Accurate manycore-accelerated manifold surface remesh kernels

Hoby Rakotoarivelo and Franck Ledoux

Abstract In this work, we devise surface remesh kernels suitable for massively multithreaded machines. They fulfill the locality constraints induced by these hardware, while preserving accuracy and effectiveness. To achieve that, our kernels rely on:

- a point projection based on geodesic computations,
- a mixed diffusion-optimization smoothing kernel,
- an optimal direction-preserving transport of metric tensors,
- a fine-grained parallelization dedicated to manycore architectures.

The validity of metric transport is proven. The impact of point projection as well as the accuracy of smoothing kernel are assessed by comparisons with efficient existing schemes, in terms of surface deformation and mesh quality. Kernels compliance are shown by representative examples involving surface approximation or numerical solution field guided adaptations. Finally, their scaling are highlighted by conclusive profiles on recent dual-socket multicore and dual-memory manycore machines.

1 Context, issues and features overview

In this work, we focus on surface mesh adaptation kernels suitable for massively multithreaded architectures without sacrificing effectiveness or accuracy, in context of adaptive simulation involving both a numerical solver and a 3D remesher. Given an initial uniform triangular mesh and a budget of points, we aim at providing a mesh which minimizes the surface approximation error or the numerical solution interpolation error with respect to the given budget of points (as depicted in Fig. 1), while achieving good cell aspect ratios in both isotropic and anisotropic context.

Hoby Rakotoarivelo
CMLA, ENS Cachan, France. e-mail: hoby.rakotoarivelo@ens-cachan.fr

Franck Ledoux
CEA, DAM, DIF, France. e-mail: franck.ledoux@cea.fr

RELATED WORKS. Many robust open-source libraries for surface-volume remeshing exists such as tetgen, CGAL, MMG3D, MMGS, GMSH among others. The issue is that the involved kernels do not expose sufficient locality required by these architectures. Indeed, each operation on a mesh point or cell should involve a small, static and bounded vicinity. This locality constraint is far from being trivial, and most of the state-of-the-art kernels rely a dynamic point or cell vicinity. For instance,

- cavity-based remeshing kernels (surfacic Delaunay or hybrid cavity) enable to significantly improve mesh quality [1, 2]. They aim at removing bad cells when adding or removing points. However those cells are not statically known.
- atlas-based remeshing kernels provide well-sampled surface meshes, with a quality comparable to a variational scheme [3]. It consist of on-the-fly embedding the surface into a plane, and then to remesh the embedded region using 2D kernels. The issue is that it often require growing a non-predefined region at each time [4].
- numerical schemes are more stable on quasi-structured meshes. Relaxing mesh point degrees is tedious though, due to numerous local minima. To address this issue, related heuristics (such as 5-6-7 or puzzle solving) rely on a dynamic edge flip-refine-collapse sequence [5, 4]. Therefore, impacted cells cannot be inferred.

In fact, kernel locality and effectiveness are two antagonistic constraints. To achieve locality, one have to resort to basic kernels (no dynamic cavity, no local embedding, no dynamic sequence of operations). To achieve effectiveness though, one have to resort to dynamic kernels to quickly achieve mesh convergence, in terms of surface approximation, mesh quality, or solution interpolation of a finite element simulation.

CONTRIBUTIONS. In this work, we aim to conciliate both constraints. To achieve maximal locality, we resort to basic kernels depicted in Fig. 2, and we avoid any of the dynamic features described above. To achieve effectiveness, we rely on advanced geometric features instead. They include :

- POINT PROJECTION (for refinement, simplification and smoothing). In our case, the real surface is only known on mesh points, and we do not have an atlas for local parametrization. Thus we have to find a way to accurately put points on the real surface after cutting an edge, merging two points or moving a point. If the point lies on the mesh then it is not a problem since we may use a BEZIER curve, a PN-triangle or a local quadric surface in that case. However, if it does not lie on the mesh, then the situation is not clear. In that case, one have to project it on the mesh before projecting it to the surface, with a potential loss of accuracy. Here, we provide a point projection scheme which relies on geodesic curve computations using a differential geometry operator (sec. 2). It ensures that the loss of accuracy remains small, and contributes to speedup mesh convergence in terms of surface approximation.
- POINT RELOCATION (for smoothing). To reduce their number of rounds, refinement, simplification and swapping kernels are performed without taking care of cell qualities. Hence, improving mesh quality is entirely entrusted to smoothing

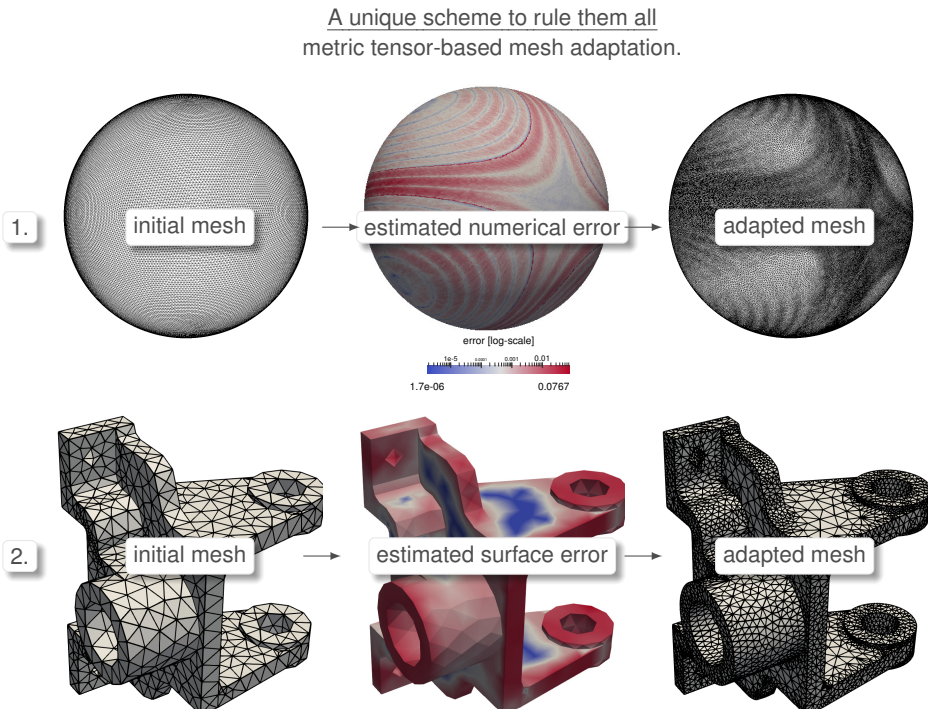


Fig. 1: Kinds of adaptation supported. If the mesh is intended to be adapted to the error of a numerical solution u (defined on each point), then the metric field may be derived from local Hessian matrices of u. If it is intended to be adapted to the error of the surface itself, then the metric field may be derived from local curvature tensors. Finally, both errors may be simultaneously supported using metric field intersection.

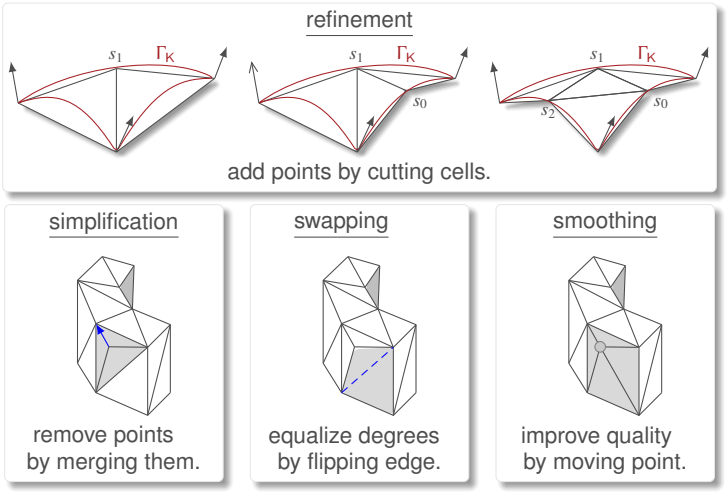


Fig. 2: Our static remesh kernels. Here, refinement and simplification aim at resampling the surface, whereas swapping and smoothing aim at regularizing it. They actually involve a cell, a couple of cells or the direct vicinity of a point.

kernel, which aims at moving points to improve surrounding cells quality. However, relocating a point involves an unavoidable surface deformation. Here, we provide a point relocation scheme for smoothing kernel, which aims at improving mesh quality while reducing the surface degradation. It relies on both a diffusion filter which is applied first, and a non-linear optimization routine invoked on failure cases (sec. 3). It contributes to speedup mesh quality convergence.

