Eurographics/ IEEE-VGTC Symposium on Visualization 2008
A. Vilanova, A. Telea, G. Scheuermann, and T. Möller
(Guest Editors)

Volume 27 (2008), Number 3

GPU Local Triangulation: an interpolating surface reconstruction algorithm

C. Buchart, D. Borro and A. Amundarain

CEIT and Tecnun (University of Navarra)
Manuel de Lardizábal 15, 20018 San Sebastián, Spain

Abstract

A GPU capable method for surface reconstruction from unorganized point clouds without additional information, called GLT (GPU Local Triangulation), is presented. The main objective of this research is the generation of a GPU interpolating reconstruction based on local Delaunay triangulations, inspired by a pre-existing reconstruction algorithm. Current graphics hardware accelerated algorithms are approximating approaches, where the final triangulation is usually performed through either marching cubes or marching tetrahedras. GPU-compatible methods and data structures to perform normal estimation and the local triangulation have been developed, plus a variation of the Bitonic Merge Sort algorithm to work with multi-lists. Our method shows an average gain of one order of magnitude over previous research.

Categories and Subject Descriptors (according to ACM CCS): G.1.0 [Numerical Analysis]: General G.1.1 [Numerical Analysis]: Interpolation I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

1. Introduction

Surface reconstruction from a set of points is a well known problem in computer graphics and has a wide range of applications such as image synthesis, modeling and volume rendering. Gopi et al. [GKS00] define the problem as *given a set of points P which are sampled from a surface S in* $\mathbb{R}^3$*, construct a surface* $\hat{S}$ *so that the points of P lie on* $\hat{S}$, assuming there are neither outliers nor noisy points. In other words, it defines an interpolating reconstruction. The resulting surface $\hat{S}$ usually is a set of triangles. If noise is present, an approximating reconstruction is needed in order to avoid interference caused by points off the surface. However, noisy point-clouds do not fall within the scope of this research.

The input data is usually comprised of a set of unorganized points (e.g. see Figure 1) while other, like normals or data structure, can be obtained depending on the acquisition technique employed. In this work, no additional information is assumed to be given, but if present, our method may take advantage of it in order to improve efficiency.

Nowadays improvements in acquisition methods and computers and the demand for more detailed models have led to more and more work with larger point clouds, increasing reconstruction time and memory requirements.

Figure 1: *Turbine Blade (882K) dataset.*

In this paper, we present the GLT (GPU Local Triangulation), an interpolating surface reconstruction algorithm as an extension and improvement to [GKS00] and its implementation in the GPU, which takes advantage of the SIMD

programming model (Single Instruction Multiple Data) to accelerate mesh generation.

The contributions of this paper include:

- SIMD local triangulation, as a GPU-compatible option in surface reconstruction,
- An improvement to the Bitonic Merge sorting algorithm for multi-list sorting and
- A robust GPU implementation of an existing technique for normal estimation.

2. Previous works

Over the last few years, several approximating and interpolating methods for surface reconstruction have been proposed.

The approximating methods of Hoppe et al. [HDD*92], Bajaj et al. [BBX95], Wang et al. [WOK05], Kazhdan [Kaz05] and Jalba & Roerdink [JR06] especially stand out. These are spatial subdivision methods which perform the reconstruction over a voxel representation of the data. [HDD*92] evaluates a signed distance function to tangent planes to select voxels intersected by the surface. [BBX95] uses α-shapes to determine the signed distance function. [WOK05] uses a uniform partitioning of the bounding box of the object. [Kaz05] computes the characteristic function of an oriented point set. [JR06] makes use of regularized-membrane potentials for aggregating points to cells. The problem is then transformed into an iso-surface reconstruction problem, where robust and parallel algorithms have been created, such as [LC87], [Blo94] and [GH95]. Finally, Fleishman et al. [FCAS03] nor Lipman et al. [LCLT07] are not proper surface reconstruction techniques but points-based visualization techniques. [FCAS03] makes use of a projection operator which maps points onto local polynomial approximations and [LCLT07] is a surface approximation operator (formally the Locally Optimal Operator, LOP).

Among interpolating approximations, the use of advancing front techniques (AFT) is a common approach. AFT starts with a minimal subset of final reconstruction and expands its boundaries iteratively to cover the whole surface. Current methods based on AFT perform such iterations one point at a time. Crossno & Angel [CA97] use local triangulations where a point is not marked as finished until it is completely surrounded by triangles. Bernardini et al. [BMR*99] start with a seed triangle and add subsequent triangles using a ball pivoted around the edges of the mesh boundaries.

