Using Change-sets to Achieve a Bounded Undo and Make Tutorials in 3D Version Control Systems

Rafael S. T. Vieira, Joaquim B. Cavalcante-Neto, Creto A. Vidal
Computer Science Department
Federal University of Ceará (UFC)
Fortaleza, Brazil
Email: rafaelstv@lia.ufc.br

Guillaume Vialaneix, Claudio T. Silva
Computer Science and Engineering Department
NYU-Poly
New York City, US
Email: csilva@nyu.edu

***Abstract*—A system that records changes made to a file is called a Version Control System (VCS). Even thought VCSs may store all kind of files, we focus on changes made to polygonal meshes files. Our method allows the user to track the history of topological and geometrical changes to part of a model. This part is selected through a bounding box selection mechanism, and the user can track the change-sets of the selected mesh subset, i.e., the user can see the difference between any consecutive versions of the modeling sequence. With that mechanism, it is possible to construct a sub-tree associated with the selected region to serve as a tutorial on how that part was modeled. That sub-tree also allows the user to undo local changes that do not propagate to the whole mesh. That, so called, bounded undo is an important feature of our method. Despite the important contributions, we also point out some current limitations to our method and discuss ways that might overcome them.**

***Keywords*-mesh editing; modeling.**

I. Introduction

Our problem is how to allow an artist to undo unwanted changes on a polygonal mesh respecting spatial constraints. Currently 3D modelers allow to undo changes, but only linearly in time. This imposes an order, as such, more than an artist wants can be removed.

The data structure that stores changes made to a polygonal mesh is a version control tree (VCT). VCT implicitly infers a hierarchy of dependence between operations, which means that removing a node forces dependent nodes to be removed. While traversing a VCT, we find that the numbering of faces and vertices may change, egde conections may differ, and overlapping elements and non-manifold structures may emerge. Our algorithm is able to deal with all those cases.

Contribution: Our method offers two main contributions. First, it generates specialized sub-trees for any mesh regions, which are useful for tutorials. Second, it offers bounded undo in a mesh VCS, i.e., it allows the user to select a region of the mesh, and reverse the effects of modeling changes restricted to the selected region (to undo features from a mesh). For instance, we are able to remove a nose without creating holes or necessarily changing its topology (Figure 1).

In the Section II, we present a brief history for version control systems, and an overview of existing methods. Then, we describe our method in the Section III, show our results in the Section IV, and summarize our method as well as discuss future work in the Section V.

II. Background and Related Work

Version control system: A version control system (VCS) is a tool that allows tracing changes made on data. In this paper, VCSs for 3D meshes are referred to as 3D VCS.

Categories of 3D VCS methods: We can sort the methods available for 3D VCS in three categories:

- Database[1], [2], [3], [4]: in which modeling software deal with mesh data either by storing their information in data files, or by generating a sequence of mesh commands on the fly. These methods try to determine the best course to follow.
- Visualization[5], [6], [7], [8]: These methods seek to measure the editing frequency of mesh regions in the mesh construction sequence, or to display meshes in the most human-readable way possible by studying and analyzing the data.
- Geometry Processing [9], [10], [11], [12], [13]: in which the properties of the mesh are used in order to improve the descriptive power of the VCS. The systems in this category use an enriched version control tree to generate new meshes or to determine provenance according to which kind of information they add or extract from the mesh.

Previous to our work, Denning et al.[6] proposed a way of gathering commands by frequency for building a tutorial, and later Denning and Pellacini [10] worked on a merge algorithm for 3D version control systems using mesh edit distance. Dobŏs working in the area of 3D VCS [14], uses clustering based on time for provenance [12], and developed a merge algorithm using scene graphs [9]. However, most of his work relates to the database category [2], [3]. His thesis summarizes most of his previous works [13] for further information.