- METRIC TRANSPORT (for smoothing and refinement). In our case, desired edge size are prescribed on the mesh using metric tensors. It enables us to adapt the mesh to the error of a numerical solution as well as the error of the surface itself in anisotropic context (see Fig. 1). For any point, its metric tensor encodes the edge length at any direction incident to it. Hence, when a point p is created or relocated, then its metric tensor has to be interpolated from its neighbors. However, repeated metric interpolation would involve a loss of anisotropy. To reduce that, one may move all involved metric tensors at p before averaging them (if necessary). However, moving a metric tensor on a surface may deviate its directions if no precautions are taken. Here, we provide a safe way to move them on the surface without altering their principal directions (sec. 4). It is based on the notion of parallel transport in differential geometry, and contributes to speedup mesh convergence in terms of surface approximation and numerical solution interpolation.

Hence, the locality constraint induced by the hardware is respected. Indeed, our kernels involve a small and static vicinity for surface resampling, cells regularization or degree equalization (Fig. 2). Besides, their effectiveness is preserved thanks to the three geometric features introduced above. The remaining question is about how to efficiently port them on our target hardware:

- FINE-GRAINED PARALLELIZATION (for all kernels). Manycore machines consist of numerous underclocked cores with a very limited cache-memory per core. Besides, multicore machines have less but faster cores. The latter have multiple cache and memory levels, leading to unequal data access latencies (NUMA). Hence, kernels must expose a huge amount of parallelism and a high rate of data reuse to scale on both machines. However, remesh kernels are not trivially parallelizable in high performance computing context. Indeed they are data-driven since task dependencies evolve at runtime, and cannot be statically predicted. They are also data-intensive since most of their instructions are data accesses but often on different data. Hence, usual optimizations (such as cache tiling, static load balancing, prefetching) will not work well on them. Here, we devise a multithreaded scheme enabling to ease both data-driven and data-intensive issues (sec. 5). It is based on our work in [6, 7], but extended in 3D with ridges support.

2 Accurate point projection based on the exponential map

As stated before, we need to find a way to accurately put points on the real surface during refinement, simplification or smoothing. Hence, if the point lies on the mesh,

then it may be directly projected to this surface through a local parametrization scheme. The main issue is on finding an effective way to project the point when it does not lie on mesh. First, we show how we locally recover the real surface, and then we will describe how to accurately project the point on this real surface.

RECONSTRUCTION. In fact, the real surface Γ is only known on mesh points and needs to be locally recovered. It may be done by many ways, however achieving both accuracy and regularity is quite difficult. For instance, one may recover it in the vicinity of each point using a quadric surface obtained by a least squares method [ref]. Since Γ is approximated by a smooth surface, then it achieves regularity. However, it only provides a quadratic reconstruction of Γ. Besides, remeshers and shaders such as Inria's MMGS or the DIRECT-X 11 tesselation engine often resort to a cubic reconstruction through PN-triangles [8]. Despite their compacity, they do not ensure point tangent plane unicity accross patch boundaries. Here, we aim at conciliating both constraints. For that, we resort to a per-cell local parametrization with complete G1-continuity. It is achieved through a GREGORY PATCH construction with twist points BLENDING like in [9] but with special care for ridges and corners.

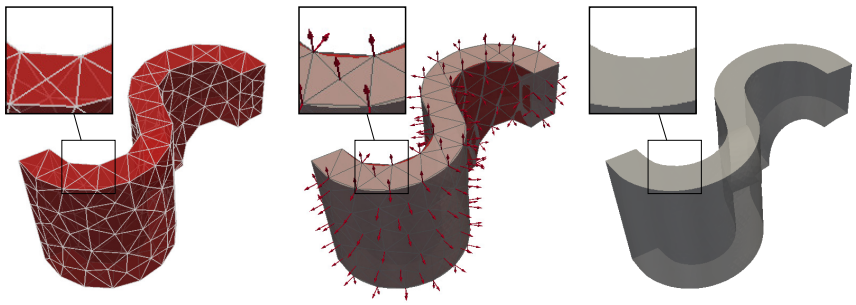
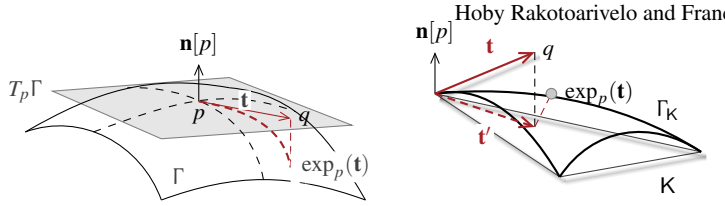


Fig. 3: Surface reconstruction using quartic patches with G^1-continuity.

POINT PROJECTION. In fine, the projection of a point p on the ideal surface may be directly obtained as long as its tangent plane $T_p\Gamma$ is supported by a mesh cell. However, there are situations where p does not lie on the mesh, and cannot be directly parametrized. During smoothing for instance, the projected point q has to be computed such that the length of underlying geodesic curve of $[pq]$ directed by a displacement vector $\alpha\mathbf{t}$ must be equal to $\|\alpha\mathbf{t}\|$. Such an operator exists in continuous context, it is the *exponential map* described in Def. 1.
Intuitively, $\exp_p(\mathbf{t})$ may be seen as an operator giving the optimal projection of q for a specified tangent vector $\mathbf{t} = \overrightarrow{pq}$ as shown in Fig. 4. It may be obtained through a reparametrization of γ by arc length. For that, we have to compute curve length $\mathrm{s}(t) = \int_0^t \|\frac{\mathrm{d}\gamma}{\mathrm{d}t}(x)\|\,\mathrm{d}x$ with $\frac{\mathrm{d}\gamma}{\mathrm{d}t}(0) = \mathbf{t}$, then express $\mathbf{t}$ and thus γ in terms of s. However, we do not have a parametrization of $\frac{\mathrm{d}\gamma}{\mathrm{d}t}$ in terms of t, and thus no explicit expression of $\exp_p$. Therefore, we use the following heuristic to approximate $\exp_p : T_p\Gamma \to \Gamma$.



(a) view in the vicinity of p (b) view on cell pointed by $\mathbf{t}$.

Fig. 4: Discrete approximation of exponential map in the vicinity of a point.

