Linear-Time Surface Reconstruction

Benoît Hudson
Toyota Technological Institute at Chicago
bhudson@tti-c.org

July 6, 2009

Abstract

By now it is moderately well understood how to take as input a set of points that lie on an unknown manifold, and produce as output a piecewise linear approximation to the manifold. The question I explore here is: how fast can we do it? Previous solutions achieve a sequential runtime of $O(n \log n)$ in low ambient dimensions. I show that if the points are specified in floating point coordinates, we in fact achieve linear work, and we can run in logarithmically many parallel rounds.

1 Introduction

The problem of surface reconstruction is to take as input a set of points that lie on an unknown surface, and to produce as output an estimate to the original surface—typically a triangulation. Assuming the point set is sufficiently dense, a number of algorithms have been proposed that can provably solve this problem; see the book by Dey [Dey06]. Early approaches required computing the Voronoi diagram or Delaunay triangulation of the input, which can take quadratic time and space in dimension three or higher. Newer approaches take advantage of the density of the input, which is required for the algorithms to be correct, and show it is sufficient to only compute local pieces of the Delaunay. Funke and Ramos [FR02] gave a relatively complicated construction, much simplified by Funke and Milosavljevic [FM07], for computing a polygonal decomposition that approximates a 2-manifold lying in $\mathbb{R}^d$; it depends only on finding short edges in the Delaunay graph of the input. Cheng, Dey, and Ramos [CDR05] reconstruct k-manifolds lying in $\mathbb{R}^d$ by computing a weighted Delaunay triangulation of a small neighbourhood around each point. Both claim near-linear runtimes and project they can be parallelized.

Here I show how to implement the algorithms more efficiently if the input points are specified with floating point coordinates. The runtime is bounded by the runtime of integer sorting. On a machine with $O(\log n)$-bit words, this is linear work [Rei85]. On a machine with arbitrary word length it is currently $O(n\sqrt{\log \log n})$ in series [HT02] and $O(n\sqrt{\log n})$ in parallel [HS02]. The parallel depth is in $O(\log \Delta)$, where Δ is the geometric spread. I prove $\Delta \in O(n)$ under a common assumption; even failing that, in practice it is often the case that $\Delta \in n^{O(1)}$. The techniques herein apply to any arbitrary dimension, albeit with constants exponential in the dimension (a curse from which the original algorithms also suffer).

Both the surface reconstruction algorithms operate on the k-nearest neighbour graph. Connor and Kumar [CK08] show how to construct it in $O(n \log n)$ work and $O(\log n)$ depth, but we can do asymptotically better by using the balanced quadtree of Bern, Eppstein, and Gilbert [BEG94]. From the quadtree, we can compute the k-nearest neighbour graph in linear work and constant parallel depth. Sampling assumptions imposed by the reconstruction algorithms mean that the quadtree has linear size [HMPS09]. Bern, Eppstein,

and Teng [BET99] showed how to construct it over $\log n$-bit integer input in time equal to comparison-model sorting. I show how to use a word RAM sort, even on floating-point input, thereby improving the work bound. The depth of this step is $O(\log \Delta)$.

Given the k-nearest neighbour graph, it is straightforward to implement the surface reconstruction algorithms mentioned above [CDR05, FM07] in only constant time per point. Less obvious is how to parallelize them. I explain how to do so using a geometric colouring technique derived from Spielman, Teng, and Üngör [STÜ07]. The depth of this step is either $O(1)$ or $O(\log \Delta)$ depending on sampling assumptions.

2 Machine model

I assume a random access machine that operates on words of length w. More precisely, in constant time, the machine can compute any function $f : [0, 1]^w \times [0, 1]^w \to [0, 1]^w$. Beyond the usual arithmetic operations, I need the following operations: computing the position of the highest non-zero bit, and interleaving the bits of d integers to obtain one integer d times longer. The last one can be implemented using $O(d \log d)$ binary operations, a constant number for bounded d. The adversary specifies input in floating point numbers. The mantissa and exponent each are integers of length w.

