Eurographics Symposium on Parallel Graphics and Visualization (2009)
J. Comba, K. Debattista, and D. Weiskopf (Editors)

Parallel Mesh Clustering

Iurie Chiosa, Andreas Kolb, Nicolas Cuntz and Marvin Lindner

Institute for Vision and Graphics
University of Siegen
56076 Siegen, Germany

Abstract

Fast and qualitative clustering of large polygonal surface meshes still remains one of the most demanding fields in mesh processing. Because existing clustering algorithms are very time-consuming, the use of parallel hardware, i.e. the graphics processing unit (GPU), is a reasonable and crucial task in this domain. However, due to the sequential nature of most of these algorithms this is hard to be achieved. In this paper we address the parallel reformulation of the existing approaches and show a suitable GPU implementation for variational or hierarchical parallel mesh clustering.

*A **boundary-based mesh clustering** framework is proposed as a new clustering concept which provides all necessary ingredients for parallel mesh clustering. Here we focus on a specific subtype of the variational clustering algorithm which does not restrict the applicability of the approach as such but reveals much better performance characteristics.*

*A **parallel multilevel (ML) mesh clustering**, for which several dual edges are collapsed in each step, is proposed as an option to the classical ML clustering, where only one dual edge collapse is applied in each step.*

We show how these algorithms can be entirely implemented (giving some non-trivial GPU-specific solutions) and accelerated on GPU.

We demonstrate both approaches applying them to Centroidal Voronoi Diagram (CVD) based clustering. For boundary-based mesh clustering we achieved speed up factors of 10 to 18.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Hardware Architecture—Parallel processing. I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Object hierarchies

1. Introduction

Fast processing of large polygonal surface meshes becomes more demanding as 3D model acquisition systems can nowadays provide models with up to several millions of faces. Most of the existing clustering algorithms, e.g. [CSAD04], [CK08], have a sequential nature of processing, thus for large meshes they become in many cases very time-consuming.

The use of the parallel hardware, i.e GPU, for speeding up these algorithms appears to be a reasonable and at the same time a crucial task in this field.

However, commonly employed mesh clustering algorithm such as *Variational Clustering (VC)* [CSAD04] is not suitable for a GPU implementation. The use of a global Priority Queue (PQ) from which at each step a triangle is popped and assigned to a cluster is the main obstacle in using it for parallel computations.

The recently introduced multilevel mesh clustering approach [CK08] has shown to be a good solution in overcoming the inherent problems of the variational and hierarchical clustering methods. Due to the locality of the decision in the process of the optimization and because no priority queue is used, the *Energy Minimization by Local Queries (EMLQ)* algorithm proposed in [CK08] as a generalization of the approximative Centroidal Voronoi Diagrams [VC04] algorithm, can be used for a parallel clustering. The only drawback of this algorithm lies in its limitation (imposed requirements) on the energy functional. For the hierarchical part of this framework a parallel identification of the optimal

DOI: 10.2312/EGPGV/EGPGV09/033-040

dual edges would substantially improve the performance. However, the procedure of one dual edge collapse in each step, limits a generally required parallelism.

The contributions of our paper are:

- The *boundary-based mesh clustering* algorithm is proposed as a new approach for a parallel cluster optimization. The algorithm redefines the classical variational clustering approach [CSAD04] and makes use of the idea of local queries from [CK08], [VC04]. As a result the new approach avoids using any priority queue for the cluster optimization process. Additionally, no identification of the starting seeds in each iteration is performed. At the same time it preserves the proxy-based clustering concept, thus allowing, as variational clustering, a wider range of energy functionals to be used. In this form the algorithm can be used in the optimization phase of the multilevel mesh clustering [CK08], thus removing the requirement on the special representation of the energy functional for efficient computations.
- A new *parallel multilevel clustering* concept, for which several dual edges are collapsed in each step, is proposed for speeding up the multilevel mesh clustering on GPU.
- We show how both concepts can be entirely implemented on GPU. For that, we propose a new mesh representation to encode the geometry and the connectivity. That gives all necessary means for a complete Variational, Hierarchical and Multilevel GPU-based mesh clustering. We also give some non-trivial GPU-specific technical details.

The paper is organized as follows: After discussing related work (Sec. 2), the Boundary-based Mesh Clustering concept is presented (Sec. 3). Sec. 4 presents the Parallel Multilevel Mesh Clustering approach. In Sec. 5 all GPU-based implementation details for proposed concepts are described. Sections 7 and 8 present results and some final conclusions.