Definition 1 (EXPONENTIAL MAP). *Let p be a point of a manifold Γ, and R a region of its tangent plane. Its exponential map $\exp_p : \mathsf{R} \to \Gamma$ maps any tangent vector $\mathbf{t}$ of R to the geodesic curve segment γ starting from p with initial speed $\frac{d\gamma}{ds}(0) = \mathbf{t}$ and length $\ell(\gamma) = \|\mathbf{t}\|$.*

ALGORITHM. The first step is to find which cell K incident to p is pointed by $\mathbf{t}$. Indeed, $\exp_p(\mathbf{t})$ will lie in the quartic patch Γ_K related to K. Then the difficulty lies in the choice of a point $\tilde{q} \in \mathsf{K}$ such that the length of underlying curve equals to $\|\mathbf{t}\|$. In other words, we have to find a vector $\mathbf{t}' = [p\tilde{q}]$ from $\mathbf{t}$ such that $\exp_p(\mathbf{t}) = \Gamma_\mathsf{K}[\tilde{q}]$. Here, this issue is resolved by a single step linear search. To find an initial value of $\tilde{q}$, the idea is to perform a rotation of $\mathbf{t}$ on the tangent plane $T_\mathsf{K}\Gamma$ of K as shown in Fig. 4b. Then we compute the projection $\Gamma_\mathsf{K}[\tilde{q}]$ of $\tilde{q}$ on Γ_K, as well as its normal vector. Afterward, the idea is to approximate the geodesic segment γ by a B-spline curve spanned by $[p, \Gamma_\mathsf{K}[\tilde{q}]]$ and to compute its length. It enables us to adjust the norm of $\mathbf{t}'$ by a factor s such that $\|\mathbf{t}'\| = \|\mathbf{t}\|^2 \ell[\gamma]^{-1}$. By recomputing $\Gamma_\mathsf{K}[\tilde{q}]$ and γ, we actually have $\ell(\gamma) \approx \|\mathbf{t}\|$.

3 Accurate mixed diffusion-descent smoothing kernel

So far, we have an accurate way to project any point q lying either on the mesh or on the tangent plane of another point p, onto the real surface. Hence, we are able to project the resulting point on the real surface after an cutting an edge, merging two points or relocating a point. Here, we aim to specifically show how it may be used to devise an effective smoothing kernel. In our case, it is the only kernel entrusted to improve mesh quality, and thus is really important. For each point, it aims at improving surrounding cells qualities with respect to the metric tensor field.

ISSUES. Existing kernels may be classified in two categories: laplacian-based and non-linear optimization ones. The former is simple and enables both geometry denoising [10] as well as cell shape regularization. However, it tends to shrink the surface since it is not a low-pass filter [11], and does not strictly improve cell quality since it is a heuristic. The latter enables advanced features such as volume-preserving denoising [12, 13] or cell quality improvement on problematic cases [14]. However, it is clearly more expensive. Based on these facts, we devised a mixed ker-

nel like [15, 16]: a diffusion filter is primarily used for point relocation, followed by a cell quality maximization in failure cases. However, we use a different scheme for diffusion as well as for optimal step computation.

- DIFFUSION. For each point, we aim at equalizing its incident cell patches, while reducing the unavoidable surface deformation. Here the idea is to move the point p toward the projected weighted center of mass of its vicinity, as shown in (1). First, we compute the displacement direction $\mathbf{t}$ toward the center of mass of the region $\mathsf{C} = \bigcup_{j=1}^{n} \Gamma_j$. Then, we compute the geodesic segment curve γ starting from p with initial speed $\mathbf{t}$. Finally, the new point position is given by $\gamma(1)$.

$$p = \exp_p \left[\alpha \frac{\int_{\mathsf{c}} \log_{p^{[t]}}[x]\, \rho[x]\, \mathrm{d}x}{\int_{\mathsf{c}} \rho[x]\, \mathrm{d}x} \right], \text{ with } \begin{cases} \mathsf{C} : \text{continuous vicinity of } p, \\ \rho : \text{density function}, \\ \alpha : \text{scaling factor}. \end{cases} \tag{1}$$

  Here, $\rho[x] = \sqrt{\det[J_x^{\mathsf{T}} g_x J_x]}$ aims to weight the displacement of p with respect to the metric tensor g_x of each surrounding point x, and J_x is the JACOBIAN matrix of the surface parametrization at point x. To avoid useless computations, we exclude the case in which the center of mass is too far from p. Here, the displacement is accepted if the quality of the worst cell is enhanced and if the deviation of incident cells from $T_p\Gamma$ does not exceed an angular threshold θ_{max}. Otherwise, we reduce the scaling factor α, which is initially set to 1, in a dichotomic way.

- OPTIMIZATION. Here, the goal is to enforce worst cell quality improvement in the vicinity of p. Let $f_i[p]$ be the distorsion[1] of an incident cell K_i according to the current position of p. The goal is to solve $\min \max_i f_i[p]$ with the constraint that p must lie in C. For that, the idea is to move p gradually on the surface according to a displacement step α toward a direction $-\nabla f_k[p]$ until convergence or if a round threshold is achieved. The position of p is then updated as follow:

$$p^{[t+1]} = \exp_{p^{[t]}}[-\alpha \nabla f_k(p^{[t]})],\ \alpha \in [0,1], \text{ with } \begin{cases} f_j : \text{distorsion of cell } \mathsf{K}_j \\ f_k = \max_j f_j[p^{[t]}] \\ \alpha : \text{displacement step} \end{cases} \tag{2}$$

  Here, minimizing $f_k[p]$ may increase the distorsion of other cells. Hence, we must take them into account in the choice of initial step α. Let $j = \arg\min_{i \neq k} f_i[p]$. By solving $f_j[p - \alpha \nabla f_k(p)] = f_k[p - \alpha \nabla f_k(p)]$, and by considering the first order Taylor-Young expansion of f_k, we have:

$$\alpha = \frac{f_j[p] - f_k[p]}{\|\nabla f_k[p]\| - \langle \nabla f_k[p], \nabla f_j[p] \rangle}. \tag{3}$$

  The next step is determined by a linear search verifying WOLFE conditions [17]. They guarantee that f_k decreases significantly and that α is large enough to converge rapidly. Note that the convergence rate is closely related to the initial point

[1] The distorsion of a cell is just the inverse of its quality.

position (or seed). Since this routine is called after moving the point toward its center of mass, then p is relatively close to its optimal position. Thus, a few number of rounds is expected in practice (3 to 5). Finally, if the point lies on a ridge, then we relocate it on the midpoint of the curve segment related to its neighboring ridges.

4 Accurate direction-preserving transport of metrics

Up to now, we have an effective kernel designed to improve mesh quality by moving points. Note that when a point is relocated, its geometric data (normal and metric tensor) need to be recomputed or interpolated from its neighbors, since they have changed. However, repeated metric tensor interpolation would involve a loss of anisotropy due to diffusion effects. It is at least the case when a simple linear scheme is used. Besides, moving a metric tensor along a curve may deviate its directions. To ease this deviation, we resort to a parallel transport scheme, as described in Def. 2, but extended to metric tensors. Intuitively, it generalizes the notion of translation on manifold surfaces. Note that it is already used in MMGS in a heuristically way. Here, our goal is to formally prove that a direction-preserving transport of metric tensors may be simply achieved through tangent vectors parallel transport.

Definition 2 (PARALLEL TRANSPORT). *Let ∇ be an affine connection[a] related to a manifold Γ. A vector field $\mathbf{v}^{[t]}$ on the tangent bundle of Γ along a curve $\gamma : I \to \Gamma$ is said parallel with respect to this connection, if $\nabla_{\dot{\gamma}[t]}^{\bullet} \mathbf{v}^{[t]}$ vanishes for all $t \in I$. Hence, a tangent vector $\mathbf{v}$ is parallel transported from p to q on γ if the vector field $\mathbf{v}^{[t]}$ induced by its displacement is parallel.*

[a] An affine connection generalizes the notion of derivative for vector fields on a manifold.