Another well-known technique is the use of Delaunay triangulations [Del34]. A good example is presented by Attene & Spagnuolo [AS00]. In that work the Delaunay tetrahedralization is constructed and the boundary of the surface is obtained by the use of a Euclidean minimum spanning tree and a Gabriel graph. This and other similar techniques are based on global triangulations algorithms, but Linsen & Prauzsch [LP01] show that local triangulations are also well suited for surface reconstruction and can lead to faster algorithms. Gopi et al. [GKS00] present a local Delaunay triangulation-based method, where points are individually triangulated and the final mesh is obtained by merging all individual fans. Comparing the fan creation methods, while [CA97], [GKS00] and [LP01] triangulate over a radial sorted set of neighbours, they differ in how it is done: the first two methods test if each neighbour is valid in the fan ([CA97] performs a whole empty circumsphere test and [GKS00] determines if they are valid Delaunay neighbours on the tangent plane of the central point) while [LP01] assumes some uniformity of the points cloud and directly adds a few number of neighbours as valid fan points. Amenta et al. [ACK01] propose the use of the inverse transform of the medial axis to reconstruct the surface and Mederos et al. [MAVdF05] modify this algorithm to deal with input of noisy data.

Today much of the research has been devoted to GPGPU programming (General Purpose GPU). The goal of that research is, through the use of *shaders* or *kernels* (GPU programs), to take advantage of the high computational power of modern graphic processors and their specific set of instructions for computer graphics.

Several hardware-accelerated iso-surface visualisation methods have been proposed in the past, but hardware accelerated surface reconstruction is still an open line of research. Klein et al. [KSE04] and Kipfer & Westermann [KW05] implement the marching tetrahedra algorithm in the GPU; in the first research a full indexed tetrahedral representation is used while the second only stores the edges in order to decrease the amount of data processed by the GPU. [JR06] mentioned a GPU implementation of Bloomenthal's implicit surface polygonizer [Blo94]. These works have shown the power of GPU using spatial subdivision methods, but no interpolating implementations have yet been presented. One of the main drawbacks of traditional AFT is their low parallelisation level, due to repeated iterations and the information needed from the current reconstructed mesh. On the other hand, local triangulations are more suitable for parallel work if proper data structures are given.

3. Our proposal - GLT (GPU Local Triangulation)

Our method is inspired by the previous work of [GKS00]. Local triangulations, especially local Delaunay triangulations (LDT), are quite suitable for parallelisation as every point can be processed independently of any other. We took advantage of this property and the high parallelisation level of modern graphics hardware to develop a new, highly-efficient reconstruction method from unorganized points. A general outline of our proposal is shown in Figure 2.

We have based the sampling criteria of models on the Nyquist theorem: the maximum distance δ between two adjacent points must be, at most, half the distance between the two nearest folds of the surface.

First, given an unorganized data set of n points the k-nearest neighbours to each point are computed. Then the problem is divided into smaller sets or *passes*, which are independently processed in the GPU. For each pass, the normal of every point is estimated, their neighbours are sorted and the local Delaunay triangulation is constructed. Finally, all passes are joined into a single mesh and duplicated triangles are removed. In the following sections, the GLT with its data structures and parameters will be described.

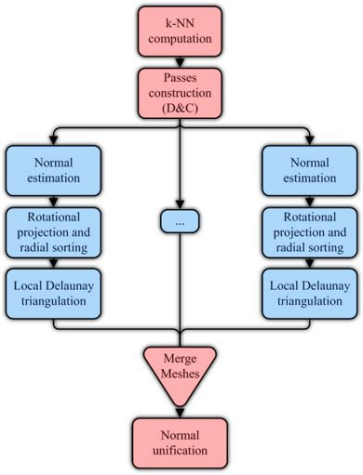


Figure 2: *Flow diagram of our parallel reconstruction algorithm. CPU steps are shown in red and GPUs in blue.*

3.1. Computing the k-NN

In this stage, the k-nearest neighbours to each point $p \in P$ ($Nbhd(p)$ or candidate points of p) is obtained. k-NN is a well known problem in computer graphics, robotics and networking, but in our case the size of data and the fact that the k-NN has to be computed for each point complicates the situation.