In the parallel models, we have the ability to use n processors in parallel that share a single memory. The two quantities of interest in an algorithm now become the **parallel depth**, which is the time used if we have n physical processors, and the **work**, which is the time used if we simulate the parallel machine on a single processor. The depth is a slightly fuzzy number; depending on unilluminating details of the machine model, we can suffer a synchronization delay that increases the depth by a factor up to the logarithm of the number of physical processors. I count in rounds, ignoring this factor. After all, when the number of physical processors is very large, the assumption of random access to memory becomes unrealistic.

3 Input assumptions

I assume there exists a closed manifold M without boundary, which we do not know. It lives in $\mathbb{R}^d$, but has *intrinsic* dimension $k < d$. M induces a function on itself, the *local feature size* $\mathrm{lfs} : M \to \mathbb{R}$, which at each point $x \in M$ is the distance from x to the medial axis of M. Frequently, inputs will oversample in at least some areas. To unify the proofs, I allow specifying a function $f : M \to \mathbb{R}$ that is 1-Lipschitz, and everywhere bounded by lfs. Our input is a set of n points $S \subset M$.

Definition 3.1 (Adaptive sample) *S is an adaptive (ϵ, δ)-sample of M if the following conditions hold:*

- *f is a 1-Lipschitz function everywhere smaller than* lfs*.*
- *For each point $x \in M$, there is a sample $p \in S$ such that $|px| \leq \epsilon f(x)$.*
- *For each sample $p \in S$, no other sample $q \in S \backslash \{p\}$ has distance $|pq| \leq \delta f(p)$.*

There exist universal constants for ϵ and δ such that the surface reconstruction algorithms work correctly: they produce surfaces homeomorphic to M, pointwise close, and with approximately correct normals [Dey06]. Many inputs sample nearly uniformly, in which case the algorithms can be simplified:

Definition 3.2 (Regular sample) *Define the **reach** to be $\rho \equiv \min_{x \in M} \mathrm{lfs}(x)$. S is a regular (ϵ, δ)-sample of M if it is an adaptive (ϵ, δ)-sample of M with constant f. That is, $f(x) = f_0 < \rho$ for all $x \in M$.*

Within a regular sample, a ball of radius r contains at most $O(r^d)$ samples. In fact, a ball or radius less than the reach contains only $O(r^k)$ samples, where k is the intrinsic dimension of the manifold. When working with adaptive samples, this is only true locally around each vertex.

Definition 3.3 (Locally regular sample) *S is a locally regular sample of the k-manifold M if it is an adaptive sample, and also there exists η such that for any sample $p \in S$, the ball $B(p, r\,\mathrm{lfs}(p))$, centered at p with adaptive radius $r < \eta$, contains $\Theta(r^k)$ samples.*

Definition 3.4 *The **spread**, Δ, is the ratio between the diameter of S and the distance between the closest pair in S:*

$$\Delta \equiv \frac{\max_{p,q \in S} |pq|}{\min_{p,q \in S, p \neq q} |pq|}$$

Lemma 3.5 *The spread of a regular sample is $O(n)$. The spread of a sample in integer coordinates is $n^{O(1)}$.*

Proof: For a regular sample, the inter-point distance is at least $\delta\rho$ by definition, while the diameter of space is at most $2\epsilon\rho n$ if the manifold is a straight line (plus a small hook somewhere to define ρ). Therefore, the spread is at most $2\epsilon\rho n/\delta\rho = \frac{2\epsilon}{\delta}n$. For an integer sample, the diameter of space is at most $\sqrt{d}2^{O(\log n)} = n^{O(1)}$. The closest pair are at distance at least 1. ∎

An adaptive sample in floating-point coordinates can have spread exponential in n. This is a rather bizarre sample, however. The main way to have large spread is by defeating the sampling assumptions: if we duplicate one point and move it by a very small amount, the closest pair will be very close. This occurs in practice in noisy samples; I mention more about this in the conclusions.