2. Related Work

Variational mesh clustering: Proposed in [CSAD04] as an extension to the Lloyd's algorithm for the extraction of the planar regions. Partitioning and fitting are two alternately repeated steps to minimize the total energy. A notion of a shape **proxy** P_i as a local representative of the cluster C_i which best approximates a given geometry is introduced. In the partitioning step a global priority queue is used to assign triangle to clusters. In the fitting step a new proxy set is recomputed. This is used for identification of the new starting seeds and for performing the next partitioning step. In [WK05], [YLW06] and [JKS05] the same approach was used to fit other shapes such as spheres, cylinders, rolling-balls, general quadric and for identifying quasi-developable surfaces.

Hierarchical mesh clustering: Introduced in [GWH01] for planar approximation of polygonal surface meshes and later employed in [AFS06] for fitting planes, sphere and cylinders. At the beginning a mesh dual graph is created. All created dual edges (DEs) are sorted in a priority queue (PQ). In each step a DE is popped from the PQ and collapsed. The result of this process is a hierarchy of clusters.

Multilevel mesh clustering: Introduced in [CK08] to resolve the inherent problems of the above mentioned Variational and Hierarchical mesh clustering. This approach is a mixture of both, i.e. in each step collapse a DE and subsequently apply an optimization step. The Energy Minimization by Local Queries as a generalization of the algorithm proposed in [VC04] is used for optimization. As a result a complete mesh analysis can be performed and better quality results are obtained. However, high time complexity and imposed requirements on the energy functional are the main drawbacks for a more wider use of this framework.

GPU-based processing: One of the most prominent work in GPU-based acceleration of the iterative (Variational) clustering is the work of Hall and Hart [HH04]. Their work does not address the algorithmical part of the problem instead they propose to use the programmable graphics hardware to perform the most computationally expensive part of the algorithm. Thus they use the CPU for the fitting step, where the GPU is used to perform all necessary point-to-cluster distance metric evaluations. For performing only *one* partitioning step, for each cluster the model data is loaded in a shader constant and used to compute the metric from all points. Although a significant speed up can be achieved with this implementation, for large number of clusters and partitioning steps it does not appear to be so efficient.

It is worth to mention that there are completely GPU-based frameworks such as in [SJP05] or [DT07], to name some. Here we do not refer to these approaches because they are proposed for other mesh processing tasks, rather than for mesh clustering.

Parallel Data Clustering: In past there were many attempts to design parallel algorithms for data clustering, for an example and an overview see [Ols95]. In field of image segmentation a parallel region growing paradigm [WLR88] was introduced. The growing is performed by identifying all possible merge partners and merging regions with mutual choices. Although this idea is proposed in other context we show in Sec. 4 how it can be used for a parallel multilevel mesh clustering.

3. Boundary-based Mesh Clustering

Suppose that an *energy functional* E is provided:

$$E = \sum_{i \in \Omega} E(C_i, P_i) = \sum_{i \in \Omega} \sum_{F_j \in C_i} E(F_j, P_i). \qquad (1)$$

with $\Omega \in \{0, \ldots, k-1\}$.

$E(C_i, P_i)$ is the energy of the cluster C_i for a given proxy P_i and $E(F_j, P_i)$ is the positive semi-definite cost of assigning the face F_j to the cluster C_i with corresponding proxy

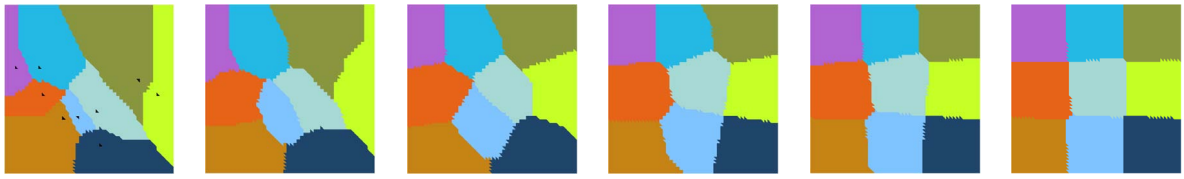


Figure 1: *Boundary-based cluster optimization steps: (init grow) Initialization (black triangles are the starting seeds) and the initial cluster grow. (6 iter.)-(46 iter.) Clustering results for corresponding number of iterations.*

P_i. In the case of a precomputed set of input proxies P_i we call the process of assigning the faces F_j to the clusters C_i a ***proxy-based clustering*** as long as the cluster's proxy is *not* recomputed after each assigned face. In [CK08], a non proxy-based clustering is performed as the cluster proxy is reupdated after each assigned face.

