cSculpt: A System for Collaborative Sculpting

Claudio Calabrese[1]* Gabriele Salvati[1]* Marco Tarini[2,3] Fabio Pellacini[1]

[1]Sapienza University of Rome [2]CNR-ISTI [3]Insubria University

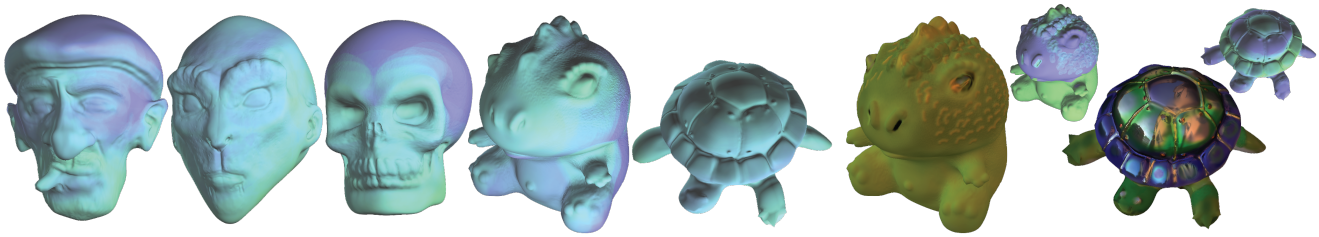

Figure 1: *Left: Models created by two artists sculpting collaboratively starting from a sphere. Right: Collaborative edits to geometry and materials. Colors on the left side and insets encode the total percentage of edits performed by each user: green for user A, blue for user B, and cyan for both.*

Abstract

Collaborative systems are well established solutions for sharing work among people. In computer graphics these workflows are still not well established, compared to what is done for text writing or software development. Usually artists work alone and share their final models by sending files. In this paper we present a system for collaborative 3D digital sculpting. In our prototype, multiple artists concurrently sculpt a polygonal mesh on their local machines by changing its vertex properties, such as positions and material BRDFs. Our system shares the artists' edits automatically and seamlessly merges these edits even when they happen on the same region of the surface. We propose a merge algorithm that is fast-enough for seamless collaboration, respects users' edits as much as possible, can support any sculpting operation, and works for both geometry and appearance modifications. Since in sculpting artists alternatively perform fine adjustments and large scale modifications, our algorithm is based on a multiresolution edit representation that handles concurrent overlapping edits at different scales. We tested our algorithm by modeling meshes collaboratively in different sculpting sessions and found that our algorithm outperforms prior works on collaborative mesh editing in all cases.

Keywords: collaborative modeling, digital sculpting and painting.

1 Introduction

Collaborative Digital Sculpting. Digital sculpting is a 3D modeling paradigm where free-form surfaces are manipulated with tools that mimic real-life sculpting of soft materials, e.g. clay. This paradigm is particularly effective when designing organic shapes, since artists do not have concerns about mesh connectivity and topology details, working always with high-resolution irregular meshes.

Cloud-based tools enable seamless, real-time collaboration between users when editing a variety of contents, from documents to code. In 3D modeling, seamless collaboration is still in its infancy. While some commercial tools are available for collaborative CAD using NURBS, e.g. [Onshape 2014], no commercial applications support collaborative sculpting of free-form polygonal meshes yet. This is the focus of our work.

Sculpting Properties. Collaborative sculpting is particularly challenging since high-resolution meshes need to be manipulated in real-time and since artists work in a very unstructured manner. For example, analyzing the workflows data from [Denning et al. 2015] it is clear that artists work alternatively at different scales, mixing large modifications to fine detail adjustments. For the large scale edits, soft brushes are frequently as large as half the model. Furthermore, sculpting is often combined with appearance painting to define material parameters.

cSculpt. We present a prototype system for digital sculpting and painting where artists work concurrently and seamlessly on the same model. In our system, each artist works on a local version of the model. Collaboration is established by automatically sharing and merging all edits at fixed time intervals that are fast enough for seamless collaboration. The core contribution of our work is a merge algorithm that can handle concurrent edits at different scales on the same mesh regions. We propose the use of a multiresolution hierarchy to encode edits at different spatial frequencies. At each level, edits from different artists are encoded as linear transformations of the underlying vertex properties and are combined together with weighted blending in an appropriate linear space. Edits at different levels are considered independent one from each other and no blending occurs. Our merge algorithm is efficient, respects artists intentions, and can handle both geometry and appearance parameters.

Comparison with *MeshHisto*. [Salvati et al. 2015] presents a system for collaborative modeling of low-polygonal and subdivision meshes, based on robustly detecting conflicts between artists and rejecting conflicted operations. If applied to sculpting, operations would be in conflict whenever two edits touch the same vertex. Since the use of large brushes is common, a conflict detection approach would not work in our case. For example, in the results of this paper,

*Joint first authors

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org. © 2016 ACM.
SIGGRAPH '16 Technical Paper,, July 24-28, 2016, Anaheim, CA,
ISBN: 978-1-4503-4279-7/16/07
DOI: http://dx.doi.org/10.1145/2897824.1111111

MeshHisto would have rejected between 11% to 37% of the total edits, effectively hindering collaboration entirely. This is not surprising since it is addressing an entirely different modeling workflow that is not comparable to sculpting. We take an entirely different approach and merge overlapping edits smoothly. Said another way, we handle conflicts gracefully without rejecting edits.

