Succinct Representation of Well-Spaced Point Clouds[*]

Benoît Hudson
Toyota Technological Institute at Chicago

Storing n points naively costs $\Theta(nw)$ bits of space on a machine with w-bit words. I show how to reduce the space usage to $O(n)$ bits, independent of the word length. This is an asymptotic reduction: since the n points are all distinct, it must be that $w \in \Omega(\log n)$. The proof is contingent on some sampling assumptions: the points must be well-spaced with respect to some (not necessarily known) subspace. This assumption applies in the case of the input for surface reconstruction, or the output for mesh refinement. In practice, the encoding uses 14 bits per vertex (bpv) for the Stanford bunny, or 18 bpv for the Lucy dataset. This compares to 96 bpv for the uncompressed point set represented with 32-bit float coordinates.

In addition, the data structure is a succinct form of a balanced 2^d-tree [BEG94] (for legibility I call this a quadtree). That is, it supports quadtree operations, such as looking up what leaf quadtree square contains a given point, or what points are contained within a quadtree square. Funke and Milosavljevic [FM07] show how to use these operations to reconstruct a 2-manifold given a set of points in three or higher dimensions. Hudson and Türkoğlu [HT08] show how to use a quadtree to augment an arbitrary set of points in d dimensions and generate a well-spaced point set, that is, one whose Delaunay triangulation will produce a good mesh for scientific applications. Whereas in an uncompressed quadtree, the operations take constant time, in this succinct format, they take $O(\log n)$ time assuming $w \in \Theta(\log n)$: the structure trades a logarithmic slowdown to achieve a logarithmic reduction in memory usage.

I give a lower bound that shows that the encoding is necessarily lossy: a point cloud that matches the sampling assumption can be randomly perturbed to be almost incompressible. The compression technique ensures that the quadtree built over the rounded points is isomorphic to the one built over the original input. I show how to compute the rounding without loading the entire file, uncompressed, into memory; this takes $O(w)$ scans of the input file. While the encoding is lossy, one can easily store additional bits of accuracy. Storing the bunny in 32 bpv allows storing an additional six bits, provably bounding the error interpoint distances to within 6%.

Prior work: It has been repeatedly shown how to compress a point cloud with known connectivity down to 10 to 20 bpv [PKK05, survey]. This matches or beats my result, but has a significant weakness: these formats require that we already know the connectivity. However, the connectivity is precisely what we want to compute in surface reconstruction, and is too expensive to maintain during mesh refinement. Blandford *et al* propose a succinct data structure that computes the Delaunay triangulation while keeping the connectivity information compressed, but the geometry uncompressed [BBCK05]. The results here complement theirs.

Input Assumption: Given a point set P and a compact space S, the *restricted Voronoi diagram* assigns to each $p \in P$ a cell V_p consisting of the set of points $x \in S$ that are closer to p than to any other point $q \in P$. In other words, it is the intersection of the Voronoi diagram of P and the space S. We can now define the input assumption:

Definition 1 *The point cloud P is ρ-**well-spaced** if for all $p \in P$, $\frac{\max_{x \in V_p} |px|}{\min_{q \in P} |pq|} \leq \rho$.*

Well-spacedness arises naturally from my targeted applications. In the mesh refinement problem, the goal is precisely to generate a well-spaced set of points. In surface reconstruction, a common assumption is that the input is an (ϵ, δ)-sample of S. Simple algebra shows that an (ϵ, δ)-sample is $\epsilon/(\delta(1-\epsilon))$ well-spaced.

[*]Long version available as arXiv:0909.3137

Rounding: It it necessary to round because an adversary can take a well-spaced set described using $w/2$-bit coordinates and add $w/2$ random low-order bits, thwarting any asymptotic space savings. I describe a rounding routine that approximately maintains inter-point distances. First we compute the balanced quadtree of Bern *et al* [BEG94], then we round points to the minimum corner of their corresponding leaf quadtree box. The minimum corner of a box is given by the leading bits of the coordinates of the points inside: rounding is masking off a principled number of low-order bits. Given original points p and q rounded to p' and q', the ratio $|p'q'|/|pq|$ is in the range $[(1+2\sqrt{d})^{-1}, 1+2\sqrt{d}]$. Storing γ additional bits improves the approximation ratio to $(1 + 2^{1-\gamma}\sqrt{d})$. Using six additional bits—32 bpv on the Stanford bunny—bounds distortion within $[0.949, 1.055]$.

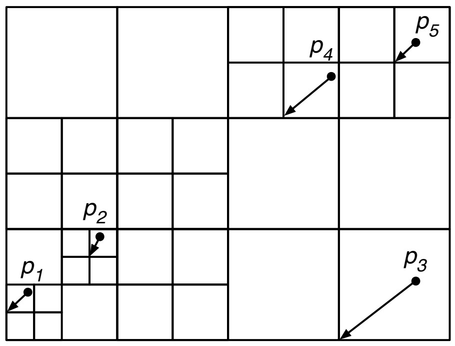

Figure 1: Balanced quadtree of five points in Morton order pointing to their rounded form.

Morton order: The Morton number of a point coordinates is constructed by interleaving the bits of the integer representations of the coordinates; this gives a new, very wide integer. Ordering points by their Morton number gives the Morton order, also known as the Z-order. Comparing two points by their place in the order takes constant time assuming standard bitwise operations, even for floating-point inputs [CK08]. Given an array of points in Morton order, the points contained in a given quadtree box are consecutive. Thus looking up what points a box contains corresponds to performing two binary searches, one each for the minimum and maximum corners of the box. The points in the Figure are numbered according to Morton order.

Compression: Essentially the compressed format stores not the points, but their corresponding quadtree boxes: the depth of the box, and the coordinates of the minimum corner of the box. Since we know the depth of the box, all lower-order bits are zero and need not be represented. Furthermore, the depth and the leading bits do not change much from point to point because they are stored in Morton order. Therefore, it pays to store the data using a difference-encoding: keeping track of the previous point in the order, store only the difference in depth from the prior point, and store only the XOR for each coordinate after rounding. Using a variable-length encoding scheme for these integers allows exploiting the fact that on average they are small.

Space usage: The space usage is proved using a correspondence between the bits of the encoding of the coordinates, and a depth-first traversal of the quadtree. The leading bit of an XOR corresponds to the depth of the least common ancestor of the current box and its predecessor: storage is proportional to the path length. Since the points are stored in Morton order, each quadtree box is entered from above by only one such path: the total number of bits is proportional to the number of quadtree nodes. Hudson *et al* [HMPS09] prove that a minimal-size volume mesh built over a well-spaced surface mesh has only a constant factor more elements. The balanced quadtree forms such a volume mesh, so it has $O(n)$ leaves for n well-spaced points. Ergo, the storage cost is $O(n)$ total bits.

References

[BBCK05] Daniel K. Blandford, Guy E. Blelloch, David E. Cardoze, and Clemens Kadow. Compact Representations of Simplicial Meshes in Two and Three Dimensions. *Int. J. Comp. Geom. & App.*, 15(1):3–24, 2005.

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

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

[FM07] Stefan Funke and Nikola Milosavljevic. Network Sketching or: "How Much Geometry Hides in Connectivity?–Part II". In *SODA*, pages 958–967, 2007.

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

[HT08] Benoît Hudson and Duru Türkoğlu. An efficient query structure for mesh refinement. In *Canadian Conference on Computational Geometry*, 2008.

[PKK05] Jingliang Peng, Chang-Su Kim, and C.-C. Jay Kuo. Technologies for 3D mesh compression: A survey. *Journal of Visual Communication and Image Representation*, 16:688–733, 2005.