4 Balanced quadtree

Traditionally, a quadtree is a recursive subdivision of the plane that breaks squares into four equal-size subsquares. Here I use the term in general dimension: it recursively divides space into cells that are either leaves or have exactly 2^d children, all of equal size, that tile the volume of the cell. By size of a quadtree cell (or "square"), I mean the length of a side of the square. The root of the quadtree has size triple the diameter of the point cloud, ensuring that no point is close to the border of the root quadtree. The balanced quadtree of a set of points S, as defined by Bern, Eppstein, and Gilbert [BEG94] imposes the following conditions:

(1) no two points of S lie in the same leaf square,

(2) no leaf square that contains a point neighbours a square (leaf or internal) of equal or smaller size that also contains a point,

(3) no leaf square neighbours a square (leaf or internal) of one quarter the size.

Neighbourhood is defined to be through both edges and corners. If we read "square" to mean d-hypercube, the definition generalizes to arbitrary dimension [MV00]. By splitting squares only if they violate one of the three conditions, we produce a balanced quadtree of minimal size.

Lemma 4.1 *The balanced quadtree of an n-point regular or adaptive sample of a manifold has size $O(n)$.*

Proof: Mitchell and Vavasis [MV00] prove that after subdividing each quadtree square a bounded number of times, they obtain a good-quality simplicial mesh whose size is within a constant factor of optimal. Hudson, Miller, Phillips and Sheehy [HMPS09] prove that the optimal-sized quality mesh for a regular or adaptive sample contains $O(n)$ points. The number of simplices in a quality mesh is linear in the number of points, and the number of quadtree squares is linear in the number of simplices, so the quadtree has $O(n)$ leaves and an asymptotically equal number of internal nodes. ∎

exponent	mantissa	expansion
4	5	1010000
3	4	0100000
2	1	0000100

Interleaved expansion: 100 010 100 000 001 000 000

exponent	mantissa
4	100 010 100
1	000 001 000
0	000 000 000

Morton number: (100 100 010 100) (001 000 001 000) (000 000 000 000)

Figure 1: The Morton number for the point $(5 \times 2^4, 4 \times 2^3, 1 \times 2^2)$ on a machine with 3-bit words.

5 Morton Numbers

Bern, Eppstein, and Teng [BET99] showed how to construct the balanced quadtree in $O(n \log n)$ time on a machine with $\log n$-bit integers, if the input is specified using integer coordinates. The construction begins by sorting the points according to the Morton order [Mor66]. The Morton number of a point is the $(d \log n)$-width integer formed by interleaving the bits of the coordinates; the Morton order, also known as the Z-order, sorts the points by these integers. For intuition of how this is useful, consider a point lying in the root quadtree square, $[0, n]^d$. The first d bits of the Morton number (i.e. the first bit of each coordinate) specify in which of the children of the root square we can find the point. In other words, the Morton number specifies a path through an implicit quadtree. We can find the least common ancestor of two points by considering the longest common prefix of the points' Morton numbers.

The Morton number cannot naively be computed for a point with floating-point coordinates because it would occupy $\mathrm{poly}(n)$ bits. However, it is sufficient for our purposes to create an integer for each point that, bitwise, compares correctly according to the Morton ordering. Consider (but do not explicitly represent) the expansion of each coordinate into a very wide integer. Among the d coordinates, there are at most d disjoint substrings of word length each that have any non-zero bits. We can therefore represent a point in floating-point coordinates using d exponents and mantissae, each mantissa being an interleaving of d integers. The total length of this integer, then, is $d(d+1)$ words. See Figure 1 for an example.