Results and Contributions. In order to test our approach we developed a collaborative sculpting system prototype. We support a variety of UI tools for mesh sculpting and painting, similar to the ones offered by Blender [Blender 2014]. Fig. 1 shows models created in collaborative sessions by two artists, where both geometry and materials are edited. The supplemental video shows how artists work seamlessly on these models. Our merge algorithm runs in fractions of a second for meshes with hundreds of thousands of faces. We tested the system both with fast synching times as well as slow ones to simulate unreliable network communications between artists. In both cases, collaboration works well in practice, as shown by the quality of the results generated by artists. We believe that in exploring this topic we made two main contributions: (1) demonstrating that seamless collaborative sculpting is possible and works well in practice, and (2) proposing a multiresolution merge algorithm that is efficient, respects user edits and handles geometry and materials.

2 Related Work

Mesh Versioning Control. A versioning control system for 3D scene was proposed in [Doboš and Steed 2012], where the scenes are split into separated components. Two versions are automatically merged if the edits act on different components, but when two users act on the same one, a conflict is generated and has to be solved manually. In contrast, we handle edits to overlapping regions of the same mesh. An approximated diff and merge algorithm was presented in [Denning and Pellacini 2013], but the slow execution times, requiring minutes for 100K faces, do not permit collaborative workflows.

Collaborative Modeling. While real-time collaborative editing of texts or 2D images is well-established, collaborative 3D modeling is still in its infancy. [Salvati et al. 2015] supports workflows for low-polygonal meshes, while [Onshape 2014] targets NURBS editing for collaborative CAD. Both use a conflict detection approach. [Li et al. 2001] uses remote locking for NURBS models disallowing conflicting edits. As discussed before, the design of these systems would not work for sculpting since most edits would be conflicted and would have to be either discarded or locked, stopping collaboration. We instead handle conflicts gracefully and merge smoothly concurrent overlapping edits.

Multiresolution Merge. The challenges posed by our merge algorithm are reminiscent of the ones encountered in the fields of mesh deformation, e.g. [Rong et al. 2008], mesh morphing [Alexa 2003] and detail transfer [Boscaini et al. 2015]. Similarly to what is done in the cited works for surfaces representation, our solution relies on geometric hierarchies, consisting in a progressively coarser to fine representations of the users' edits. Hierarchical multi-resolution geometric structures of this kind are ubiquitous tools and their usefulness has been successfully demonstrated in a variety of geometry processing applications, ranging from mesh editing [Lee et al. 1999] to mesh streaming or compression [Khodakovsky et al. 2000] and many others. The main purpose of our hierarchy is to merge edits at different spatial frequencies to better respect user intentions. Furthermore, our merge operator is orientation-agnostic as introduced by [Sorkine et al. 2004] for fine mesh details, and then adopted in multiresolution structures in [Rong et al. 2008]. From the point of view of the objectives, a related previous work is found in the shape synthesis-by-analogy technique presented in [Boscaini et al. 2015], where, however, a single deformation is to be applied on different base shapes rather than vice-versa. That approach would also be too slow for our purpose.

In spite of these similarities, the problem in our setup is unique. Our merge operator operates on *triplets* of shapes: a snapshot of the last shared mesh and the two (or more) edited ones. Consequently, our multiresolution structure targets shape *modifications* rather than shapes. Also, in our context merging happens in real time, meaning the time budget is more stringent than for typical detail-transfer applications.

3 Collaborative Sculpting

In our collaborative system, artists concurrently sculpt a model on their machine. Collaboration is established by merging all artists' edits at fixed time intervals on a central server, and broadcasting the merged version back to all artists. Merge operations are fast enough to avoid interrupting artists' workflows. Merge frequency is chosen by artists and can be made fast enough to give the impression of editing the same live version of the model.

Our prototype supports editing indexed triangle meshes with arbitrary properties stored at each vertex. We tested editing vertex positions, (i.e. 3D sculpting), and per-vertex materials, represented as microfacet BRDFs, (i.e. material painting). The same merge operation supports both. Artists' edits can be represented as the mesh difference between the last agreed upon versions of the model and the one just before the merge operation. More formally, a mesh difference ΔM can be written as the set

$$\Delta M = \{\Delta \mathbf{v}^i\} \quad with\ i \in E$$

where E is the set of the edited vertices, typically a subset of the whole mesh, and $\mathbf{v}$ are the vertex properties. Intuitively, for vertex positions $\mathbf{p}^i$, the mesh difference can be thought as the set of per-vertex translations $\mathbf{t}^i = \Delta\mathbf{p}^i$, i.e. the mesh deformation sculpted by an artist. For BRDF parameters, the difference corresponds to per-vertex shifts in the relative parameter domain.

3.1 Merge Algorithm

The core of our system is a merge algorithm that applies the edits ΔM_j concurrently sculpted by each artist j to the last agreed upon version of the model M, to compute the new merged mesh M_m (Fig. 2 shows an example merge).
Formally,

$$M_m = merge(\{\Delta M_j\}, M)$$

For edits that are disjoint on the surface, the merged version can be trivially computed by updating the vertex properties with their respective deltas. For overlapping edits, the merge operation should respect each artists' intentions as expressed by their mesh differences. A further concern that needs to be addressed is that sculpting sequences consist of both large scale variations and fine detail adjustments. Analysis of digital sculpting practices [Denning et al. 2015] reveals that both edits are alternated freely while sculpting, so situations where edits are applied concurrently at different scales are very likely.