Definition 3 (LEVI-CIVITA CONNECTION). *There exists a unique affine connection ∇ on a Riemannian manifold (Γ, g) such that:*

- *it is torsion-free: there is no tangent planes rotation along a geodesic γ when they are parallel transported along γ through ∇.*
- *it is compatible with g: the induced dot product at any point p of Γ is preserved by parallel transport of p along γ: this displacement is actually an isometry.*

For each curve $\gamma : I \to \Gamma$, a metric $g_{\gamma[t]}$ is compatible with the Levi-Civita connection ∇ of the surface (see Def. 3), if for any $t \in I$ and for each vector fields $\mathbf{u}, \mathbf{v}$ of the tangent bundle of Γ, we have:

$$(\nabla_{\dot{\gamma}[t]}^{\bullet} g_{\gamma[t]})(\mathbf{u},\mathbf{v}) = \partial_{\dot{\gamma}[t]}^{\bullet}(g_{\gamma[t]}(\mathbf{u},\mathbf{v})) - g_{\gamma[t]}(\nabla_{\dot{\gamma}[t]}^{\bullet}\mathbf{u},\mathbf{v}) - g_{\gamma[t]}(\mathbf{u},\nabla_{\dot{\gamma}[t]}^{\bullet}\mathbf{v}) = 0. \quad (4)$$

In fact, the metric tensor g_p related to a point p is a symmetric bilinear form corresponding to a symmetric matrix M in a local basis of its tangent plane $T_p\Gamma$. In particular, there exists a local basis P of $T_p\Gamma$ such that M is congruent to a diago-

Algorithm 1: Parallel transport of a metric tensor

let $\tilde{p} = p$.
for each timestep $t \in I$ **do**
  set $\tilde{q} = \gamma[t]$, and extract Jacobians and normals at $\tilde{p}$ and $\tilde{q}$.
  extract a local basis $\mathsf{P} = (\mathbf{v}_1, \mathbf{v}_2)$ of $T_{\tilde{p}}\Gamma$ such that $g_{\tilde{p}} = J_{\tilde{p}}\,\mathsf{P}^{\mathsf{T}}\,\mathsf{D}\,\mathsf{P}\,J_{\tilde{p}}^{\mathsf{T}}$.
  determine the point s as follow: ▷ SCHILD'S LADDER.
  - let r be the endpoint of $\mathbf{v}_1$,
  - compute a segment $[r\tilde{q}]$ and let m be its midpoint,
  - compute a segment $[ps]$ such that $\|ps\| = 2\|pm\|$,

  compute $\mathbf{v}_1^{[t]} = \overrightarrow{\tilde{q}s}$, $\mathbf{v}_2^{[t]} = \mathbf{v}_1^{[t]} \times \mathbf{n}(\tilde{q})$, and $\mathsf{P}^{[t]} = (\mathbf{v}_1^{[t]}, \mathbf{v}_2^{[t]})$.
  store $g_{\tilde{p}}^{[t]} = J_{\tilde{q}}\,\mathsf{P}^{[t],\mathsf{T}}\,\mathsf{D}\,\mathsf{P}^{[t]}\,J_{\tilde{q}}^{\mathsf{T}}$.
  update $\tilde{p} = \tilde{q}$ and $\mathbf{n}(\tilde{p}) = \mathbf{n}(\tilde{q})$ and return $g_p^{[1]}$.
end

nal matrix D in P. In other words, $g_p(\mathbf{v}_i, \mathbf{v}_j) = 0$ for any $\mathbf{v}_i, \mathbf{v}_j \in \mathsf{P}$. Here, we aim to show that the eigenvalues λ_i of D are invariant by parallel transport of column vectors $\mathbf{v}_i$ of P along a curve γ spanned by $[pq]$. For a given $t \in I$, let $\mathbf{w}^{[t]}$ be the transported tangent vector $\mathbf{w}$ at $\gamma(t)$, D the eigenvalue diagonal matrix at $p = \gamma(0)$, and $\mathsf{P}^{[t]} = (\mathbf{v}_1^{[t]}, \mathbf{v}_2^{[t]})$. Thus, we have:

$$\begin{aligned}
\forall \mathbf{u}^{[0]} \in T_{\gamma[0]}\Gamma : g_{\gamma[t]}(\mathbf{u}^{[t]}, \mathbf{v}_2^{[t]}) &= \langle \mathbf{u}^{[t]}, \mathsf{M}_{\gamma[t]}\mathbf{v}_2^{[t]} \rangle = \langle \mathbf{u}^{[t]}, \textstyle\sum_{i=1}^{2} \lambda_i^{[t]} \mathbf{v}_i^{[t]} \mathbf{v}_i^{[t],\mathsf{T}} \mathbf{v}_2^{[t]} \rangle, \\
&= \langle \mathbf{u}^{[t]}, \lambda_1^{[t]} \mathbf{v}_1^{[t]} (\mathbf{v}_i^{[t],\mathsf{T}} \mathbf{v}_2^{[t]}) + \lambda_2 (\mathbf{v}_2^{[t]} \mathbf{v}_i^{[t],\mathsf{T}}) \mathbf{v}_2^{[t]} \rangle, \\
&= \langle \mathbf{u}^{[t]}, \lambda_2^{[t]} \|\mathbf{v}_2^{[t]}\| \mathbf{v}_2^{[t]} \rangle = \lambda_2^{[t]} \langle \mathbf{u}^{[t]}, \mathbf{v}_2^{[t]} \rangle,
\end{aligned}$$

At the same time $\nabla^{\bullet}_{\dot{\gamma}[t]} \mathbf{u} = \nabla^{\bullet}_{\dot{\gamma}[t]} \mathbf{v}_i = \mathbf{0}$, $\forall t \in I$,

thus $g_{\gamma[t]}(\nabla^{\bullet}_{\dot{\gamma}[t]} \mathbf{u}, \mathbf{v}_i) + g_{\gamma[t]}(\mathbf{u}, \nabla^{\bullet}_{\dot{\gamma}[t]} \mathbf{v}_i) = 0$,

$$\text{hence } \nabla^{\bullet}_{\dot{\gamma}[t]} g_{\gamma[t]}(\mathbf{u}, \mathbf{v}_i) = \partial^{\bullet}_{\dot{\gamma}[t]} g_{\gamma[t]}(\mathbf{u}^{[t]}, \mathbf{v}_i^{[t]}),$$

$$\text{therefore } \nabla^{\bullet}_{\dot{\gamma}[t]} g_{\gamma[t]}(\mathbf{u}, \mathbf{v}_i) = 0 \Leftrightarrow \frac{g_{\gamma[t]}(\mathbf{u}^{[t]}, \mathbf{v}_i^{[t]}) - g_{\gamma[0]}(\mathbf{u}^{[0]}, \mathbf{v}_i^{[0]})}{\int_0^t \|\dot{\gamma}(s)\| \mathrm{d}s} = 0.$$

$$\begin{aligned}
\text{by the way } \langle \mathbf{u}, \mathbf{v}_i \rangle \text{ constant on } \gamma &\Leftrightarrow \frac{\langle \mathbf{u}^{[0]}, \mathbf{v}_i^{[0]} \rangle}{\int_0^t \|\dot{\gamma}(s)\| \mathrm{d}s} (\lambda_i^{[t]} - \lambda_i^{[0]}) = 0, \\
&\Leftrightarrow \lambda_i^{[t]} - \lambda_i^{[0]} = 0, \\
&\Leftrightarrow \mathsf{M}_{\gamma[t]} = \textstyle\sum_{i=1}^{2} \lambda_i^{[0]} \mathbf{v}_i^{[t]} \mathbf{v}_i^{[t],\mathsf{T}}, \\
&\Leftrightarrow \mathsf{M}_{\gamma[t]} = \mathsf{P}^{[t],\mathsf{T}} \mathsf{D}\, \mathsf{P}^{[t]},
\end{aligned}$$