Lemma 5.1 *The representation sorts points according to the Morton order.*

Proof: Consider two points p and q. If the first exponent is larger for p than for q (or vice versa), then the leading bit of the interleaving of the flattened floating point coordinates is earlier for p than for q, thus the bitwise ordering is correct. Conversely, if the exponents are equal, the comparing the first set of interleaved mantissae accurately compares the corresponding bits of the flattened representation. If they remain equal, the argument iterates for the lower exponents and mantissae. ■

6 Quadtree construction

Given the points in Morton order, Bern, Eppstein, and Teng [BET99] show how to construct the balanced quadtree in work linear in the number of leaves it contains, and in depth $O(\log n)$ assuming $\log n$-bit in-

tegers. They do not use the bit representation in their algorithm beyond the sorting phase. Using their algorithm, therefore, we can achieve the following:

Theorem 6.1 *We can build the balanced quadtree over an (ϵ, δ)-sample in work equal to the best word RAM sort. The depth is $O(\log \Delta)$ rounds.*

Proof: Because we can generate $O(w)$-bit integers for each of the points that sort equivalently to sorting by the Morton Order, we can sort in time equal to word RAM sorting. The parallel depth is bounded by the depth of the quadtree; the depth is $O(\log \Delta)$, which can be worse than $O(\log n)$ in pathological cases. ■

7 Neighbourhood queries

For any point $p \in S$, define $\mathrm{NN}(p)$ to be the distance from p to its nearest neighbour. Then the **α-neighbourhood** $N_\alpha(p)$ is the set of points within distance $\alpha\,\mathrm{NN}(p)$ of p. In a η-locally regular sample, $N_\alpha(p)$ has bounded size as long as $\alpha \leq \eta$. It is a completely trivial exercise to compute $N_\alpha(p)$ given a balanced quadtree: simply scan through the leaf squares in order of distance, starting at the square that contains p and stopping at radius $\alpha\,\mathrm{NN}(p)$. The η-local regularity assumption implies that all the quadtree squares that will be visited during this scan have size not much smaller than $\mathrm{NN}(p)$, therefore the scan costs constant time for any constant α.

8 Geometric colouring

For the surface reconstruction algorithms, the fundamental parallelization technique I employ is a colouring algorithm inspired by an idea of Spielman, Teng, and Üngör. Assume we have a set of operations to run in arbitrary order, one per point. Further assume that an operation that acts on a point p, whose nearest neighbour is q, only reads data about points within distance $\beta|pq|$, and only writes data to p. On a CREW (concurrent read, exclusive write) PRAM, we can conduct operations on p and p' if $|pp'| \geq \beta|pq|$ and also $|pp'| \geq \beta|p'q'|$, where q' is the nearest neighbour of p'.

Consider a set of points $Q \subset P$ such that all the points in Q have some neighbour within distance h, but no neighbour within distance ι. Given h and ι, the colouring algorithm chooses a colour for each point $q \in Q$ independently of all other points—that is, we can colour in parallel depth 1. The algorithm is as follows: First, divide the coordinates of q by $\iota/\sqrt{d}$ and round to the nearest integer. This assigns unique integer coordinates to each point. Second, take the coordinates modulo $\lceil \beta h/\iota \rceil$. Two points with identical modulo coordinates are at least βh from each other. Thus we have partitioned the points into $\lceil \beta h/\iota \rceil^d$ independent sets. Assuming h and ι are within a constant factor of each other, this is $O(\beta^d)$.

In a regular sample, we know that every point has a neighbour within distance 2ϵ, but no point has a neighbour within distance δ. In an adaptive sample, we can bin the points according to the size of their quadtree cell. Points in a quadtree square of size s have a neighbour within distance $2\sqrt{d}s$ (otherwise, their parent could not have violated any of the three conditions) but none within distance s (otherwise, the square would violate condition (2) and would have been split). There are $O(\log \Delta)$ different possible sizes of quadtree squares.