The remainder of this section will formally describe the design goals of our merge operation and present an efficient algorithm that satisfies such goals. Our algorithm supports any number of concurrent edits and arbitrary per-vertex parameters encoded in linear spaces, namely sculpted vertex locations and microfacet BRDFs, encoded

Figure 2: *Example of applying our merge algorithm to two concurrent edits of geometry (left) and geometry and materials (right). We artificially designed these cases to be challenging since the edits overlap significantly and are performed at different spatial scales. In spite of these factors, the merged geometry respects the user intentions well.*

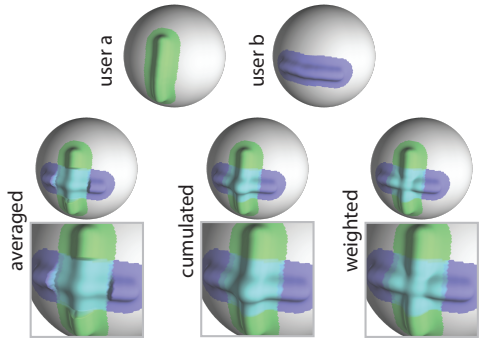

Figure 3: *Example of different per-vertex combination of translations (bottom), relative to the user edits (top). From left to right: average, cumulation and weighted sum. Note how the weighted sum is the only one that respects artists' intentions.*

in the linear spaces presented in [Di Renzo et al. 2014][1]. For simplicity of explanation though, we will present the algorithm for the case of two artists' editing vertex locations with ΔM_0 and ΔM_1.

Properties. As stated above, the merge operation should preserve all artists intentions as much as possible. This objective cannot be uniquely defined in the presence of overlapping edits. However, a few necessary properties could be described as:

- *identity*: when either ΔM_0 or ΔM_1 are locally null, then M_m must match the other difference at those locations – this cover the case of non-overlapping edits;
- *idempotence*: when ΔM_0 and ΔM_1 are locally identical, M_m must match them at those locations – this corresponds to the case where users intentions coincide;
- *commutativity*: M_m is invariant to the swapping of ΔM_0 and ΔM_1– intuitively, since edits are performed independently from each other, neither can be prioritized;
- *continuity*: small mesh differences ΔM_0 and ΔM_1 must induce small changes in M_m;
- *generality*: the merge algorithm mush not depend on specific editing operations – this allows us to support all sculpting and painting tools.

Per-vertex merge. For vertex positions, the mesh differences of each artists are sets of per-vertex translations $\mathbf{t}_0^i = \Delta\mathbf{p}_0^i$ and $\mathbf{t}_1^j = \Delta\mathbf{p}_1^j$ respectively for the edited regions $i \in E_0$ and $j \in E_1$. We can define a merge by computing a per-vertex combined translation $\mathbf{t}_m^k$ for each vertex k in the mesh. One possibility would be to average the per-vertex translations for each vertex k, i.e. $\mathbf{t}_m^k = (\mathbf{t}_0^k + \mathbf{t}_1^k)/2$. This definition though would break identity, since the effect of each sculpted edit will be halved in non-overlapping regions. This problem would be avoided by summing the translations instead, i.e. $\mathbf{t}_m^k = \mathbf{t}_0^k + \mathbf{t}_1^k$. This solution though breaks idempotence since in overlapping regions merging two identical edits will result in doubling their effect. We can obviate both concerns by averaging each artists translation weighted by their magnitudes, i.e.

$$\mathbf{t}_m^k = \frac{|\mathbf{t}_0^k|}{|\mathbf{t}_0^k| + |\mathbf{t}_1^k|} \cdot \mathbf{t}_0^k + \frac{|\mathbf{t}_1^k|}{|\mathbf{t}_0^k| + |\mathbf{t}_1^k|} \cdot \mathbf{t}_1^k.$$

This definition is equivalent to averaging translations for equal transformations, and summing them for non-overlapping edits, thus fulfilling both idempotence and identity. When edits are overlapping and non-identical, averaging them provides a reasonable definition that respects as best as possible both intentions. Fig. 3 shows an example comparing the discussed merge solutions.

Multiscale merge. When edits are performed at different scales though, the simple solution presented does not suffice. Consider the case of concurrently performing a large scale edit ΔM_0, e.g. uniformly displacing an entire portion of the edited mesh, and a fine one ΔM_1, e.g. sculpting a small detail over the displaced surface. For vertices where ΔM_1 is performed, the otherwise uniform transformation of ΔM_0 is partially negated, and in the merged modification artefacts arise. To obviate this problem, we represent edits as multiresolution hierarchies, and combine them at each hierarchical level independently. This way edits at the same spatial frequency, i.e. the same level of the hierarchy, are merged together, but independently from the other scales.

Rotational Invariance. When merging vertex positions, one surviving limitation emerges when a large scale edit modifies the local orientation of the mesh region where a concurrent detail edit is acting. The problem with this scenario is that the translation transformation is not invariant to rotation. This does not occur for BRDF parameters since rotations have no useful semantics in the appearance domain. For the geometry case, artefacts arise. Facing similar problems, [Sorkine et al. 2004] factors out the rotation of a Laplacian shape descriptor, and [Lipman et al. 2005] introduce rotation invariant coordinates to describe the shape. Our case differs in that we aim at representing shape *modifications* rather than shapes. Our solution consists simply in switching to a more generic class of transformations, namely isometries, i.e. roto-translations, instead of translations. This choice is motivated by the observation that typical sculpting edits can be locally approximated well by rigid transformations.

We represent isometries as dual quaternions [Kavan et al. 2008], since this representation is efficient and produces better interpolation than affine matrices. We efficiently approximate dual quaternion interpolation as [Kavan et al. 2008] by their weighted sum, followed by renormalization. Formally,

$$\mathbf{q}_m = \text{normalize}(\ m(\mathbf{q}_0) \cdot \mathbf{q}_0 \ \oplus\ m(\mathbf{q}_1) \cdot \mathbf{q}_1\ )$$

[1] This is akin to the support of arbitrary parameters in commercial implementation of subdivision surface algorithms