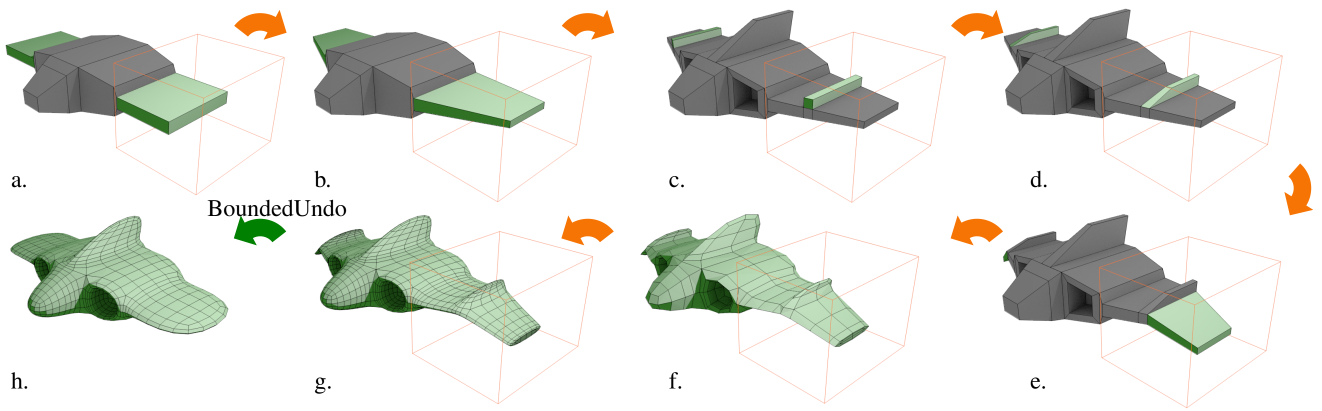


Figure 1: The construction sequence of the left wing from a spaceship's mesh (a partial tutorial; Subsection IV-A): a) Extruding faces; b) Moving vertices; c) Extruding faces; d) Moving vertices; e) Moving faces; f-g) Smoothing mesh. Bounded undo (h) was achieved by selecting the left wing (orange bounding-box), and undoing steps (e), (d), (c) and (b). Green faces are in the change-set (i.e., difference between two meshes on the original sequence); and gray faces are constant and outside the change-set. Notice that a simple undo cannot achieve the result shown in (h), because there are other steps between the operations shown.

Our method – to extract spatial subsets from a version control tree and using them for bounded undo – is in the geometry processing category. Existing methods on this category are unsuitable for bounded undo due to their ways of clustering data that are unaware of spatial changes (frequency, and scene graphs). For instance, Denning [11] allows highlighting features from models, but does not consider features that may be disconnected, or inside the mesh (for instance, undoing a mouth using his method may leave teeth behind the scene). Moreover, because his technique does not have a classification method for operations, an undo operation affects other operations that must not be removed (global operations, explained later). The same is also true to the technique from Dobŏs [13], due to his scene graph analysis only dealing with low level features. The most recent works from Denning and Pellacini [10], [11] continue clustering data by frequency.

Unrelated to 3D VCS, the works by Funkhouser et al.[15] and Zheng et al.[16] hold some similarities to ours, but lack a gradual spatial selection, and is unconcerned with change-sets. Notice that we gather changes on a volumetric region along the production of a graphic asset. And our method can work together with the method of Denning et al.[6]; while their method is a good way of displaying mesh construction, it is unable to gather changes by volumetric mesh regions. In the next section, we detail our method (Section III).

III. Method

In the following subsections, we discuss the concepts (Subsection III-A), the prerequisites (Subsection III-B), and the overview of the algorithm (Subsection III-C). Then, we present its steps: classification of operations (Subsection III-D), the propagation of the selection (Subsection III-E), and the sub-tree generation process (Subsection III-F). Finally, we present a synthetic view of the algorithm (Subsection III-G).

A. Concepts

This subsection describes necessary concepts for understating our work.

Version Control Tree (denoted E): The data structure used by version control systems is a version control tree (VCT): a set of connected nodes without cycles. The tree is also a directed acyclic graph (DAG) [17] if a tree has orientation between nodes. A DAG may serve as input to our algorithm, but, throughout this paper, a version tree refers to a VCT.

Node (denoted $t \in \mathbb{N}$): A unit of information, which contains ids, labels, relations with other nodes, and high-level API functions, i.e., software-specific functions (with their parameters).

Operation (denoted OP): Mesh reshaping through insertion, deletion, or transformation of a mesh subset. Examples of operations are deleting a face, rotating a part of a mesh, adding hair details to a given mesh, etc.

Change-set (denoted Δ): The result of an operation, i.e., the set of differences between two meshes on the same tree (the mesh before and after an operation). All mesh elements (i.e., vertices, edges, and faces) changed by an operation belong to a change-set, which implies its spatial volume.

Construction sequence: A path on the tree: a sequence of nodes that goes from the root to a leaf. A set of operations made in a prefined order, e.g., $0, 1, ..., t-1, t$.

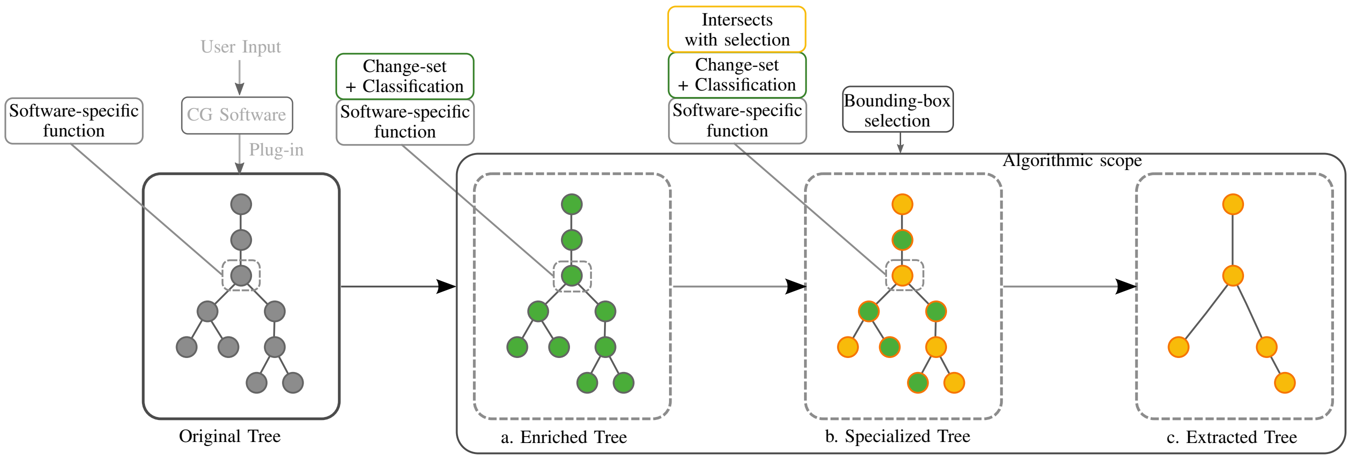

User Input
CG Software
Plug-in
Software-specific function
Change-set + Classification
Software-specific function
Intersects with selection
Change-set + Classification
Software-specific function
Bounding-box selection
Algorithmic scope
Original Tree
a. Enriched Tree
b. Specialized Tree
c. Extracted Tree