Lemma 8.1 *Consider a set of operations acting on a set of points that read out to a distance of β times the distance to the nearest neighbour of a point. If the point set is regular, we can colour the operation set into $O(\beta^d)$ independent sets. If the point set is adaptive, we can colour the operation set into $O(\beta^d \log \Delta)$ colours.*

9 Reconstructing a manifold

We can finally discuss how to actually perform the task at hand. I start by showing that the techniques previously discussed can speed up and parallelize extant algorithms.

9.1 Reconstructing a 2-manifold

Funke and Milosavljevic [FM07] show how to use a quadtree so that, given a locally regular set of points in d-space that lie on an unknown 2-manifold, we can reconstruct a triangulation that approximates the 2-manifold. The algorithm is in essence as follows:

- For each unmarked point p, $N_\alpha(p)$, for appropriate α. Mark p as chosen, mark the k neighbours as near p.
- For each chosen point p, draw an edge from a chosen point q if they had a near neighbour in common.
- Use the 1-skeleton to determine the surface using purely local combinatorial predicates.

They prove that this procedure takes constant time per point, plus the construction of the quadtree. In the sequential case with integer input, this means we immediately have an improvement to linear time overall from the merely near-linear time they previously claimed. More importantly, we can parallelize: If points p and q are not in each others' α-neighbourhoods, we can process them in arbitrary order. They write to the vertices in their neighbourhoods, so we can process them in parallel only if their neighbourhoods do not overlap. Colouring with $\beta = 2\alpha$ achieves this. The final computation looks only at vertices within bounded hop distance in the graph. Each hop is at most twice the distance of before, so again, we can colour in linear work with a bounded number of colours if the sample is regular, or $O(\log \Delta)$ colours if the sample is adaptive.

9.2 Reconstructing a k-manifold

The algorithm of Funke *et al* is quite fast in practice on 2-manifolds, but does not extend to higher-dimensional manifolds. An algorithm of Cheng, Dey, and Ramos [CDR05] can reconstruct a k-manifold from an adaptive sample. The main difficulty in reconstructing manifolds of intrinsic dimension 3 or greater is that the restricted Delaunay complex may contain *slivers*, that is, simplices of arbitrarily large aspect ratio (these can exist even if the corresponding Voronoi cells have bounded aspect ratio). Slivers defeat any attempts at producing a triangulation of the manifold with good normal estimation, and must therefore be removed. Cheng *et al* show how to eliminate them via *weight pumping*: computing a weighted Delaunay triangulation and incrementally increasing the weights of the vertices until the slivers disappear. The reconstruction proceeds as follows:

- For each vertex, compute N_α for appropriate α.
- Estimate the dimension k from the neighbourhood $N_\alpha(p)$ of an arbitrary point p.
- For $j = 3$ to $k + 1$
  - For each vertex, increase the weight until all slivers of dimension j are eliminated from the neighbourhood weighted Delaunay triangulation.
- Return the remaining weighted Delaunay k-simplices.

This algorithm runs in constant work per vertex assuming $\alpha \leq \eta$: the neighbourhoods have constant size, so all further operations on them take constant time. Weight pumping can be performed in arbitrary order; the weight a point p chooses depends only on the weights of the vertices in $N_\alpha(p)$. Therefore, we can colour the weight pumping operation using $\beta = \alpha$.

9.3 Main Theorem

Theorem 9.1 *We can reconstruct a k-manifold from an η-local (ϵ, δ)-sample if it is represented using floating point coordinates in the following work and parallel depth:*

assumptions	*work*	*depth*
$O(\log n)$*-bit words, regular sample*	$O(n)$	$O(\log n)$
$O(\log n)$*-bit words, adaptive sample*	$O(n)$	$O(\log \Delta)$
w*-bit words*	$O(n\sqrt{\log \log n})$	*sequential*
w*-bit words, regular sample*	$O(n\sqrt{\log n})$	$O(\log n)$
w*-bit words, adaptive sample*	$O(n\sqrt{\log n})$	$O(\log \Delta)$