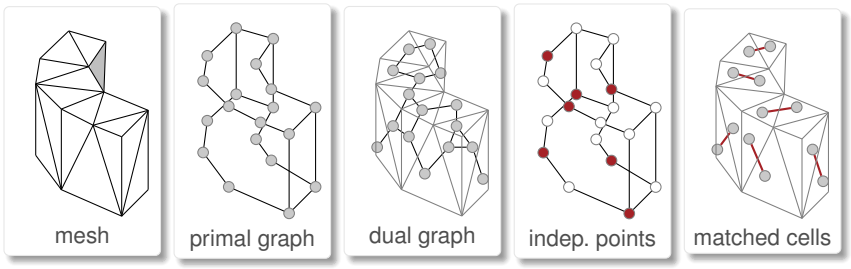
$$\text{therefore } \nabla^{\bullet}_{\dot{\gamma}[t]} g_{\gamma[t]}(\mathbf{u}, \mathbf{v}) = 0 \Leftrightarrow \begin{cases} \nabla^{\bullet}_{\dot{\gamma}[t]} \mathbf{u} = \nabla^{\bullet}_{\dot{\gamma}[t]} \mathbf{v} = \mathbf{0} \\ g_{\gamma[t]}(\mathbf{u}, \mathbf{v}) = \langle \mathbf{u}, \mathsf{P}^{[t],\mathsf{T}} \mathsf{D}\, \mathsf{P}^{[t]} \mathbf{v} \rangle \end{cases}. \tag{5}$$

According to (5), a parallel transport of a metric tensor g_p along a curve γ is equivalent to a parallel transport of its eigenvectors $\mathbf{v}_i$ through the Levi-Civita connection ∇

related to the intrinsic metric of the surface (also called the first fondamental form). In fact, since the directions of the metric tensor must be orthogonal at each point of $\gamma[t]$, then it is enough to transport a unique direction $\mathbf{v}_1$ and to find $\mathbf{v}_2$ such that $\langle \mathbf{v}_1^{[t]}, \mathbf{v}_2^{[t]} \rangle = 0$ for any $t \in I$. Here, it is achieved through the routine depicted in Alg. 1.

5 Fine-grained lock-free parallelization

At this point, we have four effective kernels for surface resampling, mesh regularization or degree equalization. They respect the locality constraint induced by our target hardware, since they only involve a small static vicinity at each time, (Fig. 2). Besides, they do not rely on a dynamic sequence of operations to achieve effectiveness. Instead, they rely on the three geometric features described in sec. 2, 3, and 4. Hence, the remaining concern is about how to parallelize them efficiently, despite the fact that they are data-driven and data-intensive as explained in sec. 1.



kernel	subgraph	heuristic
gradation	primal	coloring
refinement	none	none
contraction	primal	indep.
swapping	dual	matching
smoothing	primal	coloring

Fig. 5: Task graphs and related heuristics.

As stated in sec.1, a huge amount of parallelism is required for kernels scaling on these either low frequency and limited memory per core, or deep memory hardware (multicore, manycore). For that, we clearly advocate the use of massively multithreaded schemes instead of usual MPI-based ones. Here, we devise a fine-grained lock-free scheme based on our work in [6, 7], but with additional ridges adjacency graph support. For each kernel, data dependencies are formulated into a graph, and tasks are extracted through multithreaded maximal stable set, graph coloring and matching heuristics as shown in Fig. 5, except for refinement. Incidence graphs updates are done through a dual-step scheme: related incidence lists are grown asyn-

chronously using cheap atomic primitives (15-30 CPU cycles), then obsolete references are removed in a single sweep.

To ease their irregularity, we structure kernels into bulk-synchronous tasks [18]. Hence, each kernel is organized into sweeps, and each sweep consists of local computation, shared-data writes and a thread barrier.To ease data indirections penalities, we use a locality-aware reduction[2] scheme. It enables coalescent data writes in shared memory, and is further explained in [7].

6 Numerical evaluation

In short, we have proposed four remesh kernels for surface resampling, mesh regularization and degree equalization, which conciliates both locality and effectiveness constraints. Indeed, they do not involve neither a dynamic cell or point vicinity growing, nor a dynamic sequence of operations. Instead, they rely on three geometric features described in sec. 2, 3 and sec. 4. For sake of completeness, we also have proposed a fine-grained parallelization scheme to port these kernels on multicore and manycore architectures. At this point, we aim to numerically assess the effectiveness of each devised feature. For that purpose, the devised kernels are written in C++14 and OPENMP4 as a library called trigen. It is being integrated into a new version of GMDS [19], and will be open-source.

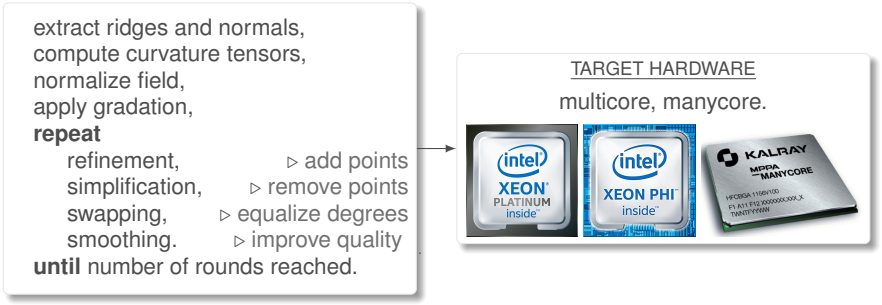


Fig. 6: Algorithm and kind of hardware involved in our numerical evaluations.

PROJECTION. To assess the effectiveness of our point projection scheme, we evaluate it on both mesh simplification and smoothing cases in terms of surface approximation error as depicted in Fig. 7. Indeed, a major issue on both kernels relies on reducing the unavoidable surface degradation while perfoming point relocation or removal. Here, we aim at assessing that our exponential-map based kernels are as accurate as those of the state-of-the-art.

[2] A reduction is an agregation of local data.