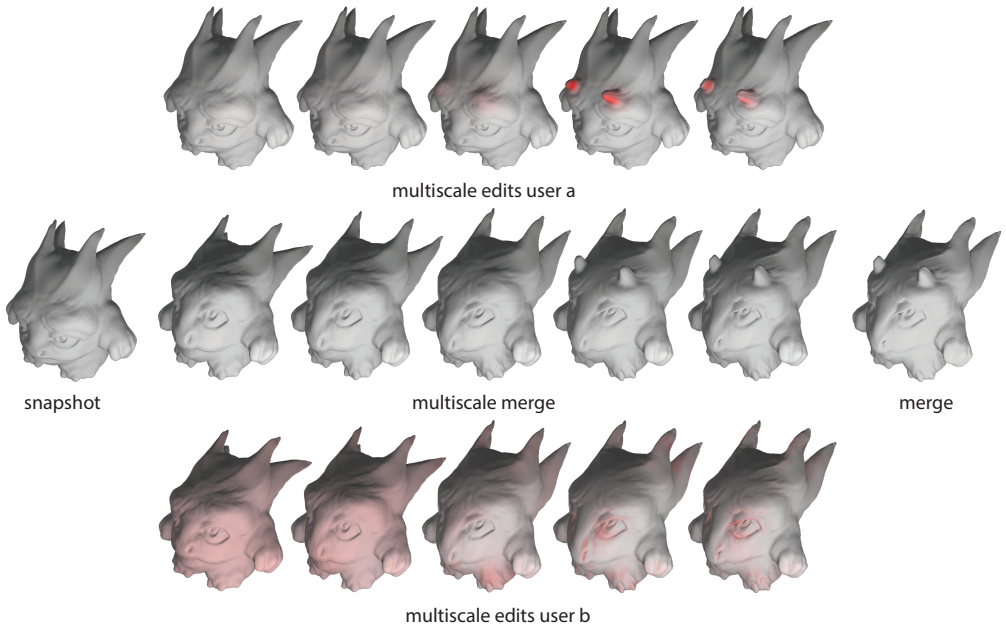

Figure 4: *Visualisation of merge hierarchy for the example in Fig. 2. Multiresolution hierarchies are shown coarse to fine from left to right. Red coloring shows the magnitude of the remaining transformation with respect to the coarser level transformations, stored at the corresponding level.*

where $\oplus$ is the quaternion addition and $m(\mathbf{q})$ is the magnitude of the isometric transformation encoded in dual quaternion $\mathbf{q}$, which we define as the distance from the identity dual-quaternion 1:

$$m(\mathbf{q}) = |\mathbf{q} - 1|$$

The isometry addition $\oplus$ is defined, as normal, as the addition of the 8 real numbers of their dual quaternion representation, preceded when needed by a flip of sign of either one operand, to take into account the sign invariance of this representation (i.e. that $-\mathbf{q}$ and $+\mathbf{q}$ represent the same isometry). Normally, the sign flip is issued when the rotational parts of the operands have a negative dot product (using the dot product of $\mathbb{R}^4$). In our case, we can make the assumption that the rotational angle is smaller than 90 degrees, so we can use a simpler and equivalent test: we just ensure that the real primal component of each operand is positive, and flip the sign of that operand otherwise. This has the important benefit of ensuring that $\oplus$ is associative in addition to commutative, when simultaneous edits from more than two users are merged together.

Multiscale Hierarchy. The simplest hierarchical structure we considered is a tree, with one leaf node per mesh vertex, and all leaves in the same level. Higher level in the hierarchy corresponds to clusters of spatially-close vertices. We store the original mesh differences by decomposing them at different levels of the hierarchy. The root node stores a global transformation, affecting the entire mesh, and each other node stores the residual transformations with respect to the summed transformation from the root to its parent. Decomposing edits over a hierarchy is not uniquely defined, with different choices producing different merged outcomes. We choose a projection that minimises the magnitudes of stored deformations, describing per-vertex transformations with a few, large modifications closer to the root, leaving the remaining minimal transformation to deepest nodes. With this projection we aim to minimize the number of nodes that will represent user edits.

In a tree, sibling nodes correspond to disjoint groups of spatially-localized mesh vertices. When edits cross these node boundaries, artefacts may appear in the final result. We address this by replacing the single parent transformation of each node with a blend of a small number of parent node transformations at the immediately coarser level, with fixed, given weights. As a data structure, this is similar to what is routinely used for skinned animations (e.g. [Kavan et al. 2008]), where each vertex is associated with a given blend of a small number of bones transformations. In our formulation, each node stores the residual transformation with respect to the averaged transformations of all its parent nodes. Fig. 4 shows an example of the multiscale hierarchy.

Hierarchy Construction. We build a multiresolution structure based on the mesh connectivity, where we take advantage of the consistent tessellation density and the relative regularity of meshes typically used in digital sculpting. Our mutiresolution structure consists of an array of levels, L_0 to L_n from finest to coarsest. Each level consists in a set of nodes, and each node stores links to up to K parent nodes in the immediately coarser level, with associated weights summing up to 1. K is a parameter of the system; our experiments indicate that $K = 7$ achieves good results. Additionally, during construction only, we use a notion of reciprocal adjacency between nodes on the same level, by adding horizontal links between pairs of nodes at the same level.

The finest level L_0 has one node for each vertex, and one horizontal link for each triangle edge. Iteratively, a coarser level L_{i+1} is constructed from the previous finer level L_i, until all levels are created. At each iteration, we populate L_{i+1} by selecting a subset of nodes of L_i with a simple heuristic, as follows. Initially, L_{i+1} is empty and no node of L_i has any parents. Then, we pass through all nodes of L_i once, in an arbitrary order: if we encounter a node n_a of L_i with no parent yet, we and add it as a new node n_b in L_{i+1}, and we add n_b in the parent-set of n_a and of all its adjacent nodes. Note that this selection depends on the arbitrary order but it is always fairly regular: no two adjacent nodes are selected, but at

least one node is selected around a non-selected node.

