Algorithme de reconstruction surfacique massivement parallèle sur GPU

Encadrants : Dobrina Boltcheva (dobrina.boltcheva@univ-lorraine.fr),
Sylvain Thery (thery@unistra.fr),
Dominique Bechmann (bechmann@unistra.fr)
Laboratoire d'accueil : ICube
Niveau du stage : Master 2 Recherche en Informatique
Durée : 6 mois
Mots clés : reconstruction de surface, programmation GPU, CUDA, OpenCL
Profil : Il n'est pas nécessaire d'avoir des connaissances préalables
en reconstruction de surface, elles pourront être acquises pendant le stage.
En revanche, de bonnes compétences en programmation C/C++ sont nécessaires,
ainsi qu'une curiosité naturelle pour l'informatique bas niveau,
c'est-à-dire proche du matériel.

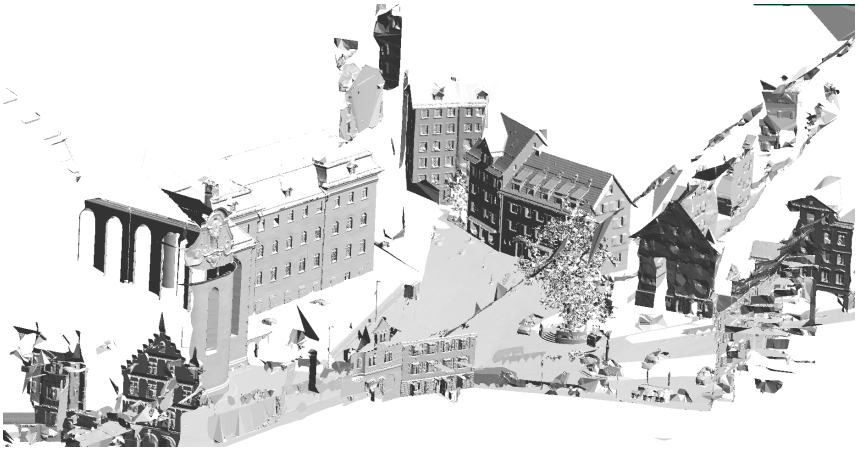


Figure 1: Un nuage de points LIDAR de la place Saint Gallen. Le maillage possède 14 millions de sommets et a été construit en 4 minutes.

1 Programmation GPU

Ces dernières années, l'amélioration des performances des processeurs généralistes s'est faite plus par l'introduction des architectures multi-cœurs que par une augmentation de la fréquence d'horloge, devenue de plus en plus difficile. Cette tendance se retrouve chez les concepteurs de GPU (Graphics Processing Units) qui sous l'influence de l'industrie du jeu vidéo ont produits des processeurs graphiques extrêmement

performants pour accroître l'interactivité et le rendu graphique 3D. Les GPU ont évolué dans le sens de la flexibilité avec l'introduction des pipelines graphiques programmables. On estime que la puissance des GPU double tous les 9 mois et que celle des CPU généralistes tous les 18 mois (loi de Moore).

La puissance brute des GPU actuels est sur le papier très supérieure à celle des CPU. Le processeur haut de gamme GeForge GTX1080 du fabricant NVIDIA contient 2560 cœurs et permet de traiter de l'ordre de 9 Tflops/s (téra (10^{12}) opérations en virgule flottante par seconde) soit deux à trois ordres de grandeur de plus que le processeur Intel Core i7 (6 cœurs), par exemple. D'autre part, les cartes graphiques actuelles sont équipées de leur propre mémoire vive avec une très grande bande passante mémoire (320 Goctets/s pour des cartes avec la puce GeForge GTX1080). Ces performances en font des cibles de choix pour de nouvelles applications en modélisation géométrique.

Le principal reproche généralement fait aux tentatives d'utiliser des architectures spécialisées performantes pour des nouvelles applications plus généralistes est la difficulté de leur mise en oeuvre logicielle. Initialement les GPU étaient programmés en langage assembleur, de très bas niveau. Les nouvelles générations de GPU offrent de nouvelles possibilités aux programmeurs de pipeline graphique avec l'introduction des langages plus haut niveau comme Cg ou GLSL qui permettent d'abstraire le matériel en introduisant les concepts nécessaires à la transformation d'une scène 3D en une image raster 2D. L'utilisation des GPU pour des applications autres que graphiques étaient limitées par le fait de devoir reformuler le problème et les algorithmes en terme de pipeline graphique où les données sont représentées par des pixels stockés dans des textures. Cependant, les outils de programmation ont évolué pour s'abstraire des concepts graphiques [1]. NVIDIA propose l'outil CUDA (Compute Unified Device Architecture) [2] comme environnement de développement et interface de programmation en langage C/C++ (compilateur nvcc) de la dernière génération de GPU. CUDA permet de gérer simplement les transferts de données de la mémoire vive de la carte mère vers celle de la carte graphique via l'interface PCI-express.

Aujourd'hui, le calcul par le GPU permet de paralléliser les tâches et d'offrir un maximum de performances dans de nombreuses applications. Le GPU accélère les portions de code les plus lourdes en ressources de calcul alors que le reste de l'application restant affecté au CPU. Les applications s'exécutent ainsi bien plus rapidement.