By applying a clustering algorithm, the bounding box space of the data set is divided into cells of dimension δ^3 to ease the k-NN computing. For each point a full circumsphere test against the points contained in its cell and those adjacent to it is performed. All the points which fall into the δ-sphere are inserted into a priority queue, which allows us to make an on-the-fly efficient sorting of new neighbours. Currently, k-NN computing is performed in the CPU.

The selection of δ as cells' size quickly guarantees a minimum set of potential neighbours. However, if multiple sampling rates are present, the size must be set to its maximum. k depends mainly on how uniformly sampled the dataset is, the local sampling rate is only important if there are abrupt changes in density. In our experiments with uniformly distributed points, $k = 16$ is quite appropriate. It is important to note that as k does not vary, for non-uniform data sets k must suit the lowest sample rate. As a requirement for the sorting algorithm used, the GPU data structure for candidate points uses a power-of-two size (denoted as $\hat{k}$).

3.2. Parallel triangulation

As mentioned earlier, triangulation is undertaken locally around each point, so that points do not depend on each other. Hence, it can be done in a parallel fashion. As our goal is GPU implementation, data must suit graphic memory. To solve this issue, a divide-and-conquer strategy is applied, splitting the input points into smaller datasets called *passes*. Although it is a similar strategy to that used by Cignoni et al. [CMPS93], our method does not perform a spatial division but uses the storing order of the dataset.

In addition, this approach would allow easy implementation for multi-GPU architectures. Furthermore, all GPU algorithms have been implemented in the fragment shader.

3.2.1. Initial texture structures overview

Before describing individual steps for triangulation, the data structure used in these steps will be discussed briefly. Although points are processed in several passes, information about additional points may be required. To simplify the method, an index of reference points stored in a texture called T_{Points} is used. Normals are stored in a similar texture $T_{Normals}$, but they are updated after each pass.

When storing $Nbhd(p)$, points may have a different number k of candidate points. To reduce structure complexity, neighbourhoods are bounded to $\hat{k}-1$ points, where $\hat{k}$ is a power-of-two. The structure for $Nbhd(p)$ is called T_{Nbhd}. The whole neighbourhood of a point is stored in the same row and the index of each candidate point is used as an offset inside the row. Each group is sorted by the the distance to p and the nearest point is duplicated at the end to close the ring (see section 3.2.6). The size of T_{Nbhd} is a multiple of $\hat{k}$, so only complete rows are stored. We set this size at 512×512, which balances the number of passes (increasing with smaller textures) and the space lost in the last pass (increasing with bigger textures).

Our method processes data in two ways: angle computing and triangulation are done neighbour-by-neighbour, but normal estimation produces a per-point rather than a per-neighbourhood result, rendering the neighbourhood structure of T_{Nbhd} useless. Per-point processing involves the addition of another texture T_P to store the points of the pass, which is used for indexing the neighbours of p in T_{Nbhd}.

3.2.2. Texture assembly

In this step the base data structures, T_{Nbhd} and T_P, are assembled and transferred to the GPU. If a parallel system (multi-GPU architectures, clusters, etc.) is used, it is only necessary to make copies of T_P and assign a set of passes to each node.

3.2.3. Normal estimation

For normal estimation we have implemented a GPU version of the [HDD*92] method which computes the normal for each point of the pass. Another option is the equivalent [GKS00], but it needs a $3 \times k$ matrix, making its GPU implementation much harder and less efficient.

The method proposed by [HDD*92] computes, for each $p \in P$ the centroid o_i of $Nbhd(p)$ and uses principal component analysis (PCA) to estimate the normal $\vec{n}$. Therefore, the covariance matrix CV_i of $Nbhd(p)$ in respect to o_i is formed and, if $|\lambda_1| \geq |\lambda_2| \geq |\lambda_3|$ are the eigenvalues of CV_i, then the eigenvector $\vec{v}_3$ is chosen as the estimated normal.

Normals are not only used in the visualization stage, but are required to project and sort candidate points for triangulation. Numerical errors could become a major problem in this process. As the smallest eigenvalue is the most susceptible to errors due to hardware precision, the tangent plane T_P rather than the normal itself is computed. In other words, the eigenvectors associated with the two largest eigenvalues are computed and the normal is obtained by a cross product.

Given that the result is a per-input-point value, this shader uses the T_P texture as input. The output then, is the same size of T_P and has a normal for each point, which is copied to T_N before going on to the following step.