Now suppose that a proxy-based mesh clustering, i.e. partitioning, is performed (see Fig. 1: init grow) and a new set of proxies P_i^{new} is fitted. The *variational clustering framework* [CSAD04] will continue with an identification of the initial seeds for the next partitioning phase. For each cluster C_i this is done by going once through all its faces F_j and identifying the one with the smallest energy $E(F_j, P_i^{new})$ and as close as possible to the cluster center. These seeds are then used to perform a new clustering from scratch.

In practice, one can easily observe that the cluster configuration changes very rapidly during the first iterations and then starts to settle slowly. Secondly, the cluster configuration is mostly affected on the boundary only whereas the cluster's "interior" remains usually intact or changes very slowly (see Fig. 1). Thus, we state that in the process of cluster optimization, regrowing of these "interior" regions (as done in the variational clustering framework) is, in general, not necessary. Regrowing only these boundary bands (strips) will therefore be sufficient to optimize the cluster configurations.

3.1. Algorithm Overview

Based on the observations just mentioned we propose a new ***Boundary-based*** clustering approach which can be summarized in the following three steps:

Initialization: For a given number k of clusters, identify and assign a starting seed for each cluster (Fig. 1: init grow). Usually a random initialization is used.

Initial Cluster Grow: Loop over the *boundary loop (BL)* [CK08] of the cluster and grow the clusters until the entire model is covered (Fig. 1: init grow).

Optimization: Minimize the total energy of the obtained configurations by applying the *Boundary-based Cluster Optimization* algorithm (see Sec. 3.2) until convergence or until a specified number of iterations is reached (see Fig. 1 for a various number of iterations).

The initial cluster grow is done by iteratively assigning the free (not assigned to any cluster) neighboring face of the cluster's boundary edge. Generally, this process is very fast because no energy computation is involved.

3.2. Boundary-based Cluster Optimization

Suppose that an initial clustering configuration (see Fig. 1: init grow) for k clusters C_i with proxies P_i and with corresponding energy $E = \sum_i E(C_i, P_i)$ is given. The basic idea of the algorithm is that the total energy E can be minimized by simply reassigning the cluster's boundary faces to other clusters in such a way that the total energy E decreases.

A simple example of this process for two clusters C_1 and C_2 with proxies P_1 and P_2 and two neighboring faces F_m and F_n of a boundary edge e is presented in Fig. 2. The energies E^0, E^1 and E^2 can be computed as follows:

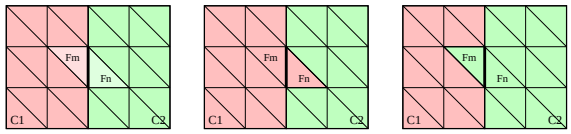
$$\begin{aligned}
E^0 &= \sum_{F_j \in C_1 \setminus \{F_m\}} E(F_j, P_1) + E(F_m, P_1) + \\
&\quad \sum_{F_j \in C_2 \setminus \{F_n\}} E(F_j, P_2) + E(F_n, P_2). \\
E^1 &= \sum_{F_j \in C_1 \setminus \{F_m\}} E(F_j, P_1) + E(F_m, P_1) + E(F_n, P_1) + \\
&\quad \sum_{F_j \in C_2 \setminus \{F_n\}} E(F_j, P_2). \\
E^2 &= \sum_{F_j \in C_1 \setminus \{F_m\}} E(F_j, P_1) + \\
&\quad \sum_{F_j \in C_2 \setminus \{F_n\}} E(F_j, P_2) + E(F_m, P_2) + E(F_n, P_2)
\end{aligned} \tag{2}$$

For comparing the energies E^0, E^1 and E^2 in the above formulas, the sums are irrelevant, thus Eq. 2 can be simplified to:

$$\begin{aligned}
D^0 &= E(F_m, P_1) + E(F_n, P_2). \\
D^1 &= E(F_m, P_1) + E(F_n, P_1). \\
D^2 &= E(F_m, P_2) + E(F_n, P_2).
\end{aligned} \tag{3}$$

For a given boundary edge e the smallest energy D is chosen and a corresponding configuration is updated, i.e.

cluster grow or shrink or no configuration changes, as done in [CK08], [VC04].



(a) (b) (c)

Figure 2: *(a)-(c) Local tests performed for a given boundary edge e (solid line) of two adjacent clusters C_1 and C_2. (a) Initial case: configuration energy corresponds to E^0. (b) C_1 Grows: configuration energy corresponds to E^1. (c) C_1 Shrinks: configuration energy corresponds to E^2.*

Observe that the definition and the computation of D^0, D^1 and D^2 is quite different from that specified in [CK08], although it has the same locality-based check paradigm. This provides the base for performing these checks in parallel. Additionally there is no such strong limitation on the energy functional as in [CK08] where a special energy representation (not easily obtained for most of the existing energy functionals) is required in order to compute efficiently D^0, D^1 and D^2. With this algorithm any energy functional which obey the proxy-based paradigm can be used, thus allowing for a larger set of energy functionals.