Pour comprendre les différences fondamentales entre un CPU et un GPU, il suffit de comparer leur manière de traiter chaque opération. Les CPU incluent un nombre restreint de cœurs optimisés pour le traitement en série, alors que les GPU intègrent des milliers de cœurs conçus pour traiter efficacement de nombreuses tâches simultanées. La difficulté principale lors du portage d'algorithmes sur le GPU vient de la contrainte que tous les threads doivent faire la même chose, ce qui implique que les algorithmes doivent être écrits suivant le principe du *parallélisme par distribution de donnée* (*data parallelism* en anglais).

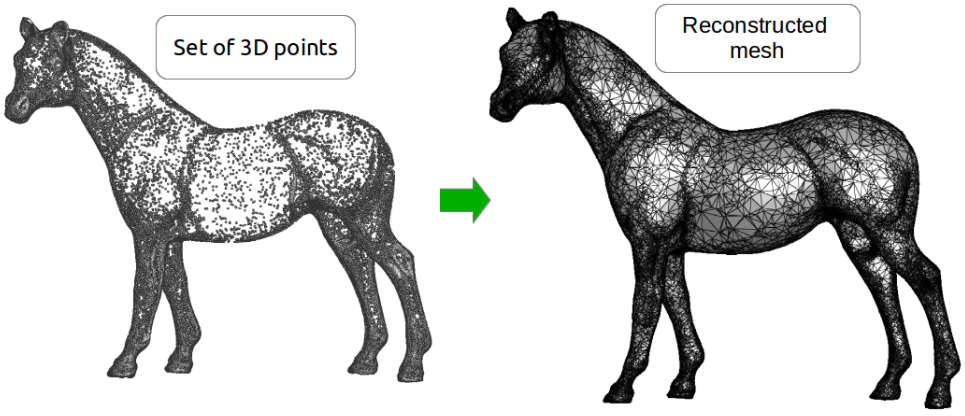
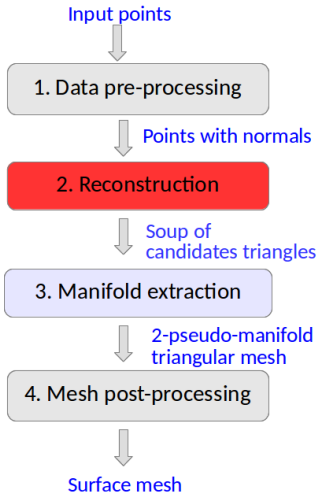


Figure 2: A gauche, nuage de points 3D échantillonnés sur la surface d'un cheval. A droite, la surface reconstruite sous la forme d'un maillage triangulaire connectant les points d'entrée.



(a)

Data: the index i of the point x_i, the disk D_r centered on it, the number of neighbors n and the set of points P
Result: Restricted Voronoi Facet at p_i: $RV(p_i) = V(p_i) \cap D_i^r$

$RV \leftarrow D_i^r$
$R_k \leftarrow \max\{d(p, p_i) | p \in RV\}$
$k \leftarrow 1$
while $d(x_i, x_{j_k}) < 2R_k f$ *and* $k < n$ **do**
  $RV \leftarrow RV \cap \Pi^+(i, j_k)$
  $R_k \leftarrow \max\{d(p, p_i) | p \in RV\}$
  $k \leftarrow k + 1$
end

Algorithm 1: *Computes a restricted Voronoi cell as the intersection between a tangential disk and the Voronoi cell of a point.*

(b)

Figure 3: (a) Schéma général de la méthode. (b) Algorithme de calcul de la facette de Voronoï restreinte en un point.

2 Reconstruction de surface

La reconstruction de surface à partir d'un nuage de points 3D est un problème fondamental et difficile. Il trouve des applications dans de nombreux domaines dans lesquels des outils de numérisation (scanner laser, photogrammétrie,...) sont utilisés tels que la CAO, la fabrication additive, la simulation scientifique et le domaine médical, par exemple.

Le problème de la reconstruction de surface consiste à construire un maillage de triangles connectant des points 3D échantillonnés sur la surface d'un objet, comme illustré sur la Figure. 2.

Récemment, nous avons développé une méthode simple mais puissante permettant de reconstruire de nuages de points de plusieurs millions de points en moins de quelques minutes, [4]. Le schéma général de la méthode est illustré sur la Fig. 3(a).

La composante principale de notre méthode consiste à calculer l'intersection entre le diagramme de Voronoï des points d'entrée avec un ensemble de disques centrés aux points et orthogonaux aux normales estimées de la surface aux points. L'algorithme correspondant est montré sur la Fig.3(b). On remarquera qu'il s'agit de calculer la facette de Voronoï restreinte (par une routine de *polygon-plan clipping*) pour chaque point d'entrée indépendamment des autres, ce qui résulte en un algorithme massivement parallèle par nature. Un tel algorithme devrait donc pleinement tirer parti des architectures multi-cœurs actuelles comme les GPU. Le but de ce stage est donc de porter cet algorithme sur GPU.

References

[1] General Purpose computing with GPU: `http://gpgpu.org/` et `http://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units`

[2] NVIDIA, CUDA framework: `https://developer.nvidia.com/cuda-zone`

[3] OpenCL framework: `https://www.khronos.org/opencl`

[4] Dobrina Boltcheva, Bruno Lévy, "Surface reconstruction by computing restricted Voronoi cells in parallel", SPM 2017, Computer-Aided Design, vol. 90, p.123-134, 2017