It is important to point out that the [HDD*92] method must change the normal sign (if needed) to guarantee that nearby points and tangent planes have the same orientation. As LDT is the same, regardless of the normal orientation, we only use the [HDD*92]'s technique at the end of the reconstruction process (CPU). This method views orientation problem as a spanning tree problem where vertices of the graph are the input points, and the connectivity is given by triangulation. An arbitrary point is assumed to have the correct normal and is then propagated from vertex i to vertex j, if $\vec{n}_i \cdot \vec{n}_j < 0$ then the normal $\vec{n}_j$ is flipped.

3.2.4. Rotational projection and angle computation

Following [GKS00], $Nbhd(p)$ onto the tangent plane T_P is mapped. If p_j denotes the j-th candidate point of p and $\vec{p}_j = p_j - p$, then the projected point $\hat{p}_j$ is given by the rotation of $\vec{p}_j$ on the plane defined by $\vec{p}_j$ and $\vec{n}$. In order to reduce the number of instructions, we substitute the projection with the equivalent parallel projection of $\vec{p}_j$ onto TP and then scale the resulting normalized vector by $\left\|\vec{p}_j\right\|$.

Let $\vec{w}_j = (\hat{p}_j - p)/\left\|\hat{p}_j - p\right\|$. The angle of every $< \vec{w}_0, \vec{w}_j >$ pair is computed using the method proposed by [CA97] and described in Formula 1.

$$\alpha_{i,j} = \left\{ \begin{array}{ll} (\vec{w}_0 \times \vec{n}) \cdot \vec{w}_j \geqslant 0 & \arccos(\vec{w}_0 \cdot \vec{w}_j) \\ (\vec{w}_0 \times \vec{n}) \cdot \vec{w}_j < 0 & 2\pi - \arccos(\vec{w}_0 \cdot \vec{w}_j) \end{array} \right\} \quad (1)$$

In this stage T_{Nbhd} is employed to perform this work in parallel fashion. As a result, two additional textures are generated: T_α which replaces T_{Nbhd} with the angle and another with the projected point (T_{Proj}).

3.2.5. Radial sorting

Neighbours are sorted by angle using a modified version of the GPU Bitonic Merge Sort algorithm ([Bat68] and [BP04]), where instead of a single big block of data we sort many small blocks of a fixed size. To sort neighbourhoods smaller than $\hat{k}$, the list is completed with false points with a big angle, so they will always remain at the end of the list.

By taking advantage of single row candidate points packing of the T_{Nbhd} texture, we perform a multi-list sorting with very few modifications: each block of candidate points can be treated as a unique set to be sorted. The original algorithm uses the index of the current element to swap with the other, in our case this can be replaced with the rasterizer X coordinate. As all the sets have the same number of elements $\hat{k}$, they can be sorted in the same number of passes. Additionally, our implementation avoids the use of 1D-to-2D coordinate conversions when reading elements.

3.2.6. Local triangulation

The triangulation step of our method is an improvement of [GKS00]'s localized Delaunay triangulation algorithm. We change the candidate point validation procedure to a more efficient, non recursive process which can be implemented in SIMD architectures – in our case, GPUs.

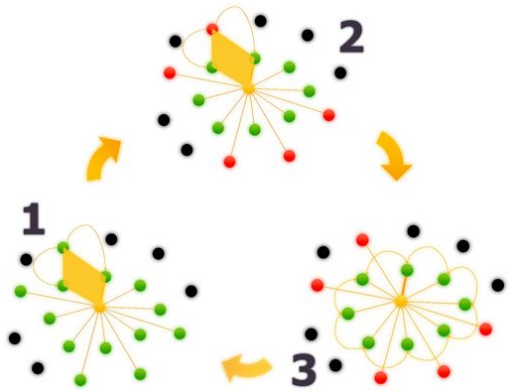


Figure 3: *GLT uses a ring to check all neighbours in parallel, discarding invalid points in each iteration. Here, only one point is shown: given a point and its neighbours (1), this is checked for invalidity (2) and then updates its neighbours (3). Iterations stop when no invalid points are detected.*

[GKS00]'s method verifies if each projected candidate point $\hat{p}_j \in Nbhd(p)$ remains as a valid Delaunay neighbour when compared with $\hat{p}_{j-1}$ and $\hat{p}_{j+1}$; if it is the case it continues with the next point, otherwise it is removed and backtracks to verify the validity of $\langle \hat{p}_{j-2}, \hat{p}_{j-1}, \hat{p}_{j+1} \rangle$. As candidate points are sorted by angle, and are therefore cyclic,

any point can be chosen as $\hat{p}_0$. As the nearest neighbour is always a Delaunay neighbour of p, the process starts with it. In the end, triangulation is constructed with valid points which are consecutive Delaunay neighbours to each other.