Figure 2: Overview of our algorithm (Subsection III-C) with the original tree in gray as its input: a) In green, all nodes of the original tree are enriched with the information of its change-set and a classification; b) A bounding-box selection is made and propagated, each node (with its change-set) that intersects with the propagated selection is in yellow; c) All non-marked nodes are discarded to obtain the sub-tree.

Version (denoted $\mathbb{V}$): The resulting mesh of a construction sequence, i.e., a mesh after all change-sets are applied to the mesh at the beginning of the sequence.

In the next subsection, we present the prerequisites to our algorithm (Subsection III-B).

B. Prerequisites

The required inputs to our method are a version tree (or a DAG) and a selection. All commands issued by the user, while using some modeling software, must be stored in a VCT. The root of a version tree contains an API function that inserts, for instance, a cube or a triangle in the scene, and every API function applied to the initial object becomes a node in the version tree.

Operations from the version tree must have input parameters (a mesh region), and an API function [18] from a 3D modeler(Figure 3). They must also guarantee atomicity, i.e., if two or more API functions use the same parameters, then the software that is building the VCT must save each API function with its parameters. An operation is just the API function when parameters are unnecessary.

Our method extracts a change-set from each operation; therefore, parameters inside a node must contain all elements (i.e., faces, edges and vertices) on which the API function acts. A connection between attributes of the elements, analogous to the one Maya allows [19], is also a parameter to the purpose of this paper. In the next subsection, we give an overview of our algorithm (Subsection III-C).

C. Overview of the algorithm

Our algorithm uses as input a version tree and a bounding-box selection (with an index for a VCT node) and proceeds as follows (Figure 2): first, the change-sets of the operations are found and VCT nodes are classified as local or as global

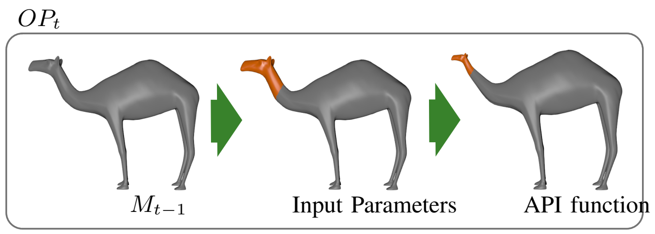


Figure 3: An operation (OP_t) in a node from the version tree acts upon a state of the mesh (M_{t-1}) and requires its input parameters (in orange) and an API function from the modeler (Subsection III-B).

(Subsection III-D); second, the input bounding-box selection is propagated upwards and downwards in the version tree according to the classification obtained from the first step (Subsection III-E) – depth-first search [20] was used to visit every node – and intersections between the propagated selection and the change-set for each node are computed; last, a sub-tree (i.e., a subset of the tree) is generated with nodes that belong to any non-empty intersection set (Subsection III-F).

Figure 2 provide a graphical overview of our algorithm. In the next subsections, we detail these three steps, starting with the classification of the operations (Subsection III-D).

D. Classification of the operations

We have first traversed the whole VCT, generating meshes from each node. Geometric hashing[21] allow us to use coordinates of a vertex as an index (coordinates are used as entries in a formula to output an index which references a fixed-size array), and to find a mapping set of vertices between all related meshes (parent and child) in a construction

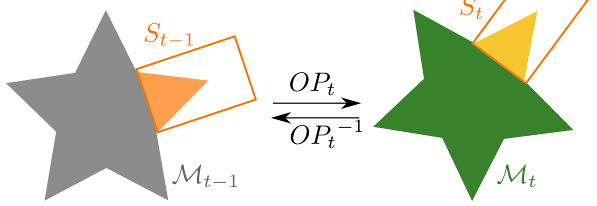


Figure 4: Evolution of the selection with a single operation on the mesh (Subsection III-E): assuming that $t-1$ is the parent of the node t without loss of generality, whenever the operation $OP_t : M_{t-1} \to M_t$ is global and modifies the geometry, the corresponding selection S_{t-1} is transformed into S_t by the same operation. When propagating upwards, the operation OP_t^{-1} is used to transform S_t into S_{t-1}. The color palette is the same as Figure 7 uses.

