram.

If you want something simpler, there are many interpolation routines which handle this quite well. If you can get ahold of a copy of Numerical Recipes, Chapter 3 is dedicated to explaining many variants for interpolation, and includes code examples and descriptions of their functional properties.

Share Improve this answer Follow

answered Feb 17, 2009 at 17:02

jasedit
662 ● 4 ● 10

From what I remember Numerical Recipes in C 2nd Edition only includes a routine for 2D linear interpolation. A bit limiting, you might also want to consider Kriging or inverse distance weighting or one of the other methods suggested in other answers. – MarkJ Feb 21, 2009 at 20:15

2

> is the algorithm to calculate the original function in that picture in the first place, provided data points with weights.

It's possible. If you start with single points you will always end up with circles, but if you weight the datapoints and take that into account you can squish the circles into ovals as in the image..

The reason you're ending up with polygons is that you're using a discrete function in your calculation - first you find the closest color, then you determine the color.

You should instead look into gradient algorithms that assigns a color for a point based on the distance and weight from the three datapoints which enclose that point in a triangle.

Gradient algorithm

It depends on what you're trying to display. A simplistic algorithm would be:

For each pixel:

- Find the three points which form the smallest triangle that surround this pixel
- Set this point to the color (HSV color system) that is affected by both the weight and distance to each datapoint:

  ```
  pixel.color = datapoint[1].weight *
  distance(pixel, datapoint[1]) *
  datapoint[1].color +
                disatapoint[2].weight *
  distance(pixel, datapoint[2]) *
  datapoint[2].color +
                datapoint[3].weight *
  distance(pixel, datapoint[3]) *
  datapoint[3].color
  ```

I'm using + here, but you need to determine the 'averaging' algorithm suitable for your application.

-Adam

Share Improve this answer Follow

edited Feb 17, 2009 at 16:23

answered Feb 17, 2009 at 16:13

Adam Davis
93.5k ● 60 ● 271 ● 333

Well, it depends on what you want the outcome to be. Ideally you'd take into account every datapoint in the universe for every pixel, but that is processing intensive, and may not be what you really want. However, it might be what you need (magnetic fields, for instance) – Adam Davis Feb 17, 2009 at 16:17

Beware of iterating over all n points for each of the m pixels in your output map. This is O(n*m), and for a 1000x1000 image and 1000 data points that is a billion operations. This will not scale. Use Shepherd's algorithm to tile the plane or something similar. – Jared Updike Feb 17, 2009 at 16:36

1

Surface interpolation seems to be a hard and mathematical problem. Another, cheaper way to do it is:

```
For each pixel:
For each point:
pixel.addWeight(weight(point, pixel))

def addWeight(w):
totalweight += w
numberofweights += 1
weight = totalweight / numberofweights
```

Example weight function:

`def weight(point, pixel):`

`return point.weight * 1/(1 + sqrt((point.x - pixel.x)^2 + (point.y - pixel.y)^2))`

It quite a brute force approach, but it's simple.

Share Improve this answer Follow

answered Feb 17, 2009 at 16:46

rjprins

1

I implemented something like this in Winamp AVS a while ago. It uses a "metaballs" type approach of calculating the inverse squared distance (to avoid the sqrt for speed) from each data point, capping it (e.g. to 1.0), and taking a sum of those distances for each point on the 2D grid. This will give a smoothly varying colour/height map.

If you want to look at the code, its in the "Glowy" preset from my J10 AVS pack.

EDIT: Just looking at it I added some other jazz to make it look prettier, the part that is most important is:

```
d1=s/(sqr(px1-rx)+sqr(py1-ry));
d2=s/(sqr(px2-rx)+sqr(py2-ry));
d3=s/(sqr(px3-rx)+sqr(py3-ry));
d4=s/(sqr(px4-rx)+sqr(py4-ry));
d5=s/(sqr(px5-rx)+sqr(py5-ry));
d6=s/(sqr(px6-rx)+sqr(py6-ry));
d=d1+d2+d3+d4+d5+d6;
```

Which takes the sum for the 6 points. Everything else done to the red, green and blue output values is to make it look prettier. 6 points isn't much but bear in mind I was trying to make this run in real-time on a 320x200 grid on a 400MHz machine when it was new (which it does at ~20fps). :)

Replace the red =, green = and blue = ... lines with red = d; etc... to see what I mean. All the prettiness goes away and you are left with a greyscale image of smoothly varying blobs around the data points.

Another edit: I forgot to say "s" is the shared weight for all of the points, changing it for each one gives weights individual to each point, e.g. d1 = 2/(...) and d2 = 1/(...) would give d1 twice as much height at its centre as d2. You might also want to cap the expression at the bottom with something like d1 = 2/max(..., 1.0) to smooth off the tops of the points so that they don't peak at infinity in the middle. :)

Sorry for the messiness of the answer... I thought posting the code example would be good enough but on inspection my code is confusing and difficult to read. :(

Share Improve this answer Follow

edited May 25, 2009 at 10:14

answered May 25, 2009 at 10:05

jheriko

3,103 ● 1 ● 22 ● 28

0

You're looking for something that Blender calls "metaballs" (Wikipedia article with links, example). Think of it this way:

Your objects are cones which stick out of the ground. They are all parabolas and the weight tells how far they stick out of the ground. Alternatively, make them all the same height and adjust the "flatness" of the parabola accordingly, so a big weight makes the cone very wide while a low weight makes it sharp. Maybe even both to a certain degree.

I suggest that you implement this and see how it looks.

Next, you need to hang a cloth or rubber sheet over the result. The cloth will stretch by a certain amount and it will generally hang down due to gravity. The cones keep it up.

As long as you are close to the center of a cone, the Z coordinate is just the position on the surface of the cone. As you leave the cone center, gravity starts to pull down and the influence of other cones grows.

Share Improve this answer Follow

edited Feb 17, 2009 at 17:00

answered Feb 17, 2009 at 16:34

Aaron Digulla
328k ● 110 ● 622 ● 834