The validation function $is\_Voronoi(p,a,b,c)$ (a, b and c are non-invalid consecutive neighbours of p) verifies if the intersection s of the perpendicular bisectors of $\bar{pa}$ and $\bar{pc}$ is not between p and the midpoint of $\bar{pb}$. As Voronoi vertices are defined by intersections of perpendicular bisectors of consecutive neighbours, a false function value means that a Voronoi vertex nearer to p than b exists, discarding b as a Voronoi neighbour of p. We will now prove that *is_Voronoi* will never invalidate a Voronoi neighbour of p.

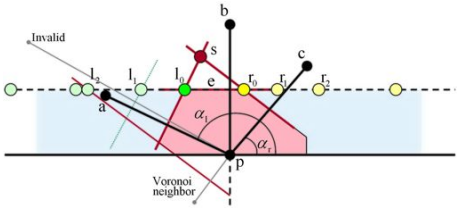


Figure 4: *is_Voronoi never invalidates Voronoi neighbours.*

Let's change our coordinate system as shown in Figure 4, so that $\vec{pb}$ lies on the Y-axis, a on its left and c on its right (if both a and c lie on the same side of Y, b is on a surface boundary and must be included in the triangulation). Now, if b is a Voronoi neighbour of p, then a Voronoi edge $\bar{e}$ must exist and is limited by points l and r, which are the intersections of the perpendicular bisectors of $\bar{pa}$, $\bar{pb}$ and $\bar{pc}$. It is clear that l is the furthest intersection to the right of perpendicular bisectors of the left side, and r is the furthest intersection to the left of the right side, making $r_x > l_x$ for any right and left intersections. Letting α_l and α_r be the angle of $\vec{pa}$ and $\vec{pc}$ respectively, with $\alpha_r \in (-\pi/2, \pi/2)$ and $\alpha_l \in (pi/2, 3\pi/2)$, we can now determine the intersection point height s_y:

$$s_y = e_y + \frac{-cos(\alpha_l)cos(\alpha_r)(r_x - l_x)}{sin(\alpha_l - \alpha_r)} \notin (0, e_y) \qquad (2)$$

This leads to the fact that *is_Voronoi* will never mark a Voronoi neighbour as invalid. To exploit parallelism we take advantage of it by replacing backtracking with simultaneous validations over the ring of candidate points (see Algorithm 1 and Figure 3). In each validation pass h, candidate points are marked as invalid if *is_Voronoi* fails, and next all valid points update their neighbours to valid points (given by *prev_valid_point* and *next_valid_point*). When no changes occur, the local Delaunay triangulation of p comprises the valid points. These operations are performed on a ring structure initially involving the $Nbhd(p)$.

Algorithm 1 Triangulation algorithm.

function TRIANGULATION(j, T^{h-1}_{Ring}) $\Rightarrow T^{h}_{Ring}$
  Let pos_j be the current rasterizer position (x,y)
  Let $r.[next|prev|i|j]$ be the components of T_{Ring}

  $r_j \leftarrow T^{h-1}_{Ring}[pos_j]$
  if r_j is valid **then**
    $r_j.prev \leftarrow prev\_valid\_point(j)$
    $r_j.next \leftarrow next\_valid\_point(j)$
    $r_j \leftarrow is\_Voronoi(p, \hat{p}_{r_j.prev}, \hat{p}_j, \hat{p}_{r_j.next})$
  end if
  return r_j
end function

4. Experiments and results

We performed all our tests on several datasets, ranging from 5,000 to 868,000 points. Our hardware consists of two PCs: an Intel Core2Duo 1.83GHz, 1GB RAM and an nVidia 7950 1GB (just one GPU was used) and an Intel P4 3GHz, 2GB RAM and a nVidia 6800 256MB. In following lines they will be referred to as machines A and B, respectively.