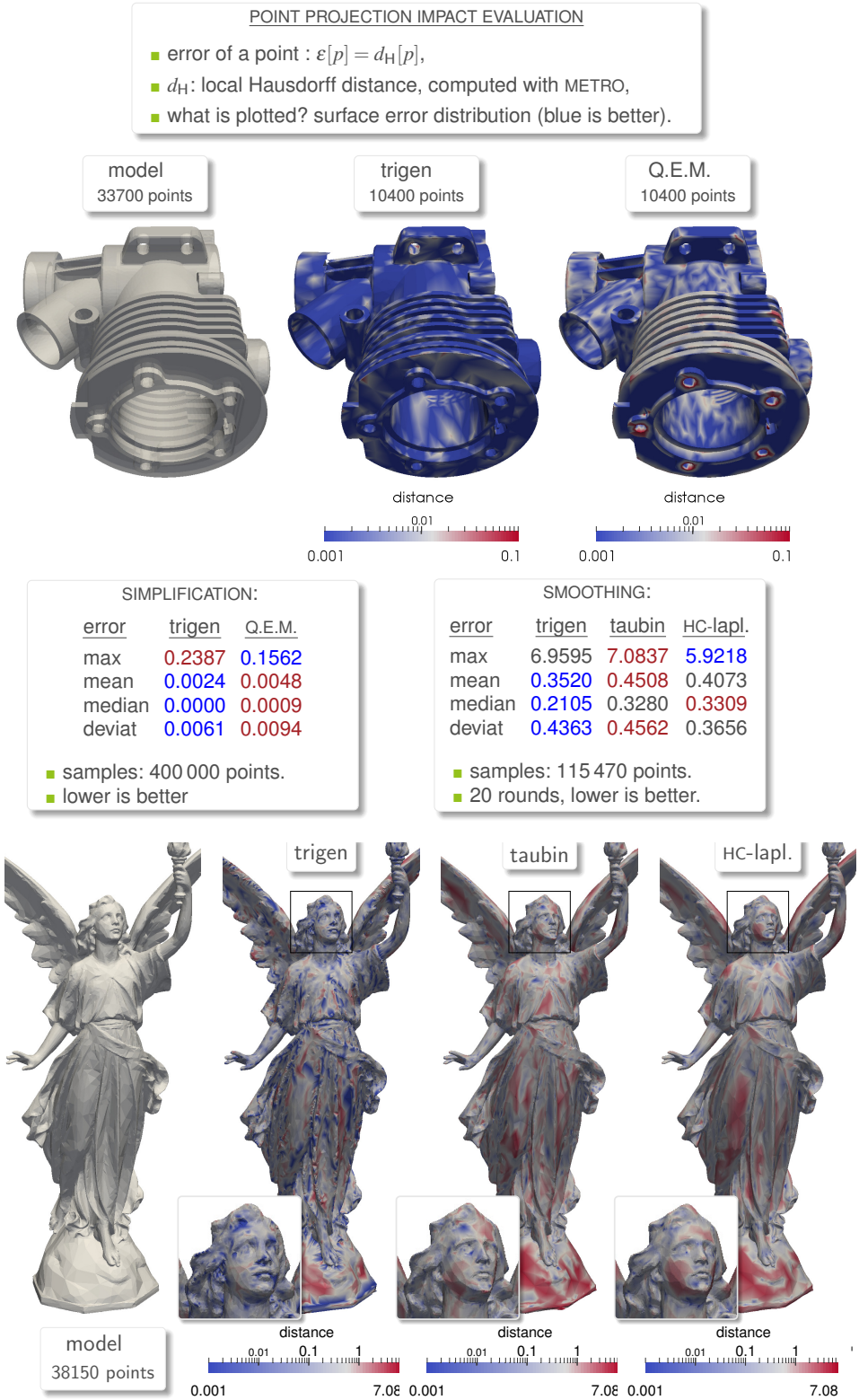


Fig. 7: Impact of the point projection scheme on kernels. Surface degradation is a major issue in mesh simplification and smoothing. Here, the goal is to assess the accuracy of our exponential-map based kernels compared to the state-of-the-art.

- First, the surface error induced by our simplification kernel compared to the quadric error metric [20] is given. Note that both algorithms are quite identical, the difference mainly rely on how do we put the remaining point after merging two points. Here, normals deviation threshold is set to $\theta_{\max} = 10^\circ$ and no edge swap is allowed. The pointwise Hausdorff distance is computed through METRO plugin [21] with roughly 400 000 sampled points. The given error distribution shows that it is better reduced and equidistributed on strong curvature areas and features curves with our scheme unlike quadric error metric.
- The surface error induced by our smoothing kernel compared to Taubin [22] and HC-laplacian [23] (using Meshlab) is given in case of a non trivial smooth manifold. Note that the two other schemes are optimized to reduce shrinkage effects induced by an usual laplacian, and contributes to reduce the surface deformation. The diameter is set to $h_{\max} = 10\%$ of bounding-box diagonal and normals deviation threshold is set to $\theta_{\max} = 7^\circ$. Here, we can see that the surface is well preserved with our kernel.

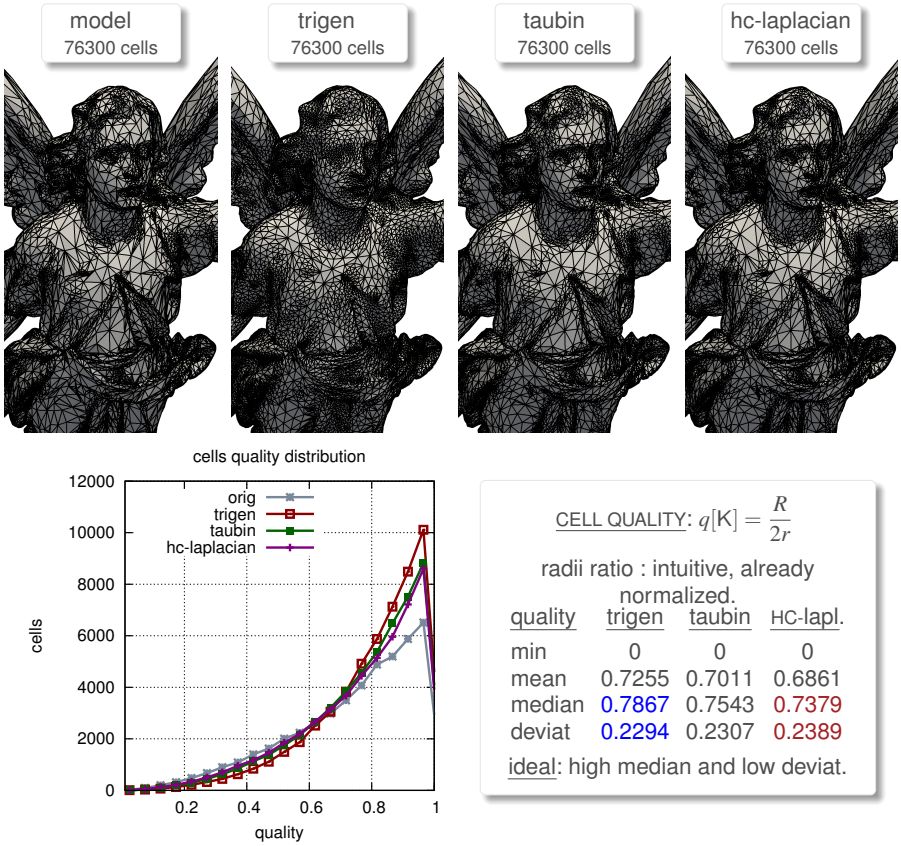


Fig. 8: Evaluation of our smoothing kernel. Here, we aim to assess if a good mesh quality may actually be achieved while minimizing the surface deformation.

SMOOTHING. Recall that in our case, smoothing aims primarily at improving cell quality. A comparison with Taubin and HC-laplacian in terms of mesh quality is given in Fig. 8. It is exactly the same testcase as in Fig. 7 with the same parameters. We have already assessed that the surface error is reduced with our smoothing kernel. Here, the distribution of cell aspect ratios shows that a good mesh quality is also achieved.

