Abstract

We present an open source tool, 3D Diff, that supports differencing and merging of 3D models. As modeling software grows in use, and as 3D models get more complex and require input from more users, there is an emerging problem of maintaining large scenes over time. A scene might be concurrently edited by different users and thus there is a need to merge different versions of a 3D model. We frame this problem in a way that is analogous to software merging: firstly, we automatically detect differences in 3D models by noting correspondences and discrepancies between them; secondly we provide an interactive tool to select between such changes in order to effect a merge. To achieve this we introduce the notions of explicit and implicit conflicts in 3D models and develop a prototype implementation to support the differencing and merging processes. We evaluate this tool with users and find the 3D Diff to be an effective way of merging 3D models. We claim that such tools have an important role to play in the maintenance of large models.


Keywords: 3D diff, conflict resolution, merging, indirect conflict

1 Introduction

In the past few years, as computers have become more capable in the scales and richness of graphics that they support, the complexity and size of 3D models as well as the amount of individuals collaborating on the same scene has grown dramatically. This explosion in content creation is easily witnessed by the large numbers of models available in repositories such as Trimble 3D Warehouse or TurboSquid. Thus, the emerging problem is that of maintaining a set of 3D models over multiple revisions and artists. Popular modeling packages such as Autodesk 3ds Max, Maya or Blender do not natively aid the merging process and usually only provide a means of manually resolving conflicting edits. We, therefore, frame the problem of maintenance of models through concurrent edits by comparing it to the analogous situation in software engineering. We use the notions of 3D differencing and 3D merging [Doboš and Steed 2012], that describe identifying and resolving discrepancies between two or more input models. However, it is not always possible to construct a consistent model by only selecting parts of the input models because scene edits might interfere with each other. This is comparable to software merging where a naive approach may not preserve the semantics or structure of the source code. We thus identify explicit conflicts where the corresponding scene component in different models has been edited differently, and implicit conflicts that are caused by the side-effects of the merging itself due to introduced bounding box interference.

Our 3D Diff tool performs 2-way or 3-way differencing and merging between 3D models, in a manner analogous to software merging [Mens 2002]. We take two 3D models and, optionally, their common ancestor, and compare them by their scene graph representations (see §3.1) down to the individual differences at the scene node level. 3D Diff then provides functionality for visualizing and resolving conflicts by allowing the user to select which parts of the 3D models to retain from which input, Fig. 1. The output may not always be completely resolvable, hence if vertex-level editing is required, we expect this to take place in a standard editing package. The two main contributions of our work are in identifying the problems of 3D differencing and merging and in presenting a prototype tool that supports interactive merging of two models. In addition, we develop visualization modes for both types of differencing, and perform a formative user study that indicates that users were able to merge models efficiently and were able to exploit the advantage of 3-way over 2-way diff including our implicit conflicts detection.

2 Related Work

The management of versions of documents presents perennial challenges in software engineering. Most individual programmers and teams use some form of version control system (VCS) such as Subversion, Git, PerForce, etc. Most VCSs support distributed access which opens up the risk of different copies of the files being edited in conflicting ways. Modern integrated development environments (IDEs) include a visual differencing tool, e.g. FileMerge for XCode or WinDiff for Visual Studio. These visualize the differences, but also create a new file that integrates changes and allow editing where a coherent merge cannot be automatically formed.

Surveys of software merging [Mens 2002; Altmanninger et al. 2009] present several concepts applicable to merging of 3D models.

1. Pessimistic version control avoids conflicts by locking which is unsuitable if parts of 3D models are reused across the scene. In contrast, optimistic version control requires conflict resolution.
2. Merging support is either state-based (only the state at the time of merging is considered), change-based (all individually performed changes are considered) or operation-based (changes can be re-run and interleaved in editor, e.g. [Chen et al. 2011]).

3. There is a difference between textual, syntactic, semantic and structural changes. Most difference tools use only textual changes in a line-by-line fashion [Hunt et al. 1998] (similar to the original diff algorithm, [Hunt and McIlroy 1976]). This works well for code, less so for highly structured or binary data.

Gleicher et al. [2011] identify three main strategies for visualizing differences: juxtaposition (side-by-side), superposition (overlay) and explicit encodings (e.g. time warp or subtractions). Superposition is known to be superior for visualizing more structured data such as trees [Munzner et al. 2003].