When the optimization has finished, a new set of proxies P_i^{new} can be computed and used to perform a new parallel boundary-based energy minimization. Because the energy E is supposed to be positive semi-defined and any boundary modification lowers the energy, the algorithm should converge in general, i.e. for a given set of proxies there are no boundary faces that can be reassigned to other clusters so that the energy decreases.

Although not observed in practice, the algorithm may in some cases (due to precision problem or nonconvexity of the energy) not converge. In these cases a predefined maximal number of iterations can be used.

Observe that in comparison to the classical variational approach [CSAD04] in this approach there is *no* identification of the starting seeds. Additionally, there is *no* local or global priority queue to decide on which face must be assigned to which cluster in the clustering process. This on the one hand leads to a lower computational cost and on the other supports a GPU-based implementation (see Sec. 5.3).

4. Parallel Multilevel (ML) Mesh Clustering

In the multilevel mesh clustering approach in each step a single *Optimal Dual Edge (ODE)* is identified and collapsed, that followed by a cluster optimization. The algorithm described in Sec. 3.2 can be used without modification for optimization phase, thus allowing fast parallel cluster optimization. However, for the hierarchical part of the ML construction *only* a parallel identification of an ODE could speed up the algorithm (see Sec. 5.4 for details).

In [CK08] as a variation of the ML mesh clustering algorithm it was shown that, for a specific region of interest, sequentially merging p% of the clusters can lead to the same quality of the results as obtained when using the standard ML algorithm (only one ODE collapse in each step). Based on this result, we propose to not restrict the ODE collapse to only one collapse in each step but to perform as much as possible in parallel, i.e. a *Parallel multilevel (PML) mesh clustering*.

Thus, the algorithm for a parallel multilevel mesh clustering can be summarized as follows:

```
Loop until # of clusters equals 1
 1. Forall clusters compute ODEs
 2. Collapse mutual ODEs in parallel.
 3. Apply optimization.
```

In the second step all clusters with mutual ODEs (both ODEs connect the same clusters) are merged. Because these merges are independent from each other they can be done in parallel (see Sec. 5.4).

It should be pointed out that, as in [CK08], the algorithm is flexible. It has no limitation on the way in which PML clustering is performed. For some clustering problems one could use the PML with optimization deactivated, i.e. perform parallel hierarchical clustering, or perform p% of ODE collapses with PML clustering and then continue with consecutive ODE collapses for the remaining steps (see Sec. 7 for an example).

5. Technical details

In this section we describe implementation details for *Boundary-based* and *Parallel Multilevel* mesh clustering using GPU-specific techniques. Both proposed frameworks are generic, thus we neglect any energy functional related details. We aim at performing the complete clustering framework entirely on GPU and at reducing as much data transfer between CPU and GPU as possible.

Remark: Our implementation is OpenGL-based mainly due to the fact that we could easily integrate and implement these algorithms in our already existing frameworks. Although a CUDA-based implementation could provide a more structured implementation, our general expectation is that this will not provide substantial performance gains.

5.1. Mesh Representation on GPU

The algorithms presented in Sec. 3 and Sec. 4 are based on local cluster boundary queries. For performing this kind of queries a halfedge data structure [Män88] for mesh representation is required. To our knowledge, for this data structure, there is currently no representation on GPU.

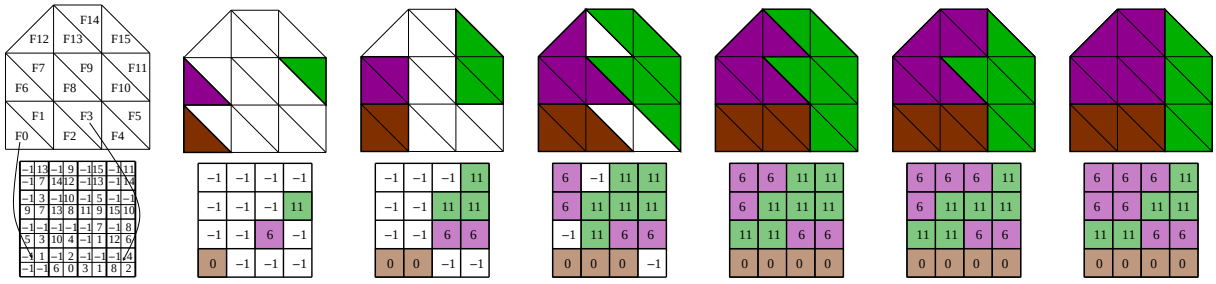


Figure 3: *An example of a Boundary-based cluster optimization steps. (a)*Top*: The mesh with corresponding IDs for faces.* Bottom*: The RGBA* FaceInfo *texture where each texel contains the (ID_{cl}, ID_{n1}, ID_{n2}, ID_{n3}) for each face. (b)-(e) Initial cluster grow. (f)-(g) Boundary-based optimization. (b)-(g)* Bottom*: Shows how the face's ID_{cl} changes in* FaceInfo *texture*

In its simplest form a mesh can be represented (reconstructed) by specifying the coordinates of the vertices together with a set of indices for faces. In general, without rebuilding the entire mesh there is no possibility to obtain any neighboring information for a face, if required.