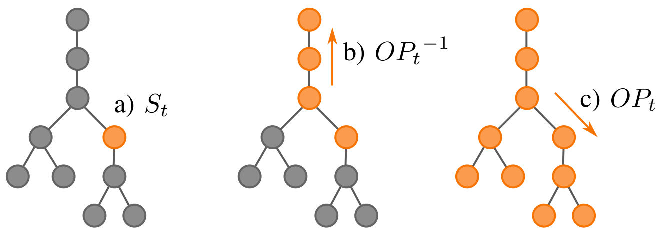


Figure 5: The propagation of a selection (Subsection III-E): visited nodes are in orange and unvisited nodes in gray. The input step is a bounding-box selection at a node t (a). From there, the spatial selection is propagated upwards by applying the inverse of global operations when needed (b). Then, the selection is propagated downwards using, whenever necessary, the same geometrical transformations applied to the mesh (c).

sequence. This means we know how each vertex that has not changed maps from one node to another. After this process is done, we have used the complement of mapped-set to obtain the change-set. Notice that this change-set can be found in both parent and child node (complement is obtained from both nodes).

Vertices that do change can be associated to the previous ones by propagating topology from mapped vertices, and, since we know change-sets from both parent and child, by mapping parent-change-set volume with child-change-set volume by hierarchical subdivision of volume[22] (bounding-boxes[23] around change-sets are divided recursively until all vertices are mapped respecting topology).

A mapping set allow us to classify a change-set in a node (denoted $\Delta_U(OP_t)$) as *global*, if the change-set region contains the whole mesh, or, as *local* if it does not. Global operations determine implicitly points in a VCT where mapping vertices start and stop, and because of this, they cannot be removed. Each global operation is a root node of its own change-set-tree.

E. Propagation of the selection

A spatial selection is made using a *Bounding Box* (BB [23]) over a version following the classification of operations (Subsection III-D). This selection is then propagated upwards toward the root and then downwards traversing the entire tree (Figure 5); hence, every node of the tree has a selection associated with it. This is done to keep track of the selected volume. Selections are affected by specific global operations that alter the mesh geometry – translation, rotation, shearing, scaling, etc (Figure 4).

F. Sub-tree Generation

The specialized sub-tree is generated after the propagation of the selection (Subsection III-E) by traversing the whole tree as well as testing which change-set vertices are inside or outside the corresponding spatial selection for each node; and, if necessary, by also testing intersection between their faces (denoted $\Delta_{S_t}(OP_t)$). To speed up this process another kind of hierarchical subdivision of space is used, so that tests can be performed locally [24].

A node is inserted into the sub-tree if it produced a change-set inside the volume defined by the selection; once a tree is traversed, a sub-tree is obtained, and can be modified. It is possible to display the original version of the main tree and the specialized sub-tree (Figure 7).

Bounded Undo: To remove operations based on a mesh region from a version tree implies removing local nodes from the generated sub-tree after all previous steps. Any bounded undo operation to the sub-tree is applied by making the node transparent in the main tree: the parent of the undone node becomes the parent of its children. If necessary, interaction with the sub-tree generates a warning of possible data loss from deleting global nodes in the sub-tree (local nodes can still be removed). In the next subsection, the whole algorithm for finding the specialized sub-trees is presented (Subsection III-G).

G. Algorithm

The algorithm for extracting the sub-tree (Algorithm 1) is described in the following. Given a subset $U \subseteq \mathbb{R}^3$, assume a single spatial selection S_t exists such that $\forall_{t \leq n} : (U \supset S_t, M_t(F_t, V_t))$, where $t, n \in \mathbb{N}$; S_t is initially an axis-aligned bounding box; M_t is a mesh, i.e. a pair of a set of faces F_t as well as a set of vertices V_t; n is the number of nodes in a VCT, and $M_0 = \emptyset$.

Next, assuming that $t-1$ is the parent of the node t without loss of generality and $y \in \{F, V\}$, we define as follows: the version tree E as the union of all operations $E = \bigcup_{t=0}^{n} OP_t$; an operation $OP_t : M_{t-1} \to M_t$ as

$$OP_t = \{x \in \mathcal{P}(\Lambda) \mid \Lambda = \{\forall t : \bigcup \{D(y), I(y), T(V_{t-1})\}\}\}, \tag{1}$$

where $\mathcal{P}(\Lambda)$ is the power set of Λ, $D : M_t \to M_t - \{y\}$ and $I : M_t \to M_t \cup \{y\}$ are respectively the deletion and the insertion of objects, T:$\mathbb{R}^3 \rightarrow \mathbb{R}^3$ is a transformation; and the region affected by a spatial selection as

$$\Delta_{S_t}(OP_t) = \bigcup_{i=t-1}^{t} \{y_i \in M_i \mid (y_{t-1} \neq y_t) \wedge ((F_i \cap S_i) \vee (V_i \in S_i))\}. \tag{2}$$

All operations inside the version tree set E are analyzed accordingly to the change-set regions Δ_U (Equation 2) in steps 1 to 5. When operations affect the whole subset U – rigid body transformations, scaling, shearing, etc – they are classified as global; otherwise they are classified as local ones. The method propagates the input selection (step 6) for previous nodes in a path (steps 9 to 15); going upwards in the version tree and applying on S_{node} the inverse matrix in OP^{-1}_{node} if it is a global node. Finally, in steps 17 to 26, the method finishes creating the spatial selection for unvisited nodes (downwards into the version tree), discovers which regions intersect with the selection $\Delta_{S_{node}}$ at Step 22 (Equation 2), and produces the set E_S. In the next section, we present the obtained results with this algorithm (Section IV).