Tool Support A system for 2D images by Chen et al. [2011] is operation-based and thus records user actions in a revision tree so that they can be replayed via a revision tree. The merge user interface shows two revision images and a preview of the merge result. MeshFlow system [Denning et al. 2011] similarly records actions in Blender editor and visualizes single mesh edits. Whilst this is a viable approach, we opted for a state-based system to support a variety of 3D editing tools.

A BMergeAB 3D Merging Once all the changes have been determined, automated merge suggestions can be proposed to the user. The output is a single 3D model with some of the conflicts resolved by taking an entire change from one of the two versions. If not satisfied with this option, conflicting edits can be left unmodified and resolved in a third-party editor.

Whilst node-to-node correspondences could be estimated automatically (§2), tools to do this are not readily available. We therefore rely on matching universally unique identifier (UUID) assigned to each scene component as done in [Doboˇs and Steed 2012]. We also assume that if a node has been modified, all of its instances get affected equally. If this was not desirable behavior, the modeller would have split the instances in one of the versions already.

3.1 3D Differencing To determine changes across 3D models, we compare scene nodes in a pair-wise manner and define their state as depicted in Tab. 1. On one hand, a direct conflict on a scene node arises if it exists in both models, but is not equal. This equality and its granularity is implementation dependent, see §4. In our prototype, if a single vertex has been deleted or repositioned, that entire mesh node would be considered as different. On the other hand, an indirect conflict arises when the semantics of a 3D object are violated such as in Fig. 2. It is reasonable to assume that any bounding box intersections in a single input model are intentional. However, when merging changes from multiple models, new intersections might be introduced that do not exist before. Even though not necessarily conflicts, they should be flagged for user attention. Hence, suppose that two meshes n and k both present in two distinct models A and B have been independently modified but did not have a bounding box intersection. If in their merged result Merge AB they do have a bounding box intersection, we consider this an indirect conflict. Therefore, an indirect conflict occurs whenever the following holds:
2-way Diff  Let A and B be sets of scene graph nodes in two 3D models and n a single corresponding node whose content can change across versions of a 3D model. Hence, for a 2-way diff the only possible states of a component n are:

\[
\text{unmodified} \iff (n_A = n_B \land n_A \neq \emptyset \land n_B \neq \emptyset),
\]

\[
\text{added} \iff (n_A = \emptyset \land n_B = \emptyset) \lor (n_A = \emptyset \land n_B \neq \emptyset),
\]

\[
\text{deleted} \iff (n_A \neq \emptyset \land n_B = \emptyset),
\]

\[
\text{modified} \iff (n_A \neq \emptyset \land n_A \neq \emptyset \land n_B \neq \emptyset),
\]

\[
\text{conflicted} \iff (n_A \neq \emptyset \land n_A \neq \emptyset \land n_B \neq \emptyset).
\]

where operator \(\square(m_M)\) determines a bounding box of a component m in a model version M. As this does not rely on the common ancestor, it is equally applicable to 2-way and 3-way differencing.

3-way Diff  In addition, let \(\Lambda n\) be the common ancestor set of scene node components of A and B. This extra knowledge of the original state informs the differencing process and can automatically resolve otherwise ambiguous cases. Hence, the possible states of a component n due to the extra information are:

\[
\text{unmodified} \iff (n_A = n_B \land n_A \neq \emptyset \land n_B \neq \emptyset),
\]

\[
\text{added} \iff (n_A = \emptyset \land n_B = \emptyset) \lor (n_A = \emptyset \land n_B \neq \emptyset),
\]

\[
\text{deleted} \iff (n_A \neq \emptyset \land n_B = \emptyset),
\]

\[
\text{modified} \iff (n_A \neq \emptyset \land n_A \neq \emptyset \land n_B \neq \emptyset),
\]

\[
\text{conflicted} \iff (n_A \neq \emptyset \land n_A \neq \emptyset \land n_B \neq \emptyset),
\]

where \(n_A \in A\) and \(n_B \in B\). Even if files’ timestamps were taken into account, they do not necessarily guarantee a temporal ordering of changes. Hence, any discrepancy in the state of the two corresponding components is considered unresolvable. Similarly, a state of a node that is present in one of the versions but not the other is ambiguous as it could have been equally likely added or deleted.

### Table 1: Schematic representation of a 2-way vs. a 3-way diff symmetric for \(n_A\) and \(n_B\). Red states cannot be automatically resolved.