If the newly created level L_{i+1} has as many nodes as L_i, we discard it and the construction is over. This happens when L_i has a single node for each connected component, i.e. normally one. Otherwise, we proceed to finalize level L_{i+1}, in two steps. First, we add horizontal edges to it. For every horizontal edge in L_i, connecting n_c and n_d, we consider their first parents n'_c and n'_d in L_{i+1}, and, if they are distinct nodes not yet connected by an edge, then we add that edge. Second, we adjust the sets of parents and relative weightings. We provisionally initialize all weights in each parent-set in L_i as one over the cardinality of the set. Then we apply a smooth operator to all sets of parents. We add to the set of parent of each node the original sets of parents of all its neighbors, then renormalize it so that their sum is one again. The sum of two sets of parent is defined as the union of the two sets, with summed weights for elements present in both. Finally, we trim all the sets of parent to leave only the K parents associated with the largest weights and renormalize the weights again.

Hierarchy Statistics. In the ideal case of a fully regular mesh, this heuristic builds coarser levels with a number of nodes ranging between a ratio of 0.16 and 0.25 of immediately finer level. In real cases, we found this ratio to be between 0.25 to 0.5. The number of levels therefore typically ranges between 6 to 14. Empirically, we found the total number of nodes in all levels to be around 1.33 times the number of vertices.

Per-Vertex Isometries. For each edit, isometries are first computed at each mesh vertex in such a way that the transformed vertex matches exactly the edited one and its transformed neighborhood corresponds approximately to the edited one. A typical solution (e.g. [Sorkine et al. 2004]) consists in first fitting an affine transformation that maps a neighborhood into the other, with least squares, and then extracting the isometry closest to the affine with SVD. We prefer a faster approximation that avoids the linear minimization step. We first compute the rotational part as a quaternion for each triangular face: we apply SVD to the unique 3×3 A matrix that maps two face edges, $\mathbf{e}_{0_a}$ and $\mathbf{e}_{1_a}$, along with the original face normal $\mathbf{n}_a$, into the two corresponding transformed faces edges $\mathbf{e}_{0_b}$ and $\mathbf{e}_{1_b}$ and normal $\mathbf{n}_b$:

$$A = (\mathbf{e}_{0_b} , \mathbf{e}_{1_b} , \mathbf{n}_b) \cdot (\mathbf{e}_{0_a} , \mathbf{e}_{1_a} , \mathbf{n}_a)^{-1}$$

We then accumulate these per-face rotational part of the quaternions at vertices, and renormalize them. This will result in the rotational part $\mathbf{q}_R$ of the complete isometry, that will be represented itself as the dual quaternion $\mathbf{q} = \mathbf{q}_R + \epsilon \cdot \mathbf{q}_T$. The translational quaternion part $\mathbf{q}_T$ of each vertex is finally computed so that each vertex originally in the 3D position $\mathbf{p}_a$ is moved exactly into its edited 3D position $\mathbf{p}_b$:

$$\mathbf{q}_T = \text{quat}(0, \mathbf{p}_b/2) \cdot \mathbf{q}_R - \mathbf{q}_R \cdot \text{quat}(0, \mathbf{p}_a/2)$$

where $\text{quat}(x, \mathbf{v})$ is the quaternion having x as the real part and vector $\mathbf{v}$ as the imaginary parts.

Multiscale Isometries. Given a per-vertex isometry for each edit, we decompose it onto the hierarchy by computing, at each level, the residual isometry from the finest to coarsest level. For each level L_{i+1}, we compute the isometry of each node as the average transformation of its children in the immediately finer level L_i. We do this by zeroing all dual quaternions in L_{i+1} and then passing over each node in L_i to sum its dual-quaternion over the nodes in its parent set, accordingly weighted. We finally normalize all accumulated dual-quaternions. We then compute the residual isometry $\mathbf{q}_r$ of each node in L_i by removing from its isometry $\mathbf{q}_f$ the isometry $\mathbf{q}_p$ averaged on its parents in L_{i+1}:

$$\mathbf{q}_r = \mathbf{q}_f \cdot \mathbf{q}_p^{-1}$$

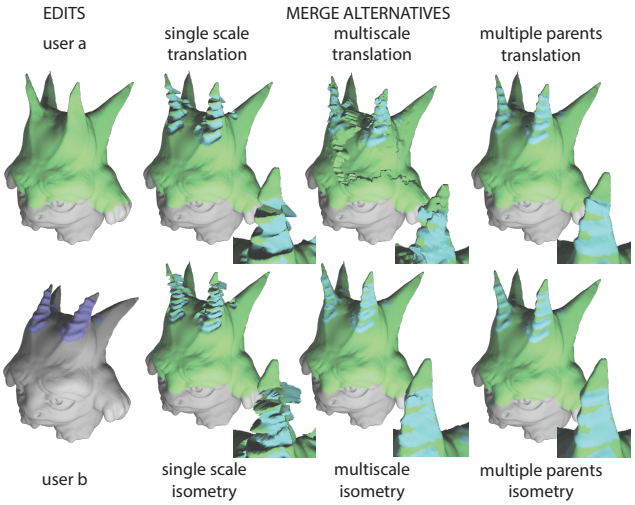

Figure 5: *Comparison of different variations of merge algorithms. Note how the inclusion of isometries, multiscale formulation and multiple parents renders the algorithm stable in this challenging case. Please refer to the text for a detailed discussion of the various algorithm's variants.*

where dual-quaternion $\mathbf{q}_p^{-1}$ denotes the inverse of dual-quaternion $\mathbf{q}_p$. During all operations, we exploit the efficiency of dual-quaternions for blending, inversion, composition, and normalization.

