# A GPU interpolating reconstruction from unorganized points (sap_0224)

Carlos Buchart Diego Borro Aiert Amundarain

CEIT and Tecnun (University of Navarra)

We present a GPU method for surface reconstruction from unorganized point clouds without additional information, based on the work of [Gopi et al. 2000]. The main objective of this work is the generation of a GPU interpolating reconstruction method by using local Delaunay triangulations. Existing algorithms accelerated by graphic hardware are approximating approaches, usually based on either marching cubes or marching tetrahedras. Our work most valuable innovation is the GPU adaptation itself; we developed GPU suitable methods to replace those steps which could not to be implemented in graphics hardware. The two most noticeable ones are the $O(log(k))$ method which identifies Delaunay neighbors (replacing the $O(k)$ backtracking) and the sorting of a great number of small lists in $O(log^2(k))$. Additionally, a divide and conquer approach was adopted to suit video memory constrains; passes are then independent, so multi-GPU parallelization can be performed without changes. Our tests show an 11 times average gain over the CPU version.

## 1 Introduction

Surface reconstruction from a set of points is a well known problem in computer graphics and can be defined 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 $S$* [Gopi et al. 2000], assuming there are neither outliers nor noisy points, i.e. $\hat{S}$ defines an interpolating reconstruction. If noise is present, an approximating reconstruction is needed. However, noisy point-clouds are out of the scope of this work, although low-level noise data sets could be treated as noise-free inputs. Our method assumes an input uniformly sampled (a Nyquist based sampling condition is imposed to guarantee a proper reconstruction) and with non-additional information, like normal vectors or points neighborhood (although these can be used to overcome some computing stages).

## 2 Method

The method consists in the construction of local Delaunay triangulations around each sampled point. To perform this construction, several additional information is required: normals and neighborhood for each point. As our input is not assumed to provide them, they have to be computed. The overall reconstruction process is as follows: given an unorganized data set of points, we first compute the $k$-nearest neighbors to each point $p$ (currently it is performed using a general CPU algorithm based on clusters and priority queues); each set of neighbors to $p$ are the *candidate points of $p$* (CP). As video memory and texture size are limited, the complete group of CP is divided into smaller subsets which fit into video memory (from here, these will be referenced as *passes*). These passes are completely independent, so they can be processed in parallel.

To optimize the memory usage in the graphic card, we use an indexed scheme to avoid duplicate point information. For each pass, two textures are employed. The first stores the CP row by row without wrapping to simplify shaders and reduce internal fragmentation. The second one has the same structure but stores the backtracking-replacement supporting structure, described further below. In our experiments, we have observed that a bigger texture size results in a small speed improvement, but not enough to determine an optimum value. We used $512^2$ RECT textures and power-of-two $k$ values (16 for uniformly sampled surfaces, 32 for frequency changing ones).

Each pass is processed in the GPU as follows: we estimate normals using principal component analysis; points are then projected onto the tangent plane (to perform the lower dimensional Delaunay triangulation); radial sorting; and, finally, the local Delaunay triangulation is made. At the end, all the passes are joined into a single mesh removing duplicated triangles.

Some adaptations for the original algorithm include:

- **Radial sorting**: Bitonic sort is a well known algorithm to perform key-sorting suitable for GPU. To preserve previously packed CP information in the pass, angle sorting must be done for all current points in parallel; it becomes a multiple data sets parallel sorting in parallel! To achieve it, the index of each element is stored with the CP and used instead of the rasterizer position of the original implementation. With this, all neighbors of the pass can be sorted in just $log^2(k)$ passes.
- **Local triangulation, backtracking replacement**: in their original method, Gopi et al. [2000] construct a local Delaunay triangulation around each point $p$ as follows: given $q_1 \ldots q_n$, candidate points of $p$ ordered by angle, they perform a simple validation test to a triplet $q_{i-1}$, $q_i$, $q_{i+1}$. If the test passes, $q_i$ is a Delaunay neighbor of $p$, other way, $q_i$ is refused and the algorithm backtracks with $q_{i-2}$, $q_{i-1}$, $q_{i+1}$ to validate $q_{i-1}$. As the nearest candidate point $q_0$ is always a Delaunay neighbor, it is used as a backtracking stop. As known, recursion is not allowed by current graphic hardware. Instead of this, we construct a non-circular candidate points list, duplicating the first point at the end to complete the ring. Candidate points are classified as $Valid$, $Invalid$ and $Static$ (where $Static$ corresponds with the first and last points) and they are associated to the previous and next valid points. Using a multipass iteration, each point marked as $Valid$ is tested with its previous and next nearest valid neighbors to update its validity. When no changes occur in a pass, all the Delaunay neighbors have been found. In the worst case, it takes $log(k)$ passes.

## 3 Results and conclusions

We presented a GPU adaptation of a local Delaunay triangulation algorithm, developing several alternative methods to replace non-GPU suitable ones in the original version. Our tests showed a performance gain between 6 and 18 times over the CPU method, e.g., 35K points models were processed in 1.06 seconds vs. 12.09 seconds. For a large data set test (882K), our method reconstructed it in 23 seconds. Our testing hardware was an Intel C2D 1.83GHz with 1GB of main memory and a 1GB nVidia GeForce 7950 (one GPU active). Finally, the divide-and-conquer-like strategy used to pack passes reduces significantly the memory usage and, in addition, it allows an easy parallelization in multi-GPU systems.

## References

GOPI, M., KRISHNAN, S., AND SILVA, C. T. 2000. Surface reconstruction based on lower dimensional localized delaunay triangulation. *Eurographics 19*, 3, 467–478.