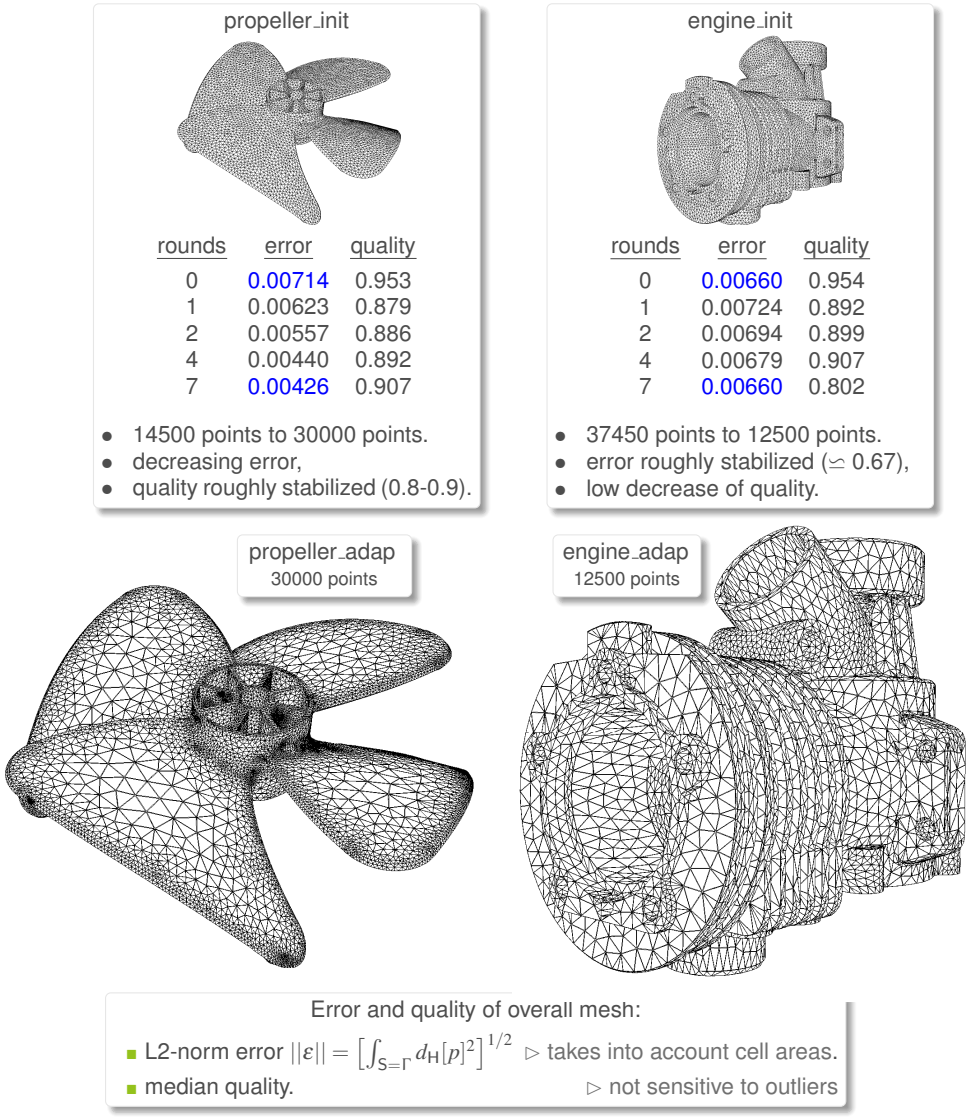


Fig. 9: Convergence in error and quality when all kernels are being combined.

CONVERGENCE. This time, we aim at assessing mesh error and quality convergence when all kernels are combined. Precisely, we aim to evaluate how both surface error and mesh quality evolve when the number of points is increased or decreased. For that, local curvature guided adaptations are first considered in Fig. 9 through isotropic model enrichment (PROPELLER, $n_{\max} = 2n$) and anisotropic coarsening (ENGINE, $n_{\max} = \frac{n}{3}$). Here, curvature tensors are computed using MEYER's scheme [24], and a simple L^{∞}-norm error estimate is used for metric field extraction. A gradation process based on [25] is used with a H-shock threshold set to 1.5. For PROPELLER, we can clearly see that points are mostly located on feature curves and high curvature areas, and that edge sizes are well-graded. For ENGINE, a surface degradation is necessarily introduced. However, feature curves are well-preserved, and cells are correctly stretched along minimal curvature direction. The surface approximation error as well as mesh quality evolutions are also depicted throughout remeshing rounds. For PROPELLER, mesh quality is relatively preserved whereas error decreases along rounds. For ENGINE, the surface approximation is relatively well-preserved despite a number of points reduced by a factor three.

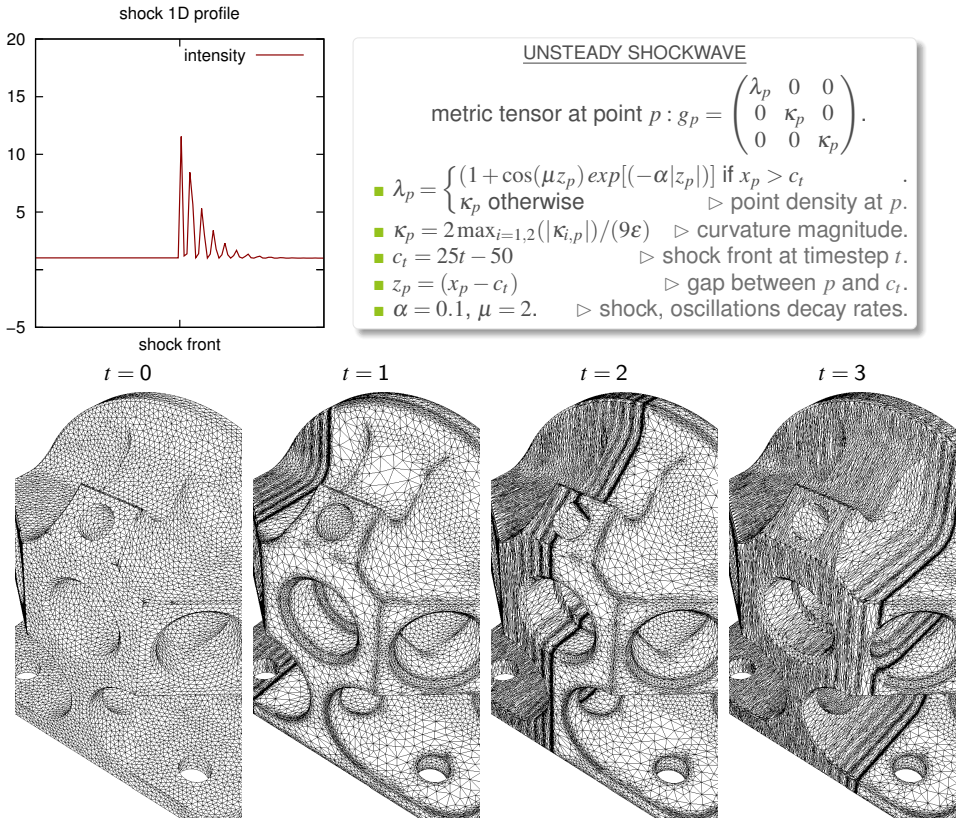


Fig. 10: Anisotropic adaptation to a user-defined size field using all kernels.

Finally, user-defined density field guided adaptation is shown in Fig. 10. Here, we can see that scales are finely captured (shock front and small flows), and a correct cell stretching is achieved.

PROFILING. For our benchmarks, we used three Intel-based machines: two NUMA dual-socket 32/48-core and a dual-memory 68-core processors (table 1). The latter involves 4 VPU per core as well as on-chip MCDRAM at 300 GB/S and usual DDR4 at 68 GB/S. Here, trigen's code has been compiled through Intel compiler (version 17) with O3 and qopt-prefetch=5 optimization flags, including auto-vectorization and software prefetching capabilities. To enable specific features, we set march=native flag while compiling on HSW-SKL, and xmic-avx512 on KNL. Thread-core binding is done in a round-robin way by setting KMP_AFFINITY to scatter on normal mode (1 per core on HSW-SKL), and to compact on hyperthreading enabled (our default mode on KNL with 4 HT per core as recommanded by Intel). Finally, two testcases were considered: (1) a curvature-based piecewise manifold isotropic adaptation of the engine model in [3] (engine) with 1 826 000 points and 3 652 058 cells, (2) the anisotropic hessian-based shock adaptation (shock) in Fig. 1 with 1 014 890 points, 2 029 772 cells and a target resolution $n = 250\,000$ for the latter.

Table 1: Benchmark machines features.

code	sockets	NUMA	cores	GHz	threads	GB/core	full reference
HSW	2	4	32	2.5	64	4.0	Xeon Haswell E5-2698 v3
SKL	2	2	48	2.7	96	7.9	Xeon Skylake Platinum 8168
KNL	1	4	72	1.4	288	1.5	Xeon-Phi Knights Landing 7250

SCALING. Strong scaling profile for four adaptive rounds on all architectures are given in Fig. 11, as well as kernel processing rate on KNL. A good scaling is achieved and a similar profile is observed on all architectures. However kernels are 3-4 times slower on KNL compared to HSW-SKL which is normal considering its per-core frequency and cache size limitations. The engine case is more CPU-expensive due to ridge-specific processing. A quasi-linear kernel processing rate scaling is achieved on KNL, except for refinement which requires no graph but involves more thread synchronization. Restitution time distribution per step on KNL is given in Fig. 12. Recall that in our context, each kernel is structured into bulk-synchronous sweeps. The time spent on each sweep is then represented here, and overheads related to parallelization are depicted in red (task extraction and synchronization involved in mesh topology updates). These overheads are negligible and scale roughly at the same rate as other steps. However, an exceptional overhead is observed on relaxation kernel at 64 cores (256 threads). Indeed, memory indirections involved by this low arithmetic-intensity kernel induces high cache contentions and RAM access penalties. This behavior is not observed on larger per-core cache/RAM machines (SKL,HSW). Note that in HPC, we clearly advocate comparison of algorithms instead of software tools. Indeed, involved schemes are not comparable if they are not implemented exactly in the same way (with the same level of code optimization). Thus no performance comparison is included here.