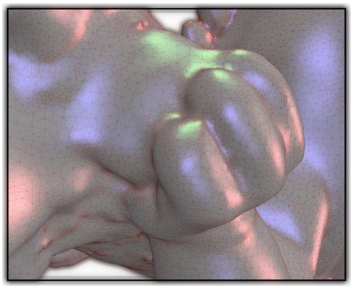


Figure 5: *EG 2007 Phlegmatic Dragon (240K) detail.*

Figure 6: *Noisy dataset – Foot (20K).*

Experiments consisted of analysing the running of GLT on the hardware specified above, varying the neighbourhood parameter k (k=16 and k=32) if needed for a proper result. Low k values are suitable in homogeneous sampled surfaces while a wider range of neighbours ($k \approx 32$) may be required

Model	Points	k	Passes	k-NN (secs.)	Pass Normal Est. (ms)	Pass Angle (ms)	Pass Radial Sort (ms)	Pass LDT (secs.)	Total Pass (secs.)	Total Reconst. (secs.)
EG′07	240,058	16	15	7.35	5.8	0.1	1.2	0.14	0.18	10.27
Dragon				10.85	11.5	0.5	0.1	0.35	0.41	17.26
Reduc. EG′07	57,875	16	4	0.67	1.6	0.3	0.1	0.12	0.14	1.36
Dragon				1.14	9.0	1.2	0.8	0.32	0.37	2.73
Hand	327,325	16	20	4.99	6.3	0.3	0.9	0.13	0.18	8.80
				7.69	9.4	0.4	0.2	0.35	0.41	16.24
		32	41	6.03	4.5	0.1	0.5	0.19	0.23	15.92
				7.87	8.2	0.3	0.4	0.64	0.70	37.07
Dragon	437,646	32	55	18.26	11.6	1.1	5.5	0.19	0.25	32.96
				27.39	8.7	0.1	0.5	0.64	0.70	66.68
Blade	882,955	16	54	12.23	5.3	0.1	1.0	0.14	0.18	23.11
				19.63	8.9	0.3	0.2	0.36	0.42	43.48

Table 1: *GPU algorithm performance. Times for both machine A and B are shown in the first and second lines of each test, respectively. Times from Normal Estimation to the Local Delaunay Triangulation are per pass average. Normal sign propagation is not shown in the table but included in the Total Reconstruction time.*

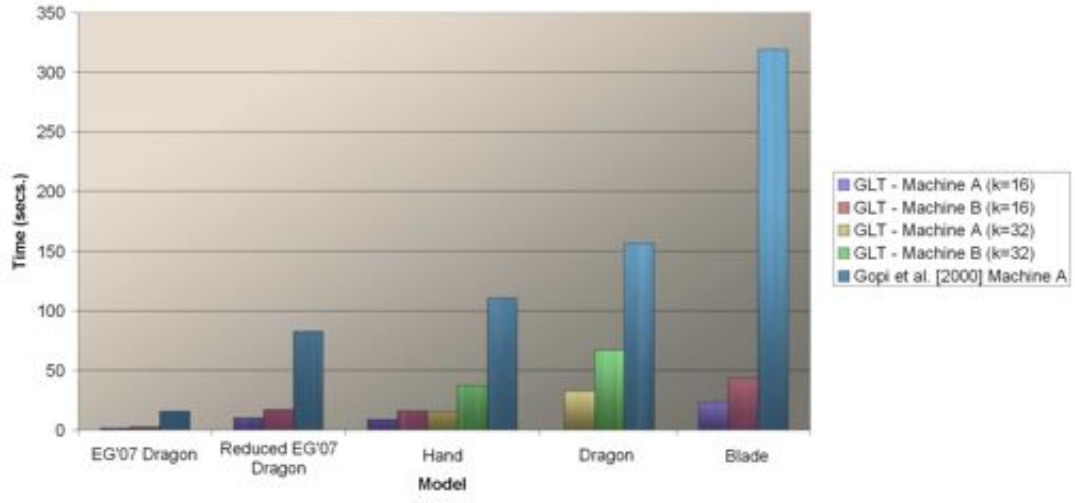


Figure 7: *Comparative graphics of GLT (machines A and B) and [GKS00] (machine A). As some models, such as the Hand and the Dragon, may need more neighbours to reconstruct properly, they need higher k values.*

in datasets with a variable sampling rate (e.g. the EG 2007 Phlegmatic Dragon, see Figure 5) or with oversampling (e.g. the Hand and the Stanford Dragon, skinny triangles are usually avoided by removing the oversampling in a pre-process step). Finally, Delaunay triangulation is not robust against noise as it is an interpolating method. However, datasets with small amounts of noise (e.g., sampling accuracy), can be reconstructed with our method (see Figure 6).

For each model, ten runs were carried out and times were averaged. Results are shown in Table 1 and an overall comparison with a CPU version of [GKS00] run on the machine A can be found in Figure 7. Finally, several reconstructed models are presented in Figure 8, along with an additional example of a noisy dataset.