Final Merged Mesh. Since each edit is now represented as different per-node assignment of residual isometries over the same hierarchical structure, we can compute the merged isometry just by combining the respective dual-quaternion associated independently at each node and each level using the weighted dual-quaternion combination introduced before. We compute per-vertex combined transformations by traversing the hierarchy from coarsest to the finest level, and, at each level, computing the node transform by applying the residuals to the parent transforms. We obtain the merged mesh by applying the combined per-vertex transformations to the starting mesh vertices. Fig. 5 shows a comparison between the discussed variants of our merge algorithm.

3.2 Prototype Implementation

Sculpting tools. Our system can in principle accommodate any sculpting tool. Our prototype implementation supports a toolset similar to the one offered by Blender and based on the work from [Angelidis et al. 2006], [von Funck et al. 2006] and [Huang et al. 2007]. In particular, controlled by a mouse or a tablet device, gaussian and textured brushes are used to target and define an influence area on the mesh surface. Relative surface portions will undergo a deformation defined by a local transformation, weighted by the brush influence and an additional global strength parameter. In this formulation brushes can apply various deformations, such as relief and engraving to simulate addition and removal of plaster, local filtering for smoothing and sharpening surface features and tools like pinching, grabbing, bulging and local twisting tools, in order to mimic clay manipulation. Finally, we support edit saturation (which, during interaction, avoids the exaggerated accumulation of transformations by clamping their weighting) and symmetry enforcement tools support that can be enabled to mirror brush strokes over the surface.

Painting tools. We edit materials with a simple vertex painting in-

Editing Session		Edits			Collaboration			Comparison w/ *MeshHisto*	
Model	Length	Total	Unser A	User B	Merges	Merge Time (s)	Sync Time (s)	Conflicts	Disabled Edits
Man	38m	1197	50%	50%	337	0.018	5	92	213 (18%)
Turtle	25m	1293	46%	54%	250	0.017	5	68	148 (11%)
Alien	34m	775	50%	50%	92	0.018	20	62	290 (37%)
Dyno	45m	1327	55%	45%	119	0.018	20	74	399 (30%)
Skull	22m	647	60%	40%	63	0.017	20	50	208 (32%)
Turtle	16m	553	39%	61%	134	0.02	5	-	-
Dyno	10m	417	55%	45%	22	0.02	20	-	-

Table 1: *Statistics of our collaborative modeling sessions for models with 320K faces. When possible, we report comparisons with [Salvati et al. 2015] that only supports geometry changes.*

terface (similar to the one offered by Blender), where a user can alter per-vertex BRDF's albedo, metallicity and roughness. Since these are not linear parameters, we merge their values by first projecting them on the linear parameterization of [Di Renzo et al. 2014]. Our system allows brushes that alter geometry and material at the same time, e.g. for breaking up a clean mirror into a noisy and rough metal.
While editing we preview the manipulated material under natural illumination, by employing precomputed convolutions of an environment map [Křivánek and Colbert 2008] with spherical gaussian kernels, chosen to be proportional to a fixed number of Phong lobes [Iwasaki et al. 2012].

Communication. Merging happens at fixed time intervals. Between merges, artists' clients communicate mesh differences via asynchronous socket connections. The asynchrony ensures that the interface is not blocked even when diffs are large. Merging happens on the server that broadcast the merged version back to all clients, again asynchronously to avoid blocking. In this prototype, any user has a local copy of the mesh he/she can keep working on: all users' copies are frequently merged together by the server.

3.3 Discussion and Limitations

Comparison with *MeshHisto*. We compare our approach with [Salvati et al. 2015], a system for collaborative low-polygonal modeling. *MeshHisto*'s diff and merge algorithm work on the complete history of edits created during the results production. In that system, a conflict occurs if two editing histories modify the same mesh elements. In that case, all conflicting operations are eliminated. The algorithm presented in this paper overcomes the notion of conflict and merge is always performed even if overlapping area exists in different histories. In Table 1, we show that if we apply *MeshHisto* to our editing sequences, the total amount of conflicting operations would be very high for sculpting, making such system entirely impractical in our domain. In particular, we found that the number of rejected operations is up to 37% of the total amount of edits made by the users. As discussed before, this is due to an entirely different workflow supported by two systems. Furthermore, maintaining the full history would not be practical storage-wise when sculpting large meshes.

Limitation: Connectivity Updates. Our prototype implementation does not currently handle mesh refinement during sculpting. While we support very high resolution meshes, the overall system would benefit from dynamically applying a local topology optimization similar to [Hoppe et al. 1993]. We believe that this could be achieved without changing significantly the system architecture nor the merge algorithm if mesh refinement is deferred to the server. In particular, connectivity update is a digital sculpting well-known subproblem with established solutions; in our case, connectivity updates would change mesh updates representation to include small sequences of local operations (edge-flips, collapses and splits). The server, right before merging, would apply the same connectivity updates to its meshes and from that, merge algorithm could continue as before. Since these are just implementative tasks, we delegate them to future work.

Limitation: Scale Invariance. The situation in which our merge operator performs the least satisfactory, toward the objective of respecting users' intentions, is when a large scale edit consist in the local *rescaling* of a large region that is merged with a fine detail edit on that region. The issue is that internally we represent residual transformation as isometries, as opposed to similarities. However, it is not trivial to extend to similarities the same "correct" way in which unit dual quaternions blend isometries. Specifically, the problem with specifying the scaling separately lies in the determination of the center of scaling for the blended transformations. Here, no trivial solution works. Affine matrices, on the other hand, are known to produce robust results, but, being linear, can only produce the same result which we would get by interpolating transformed points.

4 Results

We tested our system by collaboratively modeling meshes starting from spheres to final shapes. We ran our prototype on standard desktop machines with 3.4 Ghz Intel CPUs with Wacom Tablets as input devices, and using the local network for data transport.