Our solution to this problem is to save additionally to the vertex coordinates and indices for faces, the corresponding neighbors for each face (see Fig. 3(a)). For a triangular mesh each face has exactly three neighbors, thus these indices can be simply saved in a RGB channel. Using this representation one can compute for each face any required information, e.g. centroid or normal or curvature. It is also sufficient for performing all necessary clustering operations.

Moreover, due to the simplicity of the representation and because we mostly use textures to save any data in GPU memory, we save the input mesh in an image based *pfs* format (see [MKMS07]). Three RGB frames are used to save correspondingly the vertex coordinates, vertex indices and face neighbors IDs. In GPU memory these frames are also loaded in three corresponding textures.

5.2. Processing Concepts and Data Structure

On GPU any entity, e.g. a vertex or a fragment, is processed in parallel and independently from each other. A mesh clustering on GPU must obey the same processing concept. A decision on whether a face must be assigned or reassigned to a different cluster must be taken independently and in an arbitrary order from any decision made for its neighbors, the same is also true for cluster merging decisions. Thus a *per-face* and correspondingly *per-cluster* processing is applied.

Regardless of whether the mesh is clustered or not, each face belongs to a specific cluster with index ID_{cl}. At the beginning each face belongs to a "null" cluster, i.e. it has $ID_{cl} = -1$. We save this information together with face neighboring information (ID_{n1}, ID_{n2}, ID_{n3}) in a RGBA texture *FaceInfo* (see Fig. 3(a)). Here each texel corresponds to a face in the original mesh. Reassigning a face from cluster m, i.e face has $ID_{cl} = m$, to cluster n means that the face will have $ID_{cl} = n$ (see Fig. 3).

For multilevel clustering (Sec. 4) the starting number of clusters k is identical to the number of faces f. In the case of boundary-based clustering (Sec. 3) k is user specified and $k \leq f$ (usually $k \ll f$). Despite this we always assume $k_{max} = f$ and use a corresponding texture size, i.e. the same size as for face textures. This simplifies all cluster data fetches, e.g. for a cluster with index r the data is located in a texture with size *texSize* at the position $(mod(r, texSize), floor(r/texSize))$. For a *Cluster*-defined texture a texel then refers to an individual cluster data.

Additionally, depending on the applied energy functional the data necessary for energy computation is saved in associated *FaceData* and *ClusterData* textures.

5.3. Boundary-based Mesh Clustering on GPU

For a user specified number k of clusters, the steps of the algorithm presented in Sec. 3 are implemented as follows:

Initialization: Randomly generate or load from a file the IDs for starting faces. Reset the corresponding ID_{cl} for starting faces in the *FaceInfo* texture (see Fig. 3(b)).

Initial Cluster Grow: A face with ID^*_{cl} is a cluster ***boundary face*** if at least one of its neighbors has $ID_{cl} \neq ID^*_{cl}$. A cluster can grow only through its boundary faces. Regarded from the actual boundary of the cluster, there are two types of boundary faces : interior $ID_{cl} > -1$ and exterior $ID_{cl} = -1$. Only exterior boundary faces must be added to a specific cluster when growing. In the case when an exterior boundary face can be added to more than one cluster we assign the face to a cluster with smallest ID (see Fig. 3(b)-(e)). An initial clustering is finished if there are no exterior boundary faces that can be further added. An occlusion query can be used in this case to check if any fragment has been written, i.e. if the face ID_{cl} changed or not.

Cluster Optimization: The algorithm proposed in Sec. 3.2 is used in this case. The complete process can be summarized as follows:

```
Loop until samples != 0
  GatherClusterData()
  ComputeClusterProxy()
  samples = OptimizeBoundaryEnergy()
```

To perform a Boundary-based cluster optimization the cluster proxies are required. To compute these, all cluster necessary data is gathered (*GatherClusterData()*) in *ClusterData*. Using the information saved in the *FaceInfo* texture, which actually describes a given clustering, one can use a vertex scattering process with additive blending activated to collect the cluster's data. Using *ClusterData* the cluster's proxy can be computed (*ComputeClusterProxy()*) and the information is saved in *ClusterProxy* texture.