5. Conclusions

We have presented a GPU implementation of a pre-existing surface reconstruction method. We showed that efficient GPU implementation is possible for an interpolating reconstruction through the use of local Delaunay triangulations. Furthermore, the divide-and-conquer strategy used, not only reduces the time for mesh reconstruction, but provides a memory efficient design. In addition, its design would allow data clustering to use multi-GPU parallelisation. We

have shown some improvements to pre-existing algorithms as well, such as efficient normal estimation through the Deflation method and a multi-list GPU sorting with a modified Bitonic Merge Sort algorithm. On the other hand, the GLT is an interpolating reconstruction and thereby it is not robust against noise and outliers.

Finally, the Locally Optimal Projection operator proposed by [LCLT07] produces noise-free and homogeneous distributed points: ideal characteristics for our method. We plan to integrate LOP in future works as a pre-process step.

6. Acknowledgments

We are thankful to AT&T Labs-Research and the University of California for providing us with the source code from the original work. We also want to thank the Stanford University Computer Graphics Laboratory, the Institute of Information Theory and Automation (Academy of Sciences of the Czech Republic) and the Computer Graphics Group (Czech Technical University in Prague) for providing the datasets for our tests. We are thankful to the Asociación Amigos de la Universidad de Navarra for sponsoring this project.

References

[ACK01] AMENTA N., CHOI S., KOLLURI R.: The power crust. *Sixth ACM Symposium on Solid Modeling and Applications* (2001), 249–260.

[AS00] ATTENE M., SPAGNUOLO M.: Automatic surface reconstruction from point sets in space. *Computer Graphics Forum 19*, 3 (2000), 457–465.

[Bat68] BATCHER K.: Sorting networks and their applications. *Proceedings of AFIPS Spring Joint Computer Conference 32* (1968), 307–314.

[BBX95] BAJAJ C. L., BERNARDINI F., XU G.: Automatic reconstruction of surfaces and scalar fields from 3d scans. *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (1995), 109–118.

[Blo94] BLOOMENTHAL J.: *An implicit surface polygonizer*, vol. 2. Academic Press Professional, Inc., 1994.

[BMR*99] BERNARDINI F., MITTLEMAN J., RUSHMEIER H., SILVA C., TAUBIN G.: The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics* (1999).

[BP04] BUCK I., PURCELL T.: *GPU Gems*. Addison-Wesley, 2004, ch. 37 - A Toolkit for Computation on GPUs, pp. 621–636.

[CA97] CROSSNO P., ANGEL E.: Spiraling edge: Fast surface reconstruction form partially organized sample points. *IEEE Visualization*, October (1997), 317–324.

[CMPS93] CIGNONI P., MONTANI C., PEREGO R., SCOPIGNO R.: Parallel 3d delaunay triangulation. *Computer Graphics Forum 12(3)* (1993), 129–142.

[Del34] DELAUNAY B.: Sur la sphere vide. a la mémoire de georges voronoi. *Bulletin of Academy of Sciences of the USSR 7* (1934), 793–800.

[FCAS03] FLEISHMAN S., COHEN-OR D., ALEXA M., SILVA C. T.: Progressive point set surfaces. *ACM Transaction on Graphics* (2003), 997–1011.

[GH95] GUÉZIEC A., HUMMEL R.: Exploiting triangulated surface extraction using tetrahedral decomposition. *IEEE Transactions on Visualization and Computer Graphics 1* (1995), 328–342.

[GKS00] GOPI M., KRISHNAN S., SILVA C. T.: Surface reconstruction based on lower dimensional localized delaunay triangulation. *Computer Graphics Forum 19*, 3 (2000).

[HDD*92] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Surface reconstruction from unorganized points. *SIGGRAPH 1992 Proceedings* (1992).

[JR06] JABA A. C., ROERDINK J. B. T. M.: Efficient surface reconstruction from noisy data using regularized membrane potentials. *Eurographics/IEEE-VGTC Symposium on Visualization* (2006).

[Kaz05] KAZHDAN M.: Reconstruction of solid models from oriented point sets. *Eurographics Symposium on Geometry Processing* (2005).

[KSE04] KLEIN T., STEGMAIER S., ERTL T.: Hardware-accelerated reconstruction of polygonal isosurface representations on unstructured grids. *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference on (PG'04)* (2004), 186–195.