Algorithm 1 Calculate $E_S \subset E$

```
1: for each node of $E$ do
2:     $type(OP_{node}) \leftarrow local$
3:     if $\Delta_U(OP_{node}) \supset \mathcal{M}_{node}$ then
4:         $type(OP_{node}) \leftarrow global$
5: $S_t \leftarrow$ BB selection in a version (node $t$)
6: $node \leftarrow t$
7: $previousNode \leftarrow parent(node)$
8: while $node \neq root$ do
9:     $S_{previousNode} \leftarrow S_{node}$
10:    if $type(OP_{node}) == global$ then
11:        $S_{previousNode} \leftarrow OP^{-1}_{node}(S_{node})$
12:    $node \leftarrow previousNode$
13:    $previousNode \leftarrow parent(node)$
14: $node \leftarrow root$
15: while $node \neq t$ do
16:    if $S_{node} == \emptyset$ then
17:        $previousNode \leftarrow parent(node)$
18:        $S_{node} \leftarrow S_{previousNode}$
19:        if $type(OP_{node}) == global$ then
20:            $S_{node} \leftarrow OP_{node}(S_{previousNode})$
21:    if $\Lambda_{S_{node}}(OP_{node}) \neq \emptyset$ then
22:        $E_S \leftarrow E_S \cup OP_{node}$
23:    $node \leftarrow node + 1$
```

IV. Results

In this section, we describe obtained results from tutorials as well as bounded undo (Subsections IV-A and IV-B) and discuss the limitations of the current algorithm (Subsection IV-C).

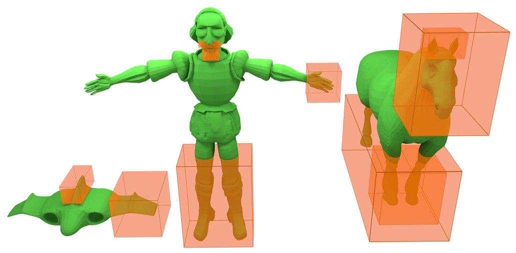


Figure 6: Versions of 3 meshes in a VCS and spatial selections (in orange) associated with each one of them. Through some of these selections, we have extracted specialized sub-trees for creating the tutorials (Subsection IV-A) and for applying the feature of bounded undo (Subsection IV-B).

Table I: Compairison of our algorithm and Denning's.

Mesh	Deltas	Timing	
		Ours	3DFlow
Helmet	1321	1.7s	5s
Hydrant	691	1.25s	4s
Robot	1810	8s	15s
Shark	1457	1.1s	6s
Biped	1267	0.45s	5s

A. *Creating a tutorial*

A tutorial is an account or explanation of a subject, printed or on a computer screen, intended for private study. Our tutorial is made of an image printed on a display device with a brief explanation on each step to teach how a specific region was created. However, a specialized sub-tree must be found before a modeling tutorial is created. Using a BB selection over a mesh region (in a version; Figure 6), our algorithm extracted all necessary nodes; the obtained construction sequences from the sub-tree are examples of tutorials such as the wings from the spaceship mesh (Figure 1), the front legs as well as the ears of the horse mesh (Figure 7), and the mustache of Don Quixote mesh (Figure 9). Next, we show the results from the bounded undo (Subsection IV-B).

B. *Bounded undo*

To achieve bounded undo, we have used a bouding-box (BB) as a spatial selection over a mesh volume in a version (Figure 6) to obtain a specialized sub-tree. Then, we removed some or all sub-tree nodes to make new versions in the VCT. For instance, we have removed some details from the wings of the spaceship mesh, which created a new version as can be seen in Figure 8; no modeling was necessary at all.

Albeit we have selected just the left wing (Figure 6), since some undone operations extended to both wings (Figure 1), the bounded undo affected both of them (Figures 8c and 8d).

In the best scenario, operations are limited to the interior of the selection; an example is the construction sequences of the front legs from the horse mesh (Figure 7). We also

Table II: Analysis of the algorithm. O: original VCT; E: enriched tree; S: specialized tree; and T: specialized sub-tree (Figure 2)

Mesh	A selection	No. of nodes: sub-tree / VCT	Timing		
			$O \rightarrow E$	$E \rightarrow S$	$S \rightarrow T$
Spaceship	Vertical Stabilizer	8 / 92	<1s	<1s	<1s
	Left Wing	16 / 92	<1s	<1s	<1s
Horse	Ears	147 / 1196	<1s	1.4s	<1s
	Head	353 / 1196	<1s	1.7s	<1s
	Front Legs	104 / 1196	<1s	1.5s	<1s
	All Legs	210 / 1196	<1s	1.8s	<1s
Don Quixote	Mustache	177 / 1648	<1s	5.2s	<1s
	Goatee	144 / 1648	<1s	5.2s	<1s
	Left Hand	110 / 1648	<1s	5.4s	<1s
	Legs	134 / 1648	<1s	7.3s	<1s