By rasterizing the *FaceInfo* texture, fragments which correspond to an individual face can be generated . Using data from *ClusterProxy* and *FaceData* the D^0, D^1 and D^2 energies (Eq.(3)) can be computed for fragments which correspond to a boundary face. The non boundary corresponding fragments are simply discarded. Remember that as in the case of initial cluster grow the cluster can change its configuration only through boundary faces, i.e interior boundary faces in this case.

The case with the smallest energy must be chosen and the configuration correspondingly updated. Thus if D^0 is the smallest energy the fragment is simply discarded, if D^2 is the smallest, i.e. cluster shrinks, then the fragment ID_{cl} is set to the ID of the opposite cluster. However, if D^1 is the smallest energy, i.e. cluster must grow, the fragment is also discarded because the ID_{cl} for the neighboring face can not be reseted from this point in the program. This limitation has no influence on the optimization because any cluster grow can be seen as a shrink of the opposite cluster, i.e. any required cluster grow will be performed by shrinking the opposite cluster.

After all fragments are processed, a new clustering configuration is obtained. That is used as a starting configuration to complete a new optimization step, i.e. gather cluster data, compute cluster proxies and apply a boundary optimization. This process is repeated until there is no change in the clusters configuration, i.e. no fragment changed its ID_{cl}. An occlusion query is used in this case to check how many fragments (*sample*) were written. For an example see Fig. 3(f)-(g) and Fig. 1.

Observe that the bigger the clusters the more fragments corresponding to interior (non boundary) cluster faces are discarded. Thus, this cluster Boundary-based optimization process is very fast (see timing in Table 1).

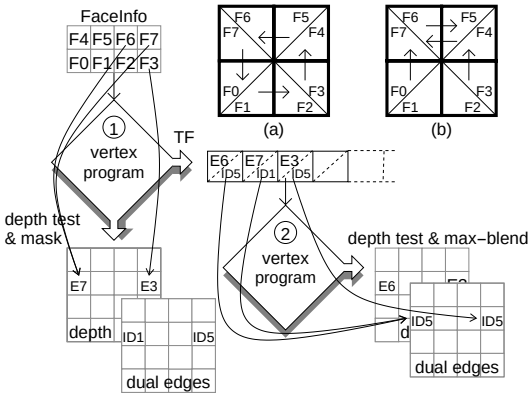


Figure 4: *Parallel multilevel mesh clustering steps example. (a)-(b): A mesh consisting of 8 faces and clustered in 4 clusters. Arrows indicates the DEs. See how applying the second pass the case (b) with two mutual DEs is obtained instead of the case (a).*

5.4. Parallel Multilevel Mesh Clustering on GPU

As a starting configuration for the multilevel (ML) approach, each face is considered as an individual cluster. For a given clustering configuration the algorithm starts with the identification of the optimal dual edges (ODEs), that followed by the collapse operations.

For performing the standard (only one ODE collapse in each step) ML clustering algorithm, only one ODE with the smallest merging energy is required. A work flow of this process can be seen in Fig. 4. The *FaceInfo* texture is loaded into a vertex stream. The vertex program (1) discards all vertices corresponding to the non-boundary faces. For vertices which pass this test, i.e. for each boundary face, a DE is computed. A DE contains the information on collapse cost and the IDs of the two merging clusters. Scattering all generated DEs into only one pixel (not shown in Fig. 4) and using as a fragment depth the DE cost, one can identify the ODE with minimal cost, by performing a depth test.

For performing the parallel ML clustering all mutual DE must be identified. The work flow in this case is identical to the above mentioned ML clustering with only difference. Now the DEs are not scattered into only one pixel but to different pixels which correspond to the face's current cluster ID. However, there can be cases where no mutual ODEs exist although there are possibilities for merging see Fig. 4(a) for an example. This mostly happens in the regions with identical merging energy, because here the direction of the merge is arbitrary. The same problem was pointed out indirectly in [WLR88], they proposed to select the neighbor with the largest ID.

To have this selection implemented efficiently we propose

(a) (b) (c) (d)

Figure 5: *A CVD construction for 15 clusters. (a) & (c) Results of the initialization. (b)&(d) Results after applying Boundary-based mesh clustering*

to use the transform feedback (TF) feature of the GPU to read back all generated DEs (see the sketch in Fig. 4) without recomputing them. These DEs can be scattered (vertex program (2)) to the correct cluster position. However to choose a neighbor with the largest ID, we additionally apply a maximum blending to the ODE's cluster ID. This way, at least one pair of mutual DEs can be obtained (see Fig. 4).