<table>
<thead>
<tr>
<th>2-way</th>
<th>3-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n_A)</td>
<td>(n_B)</td>
</tr>
<tr>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

### Figure 3: Overlay visualization (left) shows the merge window. Smart visualization (right) shows two versions that are being differentiated and a large merge. Notice how the additional information about a common ancestor in a 3-way diff resolves ambiguities.

4 Implementation

Our stand-alone client with a graphical user interface (GUI) is written in C++ and a cross-platform UI framework Qt (Fig. 3). The Open Asset Import Library (Assimp) converts common 3D file formats into a unified in-memory scene graph representation making our solution independent from any particular modeling package.

If a node with the same UUID is not present in both models, the equality does not hold. If it exists in both, we further perform an early reject comparison. Currently, our implementation considers only changes in meshes but is easily extensible to any scene graph components such as transformations, materials, shaders, textures etc. For meshes, we perform a shallow in-memory byte-by-byte comparison on the vertex, face and normal arrays which is a reasonably fast operation and compare the connections to other nodes. If there are any differences, the meshes are labelled as different. This allows us to compare models with varying scene graph topologies.

To abstract from the underlying differencing techniques, in a 3-way diff we hide the ancestral model, making the interface identical to a 2-way diff. Therefore, for both the user can select one of overlay (single merge window), standard (two differenced models plus a larger merge window) and smart (the same as standard with added indirect conflict detection) visualizations. Differences in the models are shown as color coded highlights. A merge table lists these differences and tick boxes enable a selection of individual versions to be preserved. In addition, the user has the option to link the scene navigation across the views. An automated camera further reframes the viewport to make a selected mesh the main focus of attention.
Figure 4: Questionnaire sample averages. $T_1$ to $T_4$ correspond to an increasing level of software support for merge suggestions.

5 User Study

In order to determine the best approach to 3D differencing and the level of users' trust in the automated suggestions, we ran a study with 8 PhD students in computer graphics and vision. We tested 3D merging with an increasing level of software assistance in 4 trials; $T_1$: 2-way diff overlay, $T_2$: 2-way diff standard, $T_3$: 3-way diff standard, $T_4$: 3-way diff smart. The task was to merge pre-made changes such that the most recent were preserved while not introducing new indirect conflicts. If not resolvable, the changes were to be left in a conflicted state. After each trial, the participants filled in an electronic questionnaire the results of which are in the supplemental materials. Each model set $M_i$ to $M_4$ consisted of a sample model, small industrial model with less than hundred meshes and a large city model with more than hundred distinct meshes. The order of tasks and models was shuffled according to the Latin square.

Evaluation Our participants were able to use the tool successfully even though they varied in what they considered to be a successful merge, some conservatively discarding changes whilst others reverting additions or deletions that were automatically flagged, for aesthetic reasons. Therefore, unlike software merging, 3D merging is a subjective judgement. Hence, the future tools should not assume that the chronology of edits is the primary selection mechanism but focus on the aesthetics. The positive take away is that the participants were able to express the results easily and quickly.

The highest scores (Fig. 4) for the ease of use, reliability and confidence were achieved by a standard 3-way diff method, although, on average, the participants trusted the automated suggestions more with the increasing level of software support, as expected. Based on the written comments we believe that the indirect bounding box highlighting is a useful addition, although large boxes would indicate a conflict despite no physical intersection with another mesh. As one of the participants wrote: “The conflicted BB [bounding box] is useful but needs to be calculated at lower levels of meshes.”

6 Conclusions and Future Work

We made analogies to software engineering where a vast range of research on merging has been done. We identified that textual changes on models were unlikely to work and thus we chose the scene graph abstraction as a way of structuring the scene for change detection. This automatically generates syntactically and semantically correct output as a 3D model. We further discussed the indirect conflicts as violations of semantic meaning. We built a GUI to realize these and through a user study demonstrated it to be useful.

In the future it would be easy to extend our implementation to take into account modifications of other types of scene components apart from meshes. Existing image-based differencing techniques can be combined with the approach presented here to provide a truly robust solution to 3D asset management. In addition, based on the established mesh differences, it is possible to further compare individual vertices and find only the portions of meshes that have been modified. In the near future we plan to extend this work to search random collections of unrelated 3D models to find the differences.

References