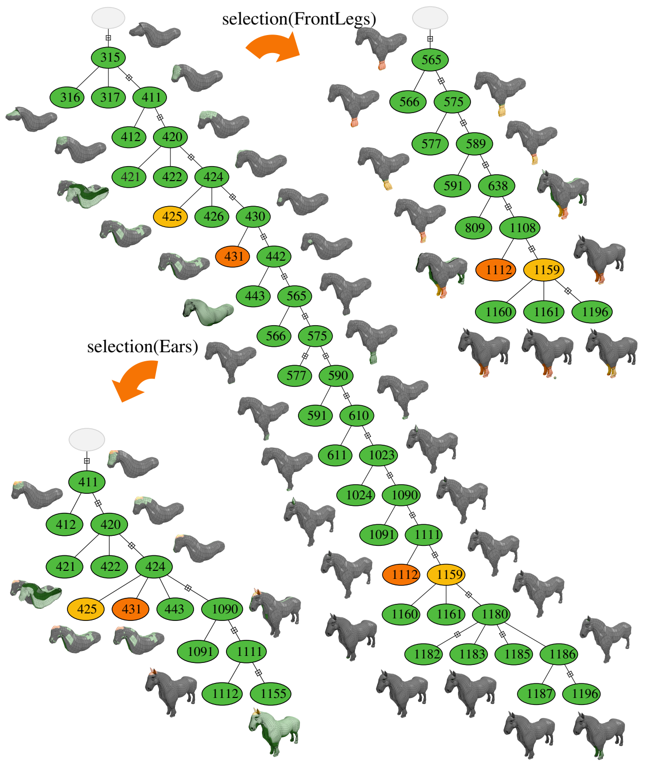


Figure 7: This image illustrates how tutorials (Subsection IV-A) are made: two specialized sub-trees were generated from the main version tree of the horse mesh. Nodes in yellow show that they have preserved their original relationship on the sub-tree, while other nodes in orange show that they have changed. The full tree from the mesh (in the middle) has 1096 nodes, the sub-tree from selecting the ears has 147 nodes (bottom left) and the sub-tree from selecting the front legs has 104 nodes (top right). Orange faces belong only to the selected region; green faces belong to the change-set; yellow faces belong to the change-set as well as the selected region; gray faces are apart from the change-set as well as the selection.

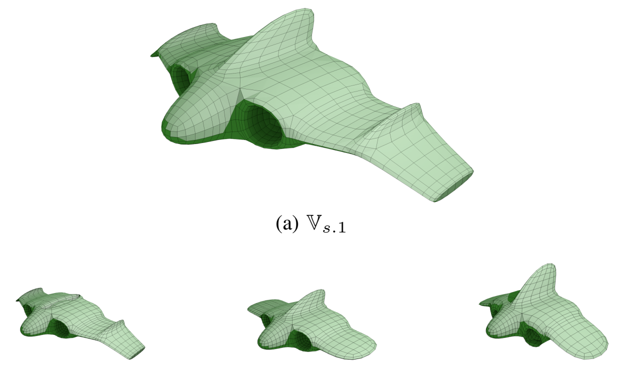

(a) $\mathbb{V}_{s.1}$
(b) $\mathbb{V}_{s.1}(\mathbb{V}_{s.2})$
(c) $\mathbb{V}_{s.1}(\mathbb{V}_{s.3})$
(d) $\mathbb{V}_{s.1}(\mathbb{V}_{s.4})$

Figure 8: Changing a specialized sub-tree, we achieved the bounded undo (Subsection IV-B): a) An unchanged version $\mathbb{V}_{s.1}$; b) The version $\mathbb{V}_{s.1}$ after removing all operations on the vertical stabilizer $\mathbb{V}_{s.1}(\mathbb{V}_{s.2})$; c) The version $\mathbb{V}_{s.1}$ after removing some operations on the wings $\mathbb{V}_{s.1}(\mathbb{V}_{s.3})$; d) The version $\mathbb{V}_{s.1}$ after removing some operations on the wings and some (a subset) on the vertical stabilizer $\mathbb{V}_{s.1}(\mathbb{V}_{s.4})$.

have removed the ears from the horse mesh (Figure 11) and some details from the Don Quixote mesh: mustache (Figures 9 and 10b), goatee (Figure 10c), and hair (Figure 10d).

In all examples, we have applied the bounded undo to the original version by deleting nodes from the sub-tree, but notice that the affected versions within a VCT depend on the undone, local operations. The removed nodes became invisible in the main tree, but it was possible to restore undone nodes from the sub-tree and the previous state of the VCT. In the following Subsection, we discuss the current limitations of the algorithm (Subsection IV-C).

C. Discussion

Storing all changes made by an artist in a VCS may be seen as a limitation if someone stores all meshes. However, a function with its parameters, as we require, has its size comparable to a single line of a text file (around 1024 bytes). As such, if the artist works over 40 hours a week, executing a function each second, we expect to have 144 MB of data