6. Energy Functional

We already mentioned the generic nature, i.e. the independence on energy functionals used, of the algorithms presented in Sec. 3 and Sec. 4. To give an example we show how an approximated Centroidal Voronoi Diagram (CVD) can be performed. This energy is chosen mostly due to the fact that it can be easily implemented and that it requires few cluster data. In our future work we plan to test this framework with other commonly employed energy functionals.

The energy of the approximated CVD can be written as:

$$E_{CVD} = \sum_{i=0}^{k-1} E_i = \sum_{i=0}^{k-1} \sum_{F_j \in C_i} \rho_j \|\gamma_j - P_i\|^2. \qquad (4)$$

where γ_j and ρ_j is the centroid and the weighted area of the face F_j, respectively. $P_i = \sum_{F_j \in C_i} \rho_j \gamma_j / \sum_{F_j \in C_i} \rho_j$ is the cluster centroid (proxy). Thus for computing the D^0, D^1 and D^2 energies (Eq. 3) one uses: $E(F_s, P_r) = \rho_s \|\gamma_s - P_r\|^2$

The dual edge collapse cost between two cluster C_1 and C_2 is computed as in [CK08] using: $DE_{cost} = E_{12} - E_1 - E_2$. For an easier computation of the merging energy, Eq. 4 can be written in a form:

$$E_i = \sum_j \rho_j \|\gamma_j\|^2 - 2P_i \cdot (\sum_j \rho_j \gamma_j) + \|P_i\|^2 \sum_j \rho_j. \qquad (5)$$

Thus in the *FaceData* texture we only need to keep the values $\rho_j \|\gamma_j\|^2$, $\rho_j \gamma_j$ and ρ_j. Correspondingly the *ClusterData* texture stores the following information: $\sum_j \rho_j \|\gamma_j\|^2$, $\sum_j \rho_j \gamma_j$ and $\sum_j \rho_j$.

7. Results and Discussion

The results presented in this paper are generated using a 3GHz Intel Core(TM)2 Duo CPU PC with a GeForce GTX 280 graphics card.

In Fig. 5 the result of a CVD construction is presented. Observe how during optimization the clusters (the green cluster is the most prominent) moved from the left leg of the model to different positions on the model. This shows that the algorithm performs very well even when starting with a "bad" initialization. At the same time observe the perfect symmetry (good visual quality) in the final clustering.

Table 1 provides a timing comparison between CPU- and GPU-based clustering results for different meshes. We compare the standard approximative CVD algorithm [VC04] with our GPU boundary-based approach. Here speedups from 10 to 18 are observed.

Model	# Faces	# Clusters	CPU *ACVD* (sec.)	GPU *BB* (ms)
Bunny	70k	1k	3	172
Bunny	70k	3k	3	187
Horse	97k	1k	3	187
Horse	97k	2k	4	218
Armadillo	346k	2k	12	1140
Armadillo	346k	5k	16	891

Table 1: *Clustering time for building an Approximated CVD (ACVD) with CPU- vs. GPU Boundary-based (BB) cluster optimization*

Table 2 also provides a timing comparison between CPU- and GPU-based clustering results for ML approach. Due the fact that only one dual edge is collapsed in each step for the GPU ML clustering a speedup of a factor 1.5 to 2.5 is only achieved. Using the GPU parallel ML speedup factors of 32 to 150 can be achieved.

Model	CPU, *ML* (sec.)	GPU, *ML* (sec.)	GPU, *PML* (sec.)
Bunny	281	183	9
Horse	710	283	5
Armadillo	10856	6633	73

Table 2: *Clustering time for Multilevel (ML) and Parallel Multilevel (PML) mesh clustering. The results are given using a CPU and respectively a GPU based implementation*

Figure 6 shows the CVD energy (Eq. 4) behavior for different number of clusters between $2k$ to $1k$ for different algorithms. Observe that using the boundary-based approach slightly higher energy is obtained comparatively to the approximated CVD algorithm [VC04]. At the same time the energy for GPU ML clustering is similar to the energy of the approximated CVD algorithm. Similar behavior was observed for most of the tested models, although for some models and specific number of clusters, e.g. a cube model with 8 clusters, the results are identical.

Figure 6 also exemplifies a variation of the parallel ML and ML algorithm. Here the PML approach was applied up to 1633 cluster, followed by the ML clustering. Observe that the clustering energy from 1200 clusters is identical to that of the ML clustering. Thus for a specific region of interest this variation of the algorithm can be used to perform a fast ML construction.