10 Conclusions

The sampling assumptions I made do not allow for any noise. Funke and Ramos [FR02] handle spurious oversampling by *decimation*: they eliminate points that are too close together. Dey et al [DS06] extend the idea to handle noise—samples that lie close to but not exactly on the manifold. In essence, the trick is to compute the k nearest neighbours, rather than the α neighbourhood, then keep throwing away the nearest point until the remaining near neighbours are in fact the α neighbourhood. Computing k nearest neighbours is no harder than computing the α neighbourhood in our setting, so we might hope to use the quadtree. However, prior to decimation, the balanced quadtree has superlinear size, thus we cannot construct it in linear time. The algorithm of Connor and Kumar [CK08] can be partially sped up because it starts by sorting the input, but it later has a step that performs work $O(\log n)$ per vertex. It would be interesting to see if we can speed that step up, perhaps by using some kind of noise-aware sampling assumption which Connor and Kumar could not make in their more general setting.

There are other modes of computation that I did not explore here; namely, external memory models, streaming, and distributed computing. All three are similar in that they can benefit greatly from independence based on geometry, which is what I exploit here for the parallelization. I conjecture in particular that by sorting the squares of the quadtree and the points by Morton order, and storing them on disk with a block size of B squares, we can perform surface reconstruction in a total of $O(n/B)$ input/output operations.

References

[BEG94] Marshall Bern, David Eppstein, and John R. Gilbert. Provably Good Mesh Generation. *Journal of Computer and System Sciences*, 48(3):384–409, 1994.

[BET99] Marshall W. Bern, David Eppstein, and Shang-Hua Teng. Parallel construction of quadtrees and quality triangulations. *International Journal of Computational Geometry and Applications*, 9(6):517–532, 1999.

[CDR05] Siu-Wing Cheng, Tamal K. Dey, and Edgar A. Ramos. Manifold reconstruction from point samples. In *SODA*, pages 1018–1027, 2005.

[CK08] M. Connor and Piyush Kumar. Parallel construction of k-nearest neighbour graphs for point clouds. In *Eurographics Symposium on Point-Based Graphics*, 2008.

[Dey06] Tamal K. Dey. *Curve and Surface Reconstruction: Algorithms with Mathematical Analysis*. Cambridge University Press, 2006.

[DS06] Tamal K. Dey and Jian Sun. Normal and feature approximations from noisy point clouds. In *FSTTCS*, pages 21–32, 2006.

[FM07] Stefan Funke and Nikola Milosavljevic. Network sketching or: "how much geometry hides in connectivity?–part ii". In *SODA*, pages 958–967, 2007.

[FR02] Stefan Funke and Edgar A. Ramos. Smooth-surface reconstruction in near-linear time. In *SODA*, pages 781–790, 2002.

[HMPS09] Benoît Hudson, Gary L. Miller, Todd Phillips, and Don Sheehy. Size complexity of volume meshes *vs.* surface meshes. In *SODA*, 2009.

[HS02] Yijie Han and Xiaojun Shen. Parallel integer sorting is more efficient than parallel comparison sorting on exclusive write prams. *SIAM J. Comput.*, 31(6):1852–1878, 2002.

[HT02] Yijie Han and Mikkel Thorup. Integer sorting in 0(n sqrt (log log n)) expected time and linear space. In *FOCS*, pages 135–144, 2002.

[Mor66] G.M. Morton. A computer oriented geodetic data base; and a new technique in file sequencing. Technical report, IBM, 1966.

[MV00] Scott A. Mitchell and Stephen A. Vavasis. Quality mesh generation in higher dimensions. *SIAM J. Comput.*, 29(4):1334–1370, 2000.

[Rei85] John H. Reif. An optimal parallel algorithm for integer sorting. In *FOCS*, pages 496–504, 1985.

[STÜ07] Daniel Spielman, Shang-Hua Teng, and Alper Üngör. Parallel Delaunay refinement: Algorithms and analyses. *IJCGA*, 17:1–30, 2007.