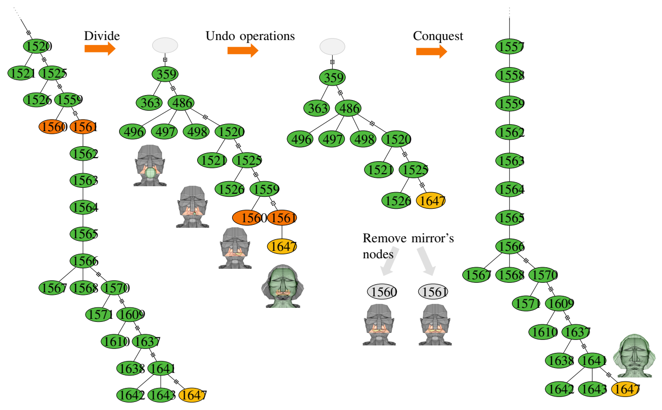
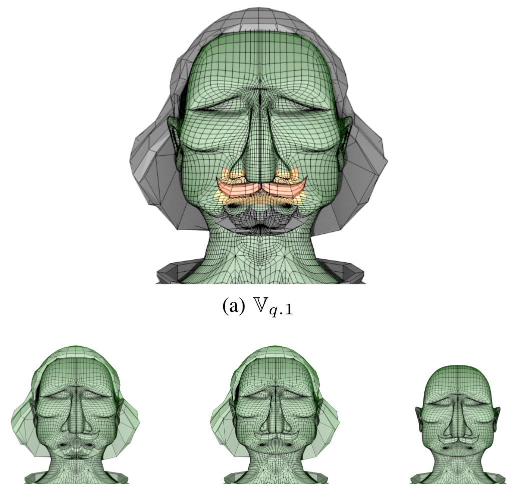


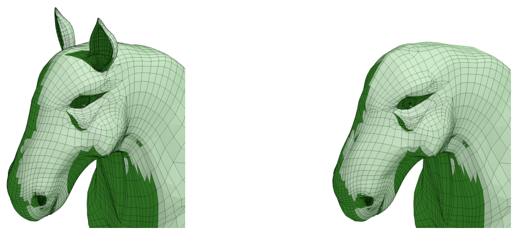
Figure 9: This sequence illustrates the bounded undo feature (Subsection IV-B), explained under the divide-and-conquer paradigm: The sub-tree is extracted (177 nodes) from the main tree with 1648 nodes (divide); two nodes are removed from the sub-tree that is merged back in the main tree (conquer). The node 1647 in yellow shows a version $\mathbb{V}_{q.1}$ and his modified version $\mathbb{V}_{q.1}(\mathbb{V}_{q.2})$ after the process is completed (Figures 8, 10, and 11 shows more examples of this process). The meshes use the same color palette as Figure 7 does.



(a) $\mathbb{V}_{q.1}$

(b) $\mathbb{V}_{q.1}(\mathbb{V}_{q.2})$ (c) $\mathbb{V}_{q.1}(\mathbb{V}_{q.3})$ (d) $\mathbb{V}_{q.1}(\mathbb{V}_{q.4})$

Figure 10: Changing a specialized sub-tree, we achieved the bounded undo (Subsection IV-B): a) An unchanged version $\mathbb{V}_{q.1}$; its color palette is the same as Figure 7 uses; b) The version $\mathbb{V}_{q.1}$ after removing operations on the mustache $\mathbb{V}_{q.1}(\mathbb{V}_{q.2})$; c) The version $\mathbb{V}_{q.1}$ after removing all operations on the goatee $\mathbb{V}_{q.1}(\mathbb{V}_{q.3})$; d) The version $\mathbb{V}_{q.1}$ after removing all operations on the goatee and on the hair $\mathbb{V}_{q.1}(\mathbb{V}_{q.4})$.



(a) $\mathbb{V}_{h.1}$ (b) $\mathbb{V}_{h.1}(\mathbb{V}_{h.2})$

Figure 11: Changing a specialized sub-tree, we achieved the bounded undo (Subsection IV-B): a) An unchanged version $\mathbb{V}_{h.1}$; b) The version $\mathbb{V}_{h.1}$ after removing all operations on the ears $\mathbb{V}_{h.1}(\mathbb{V}_{h.2})$.

per week, in a year we shall have approximately 7 GB of storage, which any regular hard disk is able to handle. Please notice, that textures or images, as well as any other files that may exist in the VCS do not matter to our algorithm, only geometric data.

Our algorithm has limitations: the specialized sub-tree may contain operations that extend beyond the desired area. For illustration, a specialized sub-tree was generated selecting the space around the mustache with a BB (Figure 6). However, since this selection intersects with part of the face (Figure 10a), some operations that are unrelated to the mustache are present in the sub-tree. In practice, this is insignificant to the size of the sub-trees, which remained smaller than the whole version tree – the sub-tree being a member from the power set of the VCT. The decrease on its size was up to over 90% (Table II) while its display and interaction improved.

A bounded undo may affect more than the desired area if the change-set is greater than the selected volume (trespassing the boundary). For instance, the left wing from the spaceship mesh was selected (Figure 1), however, some nodes had operations affecting both wings (their attributes were connected). Therefore, the removal of operations from the left wing has also affected the right wing (Figure 8).

Some objects in the scene may need to be removed after the bounded undo. For instance, the mustache of Don Quixote mesh was first created as a sphere that was dragged and transformed in the scene (Figure 9). Therefore, a sphere is left on the associated versions if we remove all operations associated with the mustache.

Traversing the whole version tree while performing geometrical operations requires time (Table II), and our algorithm was parallelized to increase its speed, and we have run multiple selections at a multi-core machine to create multiple sub-trees at a time. Our speed-up was so great that we have achieved better results than 3DFlow algorithm [25], which is currently among the state of the art algorithm for change-sets operations (Table I). Notice that it is as fast as $\mathcal{O}(n \log m)$ to access a known sub-tree; where n is the number of sub-tree nodes, and m is the number of VCT nodes.

In the following section, we summarize our work (Section V).