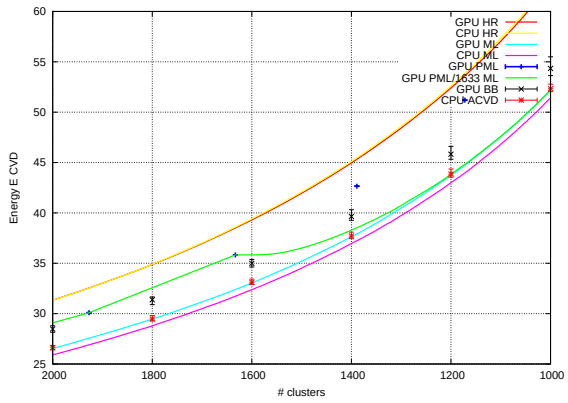


Figure 6: *CVD energy versus number of clusters for Bunny model. (HR) Hierarchical, (ML) Multilevel, (PML) Parallel ML, (BB) Boundary-based, (ACVD) Approximated CVD mesh clustering [VC04]. (PML/1633 ML) Performing PML up to 1633 clusters, then applying ML clustering.*

8. Conclusion and Future Work

We propose a new generic mesh clustering framework, which obeys a parallel clustering concept suitable for GPU-based mesh processing. Using this framework we show that considerable speedup can be obtained. The proposed parallel multilevel mesh clustering approach is very flexible. It has no limitation on the way in which parallel ML clustering can be performed.

As an example we showed how an approximated CVD can be obtained. In future we plan to implement more from existing energy functionals, e.g. to fit planes, spheres and cylinders, as done in [CSAD04], [WK05]. We also think that there are ways for redefining this approach for general data clustering propose.

References

[AFS06] ATTENE M., FALCIDIENO B., SPAGNUOLO M.: Hierarchical mesh segmentation based on fitting primitives. *Vis. Comput. 22*, 3 (2006), 181–193.

[CK08] CHIOSA I., KOLB A.: Variational multilevel mesh clustering. In *Proc. IEEE Int. Conf. on Shape Modeling and Applications (SMI)* (2008), pp. 197–204.

[CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. In *Proc. SIGGRAPH* (2004), ACM Press, pp. 905–914.

[DT07] DECORO C., TATARCHUK N.: Real-time mesh simplification using the GPU. In *I3D '07: Proc. of the 2007 Symp. on Interactive 3D graphics and games* (2007), ACM, pp. 161–166.

[GWH01] GARLAND M., WILLMOTT A., HECKBERT P. S.: Hierarchical face clustering on polygonal surfaces. In *Proc. Symp. on Interactive 3D graphics (I3D)* (2001), ACM Press, pp. 49–58.

[HH04] HALL J. D., HART J. C.: GPU acceleration of iterative clustering. In *Manuscript accompanying poster at GP^2. The ACM Workshop on General Purpose Comp. on Graph. Processors, and SIGGRAPH 2004 poster* (2004).

[JKS05] JULIUS D., KRAEVOY V., SHEFFER A.: D-charts: Quasi-developable mesh segmentation. *Proc. EUROGRAPHICS 24*, 3 (2005), 581–590.

[Män88] MÄNTYLÄ M.: *An Introduction to Solid Modeling*. Computer Science Press, 1988.

[MKMS07] MANTIUK R., KRAWCZYK G., MANTIUK R., SEIDEL H.-P.: High dynamic range imaging pipeline: Perception-motivated representation of visual content. In *Human Vision and Electronic Imaging XII* (2007), vol. 6492, SPIE, p. 649212.

[Ols95] OLSON C. F.: Parallel algorithms for hierarchical clustering. *Parallel Comput. 21*, 8 (1995), 1313–1325.

[SJP05] SHIUE L.-J., JONES I., PETERS J.: A realtime GPU subdivision kernel. In *ACM SIGGRAPH* (2005), ACM, pp. 1010–1015.

[VC04] VALETTE S., CHASSERY J.-M.: Approximated centroidal voronoi diagram for uniform polygonal mesh coarsening. *EUROGRAPHICS 23*, 3 (2004), 381–389.

[WK05] WU J., KOBBELT L.: Structure recovery via hybrid variational surface approximation. In *Proc. Eurographics* (2005), vol. 24, pp. 277–284.

[WLR88] WILLEBEEK-LEMAIR M., REEVES A. P.: Region growing on a hypercube multiprocessor. In *Proc. of the third conf. on Hypercube concurrent computers and applications* (1988), ACM, pp. 1033–1042.

[YLW06] YAN D.-M., LIU Y., WANG W.: Quadric surface extraction by variational shape approximation. In *Geometric Modeling and Processing - GMP (Lecture Notes in Computer Science)* (2006), vol. 4077/2006, pp. 73–86.