Models. Table 1 summarizes statistics of the sculpting sessions, shown in Fig. 6. Each model is created by applying between 417 and 1327 brush strokes equally subdivided between two artists, in sessions that vary in length between 22 and 38 minutes. We use meshes with 320k faces, enough to obtain detailed final outputs. From the figures, it is clear that artists work on overlapping regions very frequently, demonstrating the need for a fast, detail-preserving merge algorithm. We show sequences of edits in the supplemental video. We choose models with a variety of styles, but focus on organic creatures since that is the most effective domain for sculpting.

Performance. We merge meshes interactively, taking less than a tenth of a second. This is well below the average time between user strokes, which is easily above a second. In our editing sessions, we send on average 170 KB for each stroke, a number that obviously vary with brush size. This works quite well in our networking setup and is likely to work well also on common cloud services.

Collaboration. The models Turtle and Man were created by synching every 5 seconds. Sync time was chosen experimentally based on the brush strokes duration, both on geometry and appearance edits, that can last up to seconds when artists refine little features or carefully move big portions of the model. At the same time, 5 seconds is short enough to allow seamless cooperation during editing. We also perform stress test of the algorithm by setting the sync

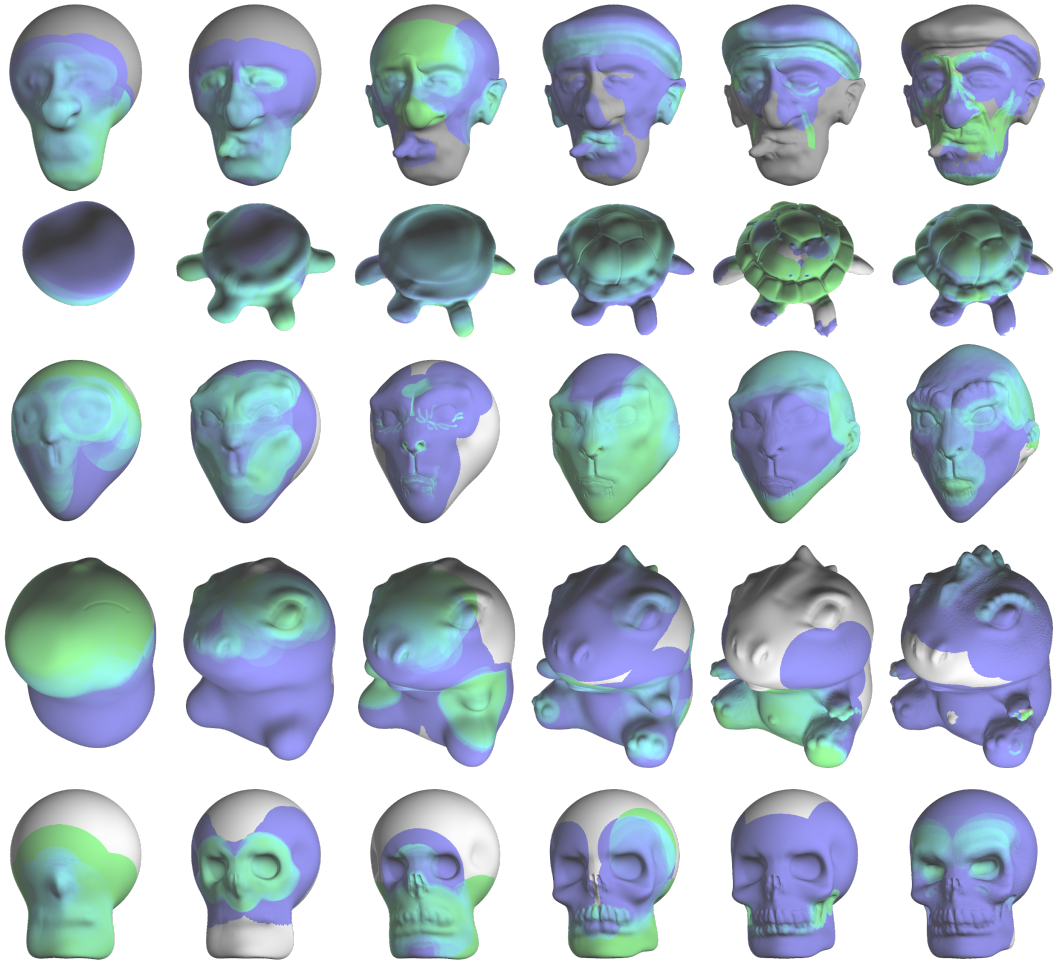

Figure 6: *Summary of the editing histories for models created in real-time collaboration between two users. Colors encode the amounts of edits applied in each snapshot by users to relative surface areas: green for user A, blue for user B, and cyan for both*

time to 20 seconds in order to simulate latency. This has the effect of increasing the number of overlapping edits. These sync times correspond to between 2 and 5 edits per users for each merge, with edits that can act on a large number of vertices depending on brush size. In both cases, our algorithm preserved user intentions very well.

In our prototype we notify that a merge was performed by using smooth fading colors of all users on edited model. An artist can immediately choose how to change its future edits for the just changed mesh. In our experiments, despite the presence of overlapping edits during interaction, this feedback allowed artists to work seamlessly without communication even in the simulated high latency case.

Materials. Fig. 7 shows sequences for adding appearance details to the Turtle and Man models. These editing sessions are between 11 and 16 minutes long. Collaborative appearance painting has overlapping frequency slightly higher than the sculpting one. This is due to the difference in speed and size of stroke interaction between painting and sculpting manipulation, e.g. area filling in albedo editing. Even in this case, the system merged seamlessly, despite the amount of data being varied is higher, since appearance manipulation require more data to be altered. Performance changes were negligible even when geometry and appearance were manipulated and merged at the same time.