V. Conclusion

We have introduced a new feature for 3D VCS, a bounded undo. Our method made a specialized sub-tree from a VCT based on change-sets and, and removed operations on a mesh volume while preserving the remaining mesh respecting the limitations discussed. Moreover, another application for our specialized sub-tree is a tutorial for specific mesh features; each construction sequence from the sub-tree with a description of the operations made is a candidate. It is possible to improve the results of tutorials by combining our work with the one from Denning et al.[6].

As future work, we expect to find more properties for change-sets, to improve the selection to a more flexible geometry, and to find new applications for our sub-tree – experimenting with the automatic generation of graphic assets for VCSs.

References

[1] H.-T. Chou and W. Kim, "A unifying framework for version control in a cad environment," in *Proceedings of the 12th International Conference on Very Large Data Bases*, ser. VLDB '86. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1986, pp. 336–344.

[2] J. Doboš and A. Steed, "3d diff: An interactive approach to mesh differencing and conflict resolution," in *ACM SIGGRAPH 2012 Talks*, ser. SIGGRAPH '12. New York, NY, USA: ACM, 2012.

[3] J. Doboš, K. Sons, D. Rubinstein, P. Slusallek, and A. Steed, "Xml3drepo: A rest api for version controlled 3d assets on the web," in *Proceedings of the 18th International Conference on 3D Web Technology*, ser. Web3D '13. New York, NY, USA: ACM, 2013, pp. 47–55.

[4] P. Nyamsuren, S.-H. Lee, and S. Kim, "A web-based revision control framework for 3d cad model data," *International Journal of Precision Engineering and Manufacturing*, vol. 14, no. 10, pp. 1797–1803, 2013.

[5] S. L. Su, S. Paris, F. Aliaga, C. Scull, S. Johnson, and F. Durand, "Interactive visual histories for vector graphics," no. MIT-CSAIL-TR-2009-031, June 2009.

[6] J. D. Denning, W. B. Kerr, and F. Pellacini, "Meshflow: Interactive visualization of mesh construction sequences," *ACM Trans. Graph.*, vol. 30, no. 4, pp. 1–66, Jul. 2011.

[7] H.-T. Chen, T. Grossman, R. Schmidt, B. Hartmann, G. Fitzmaurice, and M. Agrawala, "History assisted view authoring for 3d models," *ACM Conference on Human Factors in Computing Systems*, 2014.

[8] J. Freire, D. Koop, F. Chirigati, and C. Silva, "Reproducibility using vistrails," *To appear in Implementing Reproducible Computational Research*, 2014.

[9] J. Doboš and A. Steed, "Revision control database for 3d assets," 2012.

[10] J. D. Denning and F. Pellacini, "Meshgit: Diffing and merging meshes for polygonal modeling," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 1–35, Jul. 2013.

[11] J. D. Denning, "ModFlows: Methods for Studying and Managing Mesh Editing Workflows ," Ph.D. dissertation, Dartmouth College, Computer Science, Hanover, NH, June 2014.

[12] J. Doboš, N. J. Mitra, and A. Steed, "3d timeline: Reverse engineering of a part-based provenance from consecutive 3d models," *Computer Graphics Forum (Special issue of Eurographics 2014)*, 2014.

[13] J. Doboš, "Management and visualisation of non-linear history of polygonal 3d models," Ph.D. dissertation, UCL (University College London), 2015.

[14] "3d repo: 3d version control system," http://3drepo.org/, 2014, online; acessed 2014-02-1.

[15] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkiewicz, and D. Dobkin, "Modeling by example," *ACM Transactions on Graphics (Proc. SIGGRAPH)*, Aug. 2004.

[16] Y. Zheng, D. Cohen-Or, M. Averkiou, and N. J. Mitra, "Recurring part arrangements in shape collections," *Computer Graphics Forum*, 2014.

[17] E. Sink, *Version Control by Example*. Pyrenean Gold Press, 2011.

[18] J. Corbet, A. Rubini, and G. Kroah-Hartman, *Linux Device Drivers, 3rd Edition*. O'Reilly Media, Inc., 2005.

[19] "Maya autodesk," http://usa.autodesk.com, 2014, online; acessed 2014-02-1.

[20] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.

[21] H. J. Wolfson and I. Rigoutsos, "Geometric hashing: An overview," *IEEE Comput. Sci. Eng.*, vol. 4, no. 4, pp. 10–21, Oct. 1997. [Online]. Available: http://dx.doi.org/10.1109/99.641604

[22] R. P. I. I. P. Laboratory and D. Meagher, *Octree Encoding: a New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer*, 1980. [Online]. Available: https://books.google.com.br/books?id=CgRPOAAACAAJ

[23] C.-T. Chang, B. Gorissen, and S. Melchior, "Fast oriented bounding box optimization on the rotation group so(3,$\mathbb{R}$)," *ACM Trans. Graph.*, vol. 30, no. 5, pp. 122:1–122:16, Oct. 2011.

[24] R. Yokota and L. A. Barba, "Hierarchical subdivision of space in FMM (fast multipole method)," 03 2012. [Online]. Available: http://dx.doi.org/10.6084/m9.figshare.91440

[25] J. D. Denning, V. Tibaldo, and F. Pellacini, "3dflow: Continuous summarization of mesh editing workflows," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 140:1–140:9, Jul. 2015. [Online]. Available: http://doi.acm.org/10.1145/2766936