[KW05] KIPFER P., WESTERMANN R.: Gpu construction and transparent rendering of iso-surfaces. *Proceedings Vision, Modeling and Visualization 2005* (2005), 241–248.

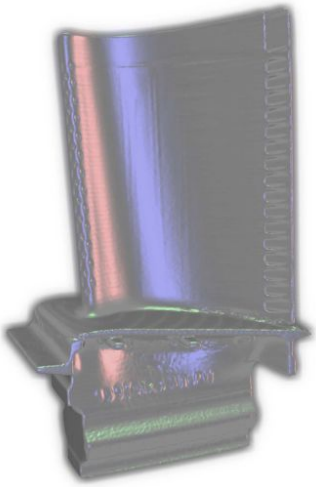
[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH 1987 Proceedings* (1987), 255–282.

[LCLT07] LIPMAN Y., COHENOR D., LEVIN D., TAL-EZER H.: Parameterization-free projection for geometry reconstruction. *SIGGRAPH '07: ACM SIGGRAPH 2007 papers* (2007), 22.

[LP01] LINSEN L., PRAUTZSCH H.: Local versus global triangulations. *Computer Graphics Forum 20* (2001).

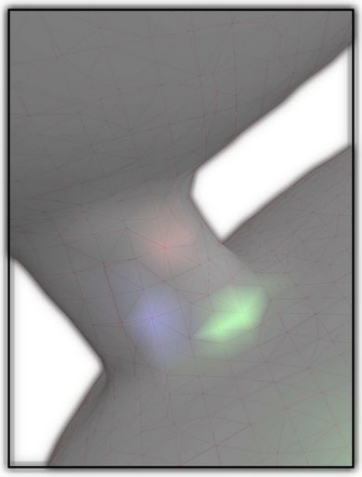
[MAVdF05] MEDEROS B., AMENTA N., VELHO L., DE FIGUEIREDO L. H.: Surface reconstruction from noisy point clouds. *Eurographics Symposium on Geometry Processing* (2005).

[WOK05] WANG J., OLIVEIRA M. M., KAUFMAN A. E.: Reconstructing manifold and non-manifold surfaces from point clouds. *IEEE Visualization* (2005), 415–422.



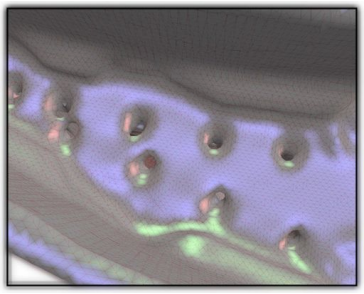
(a) Original dataset. Due to bad sampling some normals are not correctly propagated.

(a) Model generated with k=32.



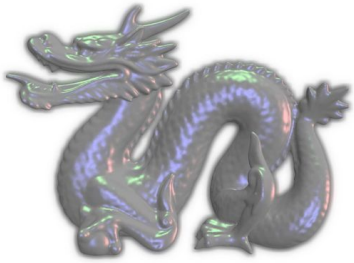
(b) Wrist detail. Note the skinny triangles due to oversampling.

Hand (327K)

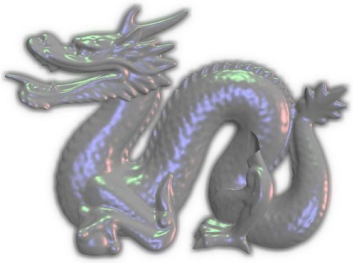


(b) Top detail.

Turbine Blade (882K)

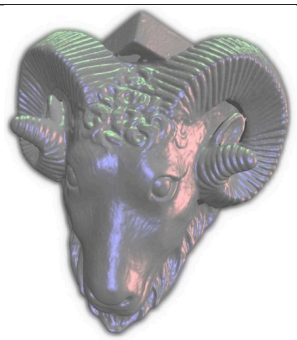


(a) Reconstructed with k=32.

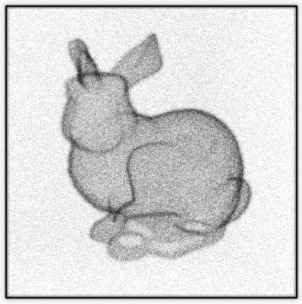


(b) k=16 is insufficient. Note the wrong normal propagation on the back leg.

Stanford Dragon (437K)



Ramorig (622K)



Noisy Stanford Bunny (191K) – High noise is not properly tolerated by GLT.

Figure 8: *Some reconstructed models using the GLT proposed in this work. See Table 1 for detailed times.*