- KNL: low frequency, low per thread memory ▷ 1.5 GHz, 375 MB per thread
- KNL: high memory-access latencies ▷ 30 ns for MCDRAM, 28 ns for DDR4.
- KNL: enable hyperthreading to hide latency. ▷ 4 threads per core.
- HSW and SKL: no hyperthreading to save GB per thread. ▷ 4 to 7.9 GB per thread.

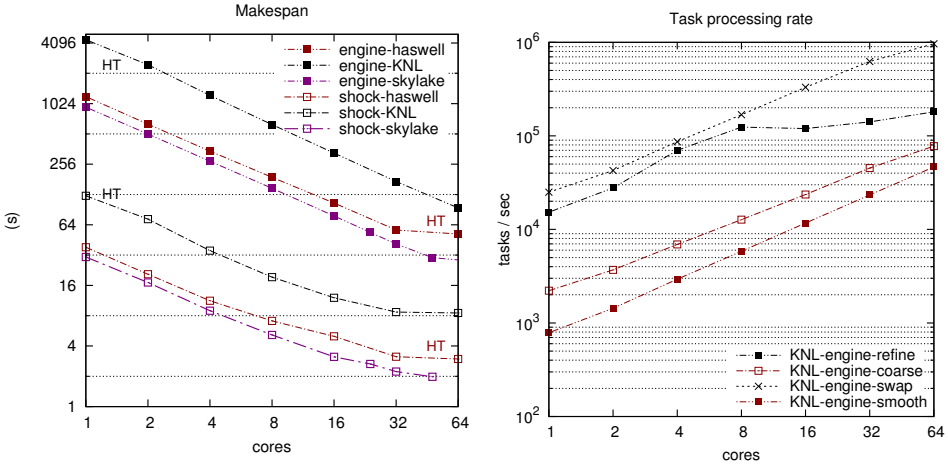


Fig. 11: Restitution time on all architectures and kernels processing rate on KNL. Note that a KNL core is more like a GPU core rather than a XEON one. It does not really make sense to compare single core performance of KNL with HSW or SKL.

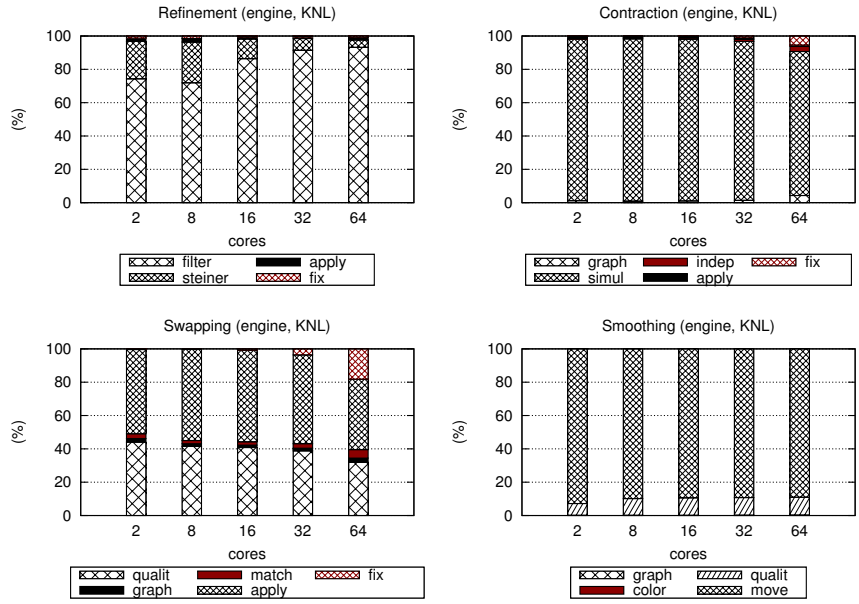


Fig. 12: Kernels time distribution per step on KNL; overheads are depicted in red.

References

1. Jean-Daniel Boissonnat, Kan-Le Shi, Jane Tournois, and Mariette Yvinec. Anisotropic delaunay meshes of surfaces. ACM ToG, 34(2):10, 2015.
2. Adrien Loseille and Victorien Menier. Serial and parallel mesh modification through a unique cavity-based primitive. In IMR-22, pages 541–558, 2014.
3. Bruno Lévy and Nicolas Bonneel. Variational anisotropic surface meshing with voronoi parallel linear enumeration. In IMR-21, pages 349–366, 2013.
4. Vitaly Surazhsky and Craig Gotsman. Explicit surface remeshing. In Eurographics, SGP'03, pages 20–30, 2003.
5. Vincent Vidal, Guillaume Lavoué, and Florent Dupont. Low budget and high fidelity relaxed 567-remeshing. Computer Graphics, 47:16–23, 2015.
6. Hoby Rakotoarivelo, Franck Ledoux, and Franck Pommereau. Fine-grained parallel scheme for anisotropic mesh adaptation. In IMR-25, pages 123–135, 2016.
7. Hoby Rakotoarivelo, Franck Ledoux, Franck Pommereau, and Nicolas Le-Goff. Scalable fine-grained metric-based remeshing algorithm for manycore-NUMA architectures. In EuroPar'23, pages 594–606, 2017.
8. Alex Vlachos, Jörg Peters, Chas Boyd, and Jason Mitchell. Curved PN triangles. In ACM I3D, pages 159–166, 2001.
9. Walton and Meek. A triangular G^1 patch from boundary curves. CAD, 28(2):113–123, 1996.
10. Gabriel Taubin. A signal processing approach to fair surface design. SIGGRAPH '95, pages 351–358, 1995.
11. Yutaka Ohtake, Alexander Belyaev, and Ilia Bogaevski. Polyhedral surface smoothing with simultaneous mesh regularization. In GMP, pages 229–237, 2000.
12. Andrew Kuprat, Ahmed Khamayseh, Denise George, and Levi Larkey. Volume conserving smoothing for piecewise linear curves, surfaces, and triple lines. JCP, 172(1):99–118, 2001.
13. Xiangmin Jiao. Volume and feature preservation in surface mesh optimization. In IMR-15, pages 359–373, 2006.
14. Daniel Aubram. Optimization-based smoothing algorithm for triangle meshes over arbitrarily shaped domains. Technical report, 2014.
15. Scott Canann, Joseph Tristano, and Matthew Staten. An approach to combined laplacian and optimization-based smoothing for triangular, quadrilateral, and quad-dominant meshes. In IMR-7, pages 479–494, 1998.
16. Lori Freitag. On combining laplacian and optimization-based mesh smoothing techniques. MeshTrends, 220:37–43, 1999.
17. Philip Wolfe. Convergence conditions for ascent methods. II: Some corrections. SIAM Review, 13(2):185–188, 1971.
18. Leslie Valiant. A bridging-model for multicore computing. JCSS, 77:154–166, 2011.
19. Franck Ledoux, Jean-Christophe Weill, and Yves Bertrand. Gmds: A generic mesh data structure. Technical report, IMR-17, 2008.
20. Michael Garland and Paul Heckbert. Surface simplification using quadric error metrics. In SIGGRAPH'97, pages 209–216, 1997.
21. Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. Metro: Measuring error on simplified surfaces. Technical report, CNRS, 1996.
22. Gabriel Taubin. Curve and surface smoothing without shrinkage. In ICCV '95, pages 852–857, 1995.
23. Joerg Vollmer, Robert Mencl, and Heinrich Mller. Improved laplacian smoothing of noisy surface meshes. Eurographics, pages 131–138, 1999.
24. Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In Visu. and Maths III, pages 35–57. Springer, 2003.
25. Frédéric Alauzet. Size gradation control of anisotropic meshes. JFEAD, 46(1):181–202, 2010.