Informal User Study. [Salvati et al. 2015] presents a user study that supports the preference for real-time collaboration when editing a model between users. Our system follows the same user workflow as theirs, so the same conclusion should be true. To further validate this claim, we performed an informal user study with the same methodology as [Salvati et al. 2015]. We refer the reader to the original paper for details. Here we briefly summarize our results. We asked 12 subjects to perform a matching and an open task. Six subjects are novices of 3D modeling while the others have knowledge of it. The matching task asks to match the final shape of Fig. 4 starting from a simpler mesh version, while the open task asks to customize the shape of the Dyno model. Each task lasted 5 minutes and was performed twice with the system synching every 5 seconds, as our results, or 150 seconds, to simulate offline collaboration. 10 subjects consistently rated realtime collaboration better than the other, while 2 preferred the offline workflow. Looking at the comments in the questionnaires, the 2 outliers declared that they prefer to work alone on their art, avoiding collaboration altogether. The other subjects expressed a strong preference for real-time collaboration, rather than offline one supporting our workflow entirely. This group includes all trained subjects. These results are exactly comparable to [Salvati et al. 2015]. We report comments from users preferring realtime collaboration: “I felt comfortable when I wanted to extend some edit in regions where the other artist is working”, “real-time would allow for a better cooperation”.

Figure 7: *Summary of the editing histories for both geometry and appearance editing realized in real-time collaboration between two users.*

5 Conclusions

In this paper we present a system for seamless collaborative sculpting, where collaboration is enabled by automatically sharing users edits and merging them in a reliable manner. We support edits in both geometry and materials allowing the creation of full meshes. By considering only variations of vertex properties, our algorithm is independent of the tools being used by artists to produce these changes. We use a multiscale representation to accurately handle concurrent edits applied at different spatial frequencies. In the future, we plan to augment our system by considering arbitrary topology variations, including concurrent changes to the genus, and the editing of skinned animations.

6 Acknowledgments

We would like to thank Simone Filia for his work in the recorded modelling sessions and all the subjects interviewed in our user study. This work has been partially funded by MIUR, Sapienza University of Rome and Intel Corporation.

References

Alexa, M. 2003. Differential coordinates for local mesh morphing and deformation. *The Visual Computer 19*, 2, 105–114.

Angelidis, A., Wyvill, G., and Cani, M. 2006. Sweepers: Swept deformation defined by gesture. *Graphical Models 68*, 1, 2–14.

Blender, 2014. Blender. http://www.blender.org/.

Boscaini, D., Eynard, D., Kourounis, D., and Bronstein, M. M. 2015. Shape-from-operator: Recovering shapes from intrinsic operators. *Computer Graphics Forum 34*, 2, 265–274.

Denning, J. D., and Pellacini, F. 2013. Meshgit: Diffing and merging meshes for polygonal modeling. *ACM Trans. Graph. 32*, 4, 35:1–35:10.

Denning, J. D., Tibaldo, V., and Pellacini, F. 2015. 3dflow: Continuous summarization of mesh editing workflows. *ACM Trans. Graph. 34*, 4, 140:1–140:9.

Di Renzo, F., Calabrese, C., and Pellacini, F. 2014. Appim: Linear spaces for image-based appearance editing. *ACM Trans. Graph. 33*, 6, 194:1–194:9.

Doboš, J., and Steed, A. 2012. 3d diff: an interactive approach to mesh differencing and conflict resolution. In *SIGGRAPH Asia 2012 Technical Briefs*, 20:1–20:4.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W. 1993. Mesh optimization. In *ACM SIGGRAPH '93*, 19–26.

Huang, X., Li, S., and Wang, G. 2007. A gpu based interactive modeling approach to designing fine level features. In *Graphics Interface '07*, 305–311.

Iwasaki, K., Furuya, W., Dobashi, Y., and Nishita, T. 2012. Real-time rendering of dynamic scenes under all-frequency lighting using integral spherical gaussian. *Comput. Graph. Forum 31*, 2, 727–734.

Kavan, L., Collins, S., Žára, J., and O'Sullivan, C. 2008. Geometric skinning with approximate dual quaternion blending. *ACM Trans. Graph. 27*, 4, 105:1–105:23.

Khodakovsky, A., Schröder, P., and Sweldens, W. 2000. Progressive geometry compression. In *ACM SIGGRAPH '00*, 271–278.

Křivánek, J., and Colbert, M. 2008. Real-time shading with filtered importance sampling. *Computer Graphics Forum 27*, 4, 1147–1154.

Lee, A. W. F., Dobkin, D., Sweldens, W., and Schröder, P. 1999. Multiresolution mesh morphing. In *ACM SIGGRAPH '99*, 343–350.

Li, F. W. B., Lau, R. W. H., and Ng, F. F. C. 2001. Collaborative distributed virtual sculpting. In *IEEE VR '01*, VR '01.

Lipman, Y., Sorkine, O., Levin, D., and Cohen-Or, D. 2005. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph. 24*, 3, 479–487.

Onshape, 2014. Full-cloud 3d cad system. https://www.onshape.com/.

Rong, G., Cao, Y., and Guo, X. 2008. Spectral mesh deformation. *The Visual Computer 24*, 7-9, 787–796.

Salvati, G., Santoni, C., Tibaldo, V., and Pellacini, F. 2015. Meshhisto: Collaborative modeling by sharing and retargeting editing histories. *ACM Trans. Graph. 34*, 6, 205:1–205:10.

Sorkine, O., Cohen-Or, D., Lipman, Y., Alexa, M., Rössl, C., and Seidel, H.-P. 2004. Laplacian surface editing. In *EG/ACM SGP '04*, 175–184.

von Funck, W., Theisel, H., and Seidel, H.-P. 2006. Vector field based shape deformations. *ACM Trans. Graph. 25*, 3, 1118–1125.