Towards a Collaborative Editing System on 3D Space

Tasuku Takahashi¹, Kengo Imae¹, and Naohiro Hayashibara²

¹ Graduate School of Frontier Informatics, Kyoto Sangyo University, Kyoto, Japan
{1i558129,1i558048}@cc.kyoto-su.ac.jp
² Faculty of Computer Science and Engineering, Kyoto Sangyo University, Kyoto, Japan
naohaya@cc.kyoto-su.ac.jp

Abstract. Collaborative editing is one of the fundamental problems in distributed systems. It includes a consistent update on shared data. Many approaches to this issue have been proposed, but most of them are for two-dimensional data such as documents, pictures, etc. In this paper, we extend the notion of ChainVoxel which is based on conflict-free replicated data types for a practical collaborative editing system for three-dimensional space.

1 Introduction

Consistency maintenance is one of the fundamental problems for building a data sharing service in distributed systems. Real-time collaborative editing in distributed systems has been discussed in many papers [4,9,10,12,13] in terms of consistent updates. They allow a group of users to edit and view the same document, picture or 3D model concurrently. One of the simple solutions for consistency maintenance on shared objects is to use mutual exclusion algorithms. However, mutual exclusion involves a risk of a system-wide bottleneck in exchange messages for data consistency.

Operational transformation approaches have been used in decentralized collaborative editing systems [3,9]. These approaches are a kind of optimistic concurrency control methods without mutual exclusion. Local operations in every site are executed immediately, and they are sent to remote sites. Remote operations are transformed before execution to resolve conflicts. They are scalable approaches for consistency maintenance regarding algorithms.

Shapiro et al. proposed the notion of the conflict-free replicated data type (CRDT) for consistent updates in distributed systems [6,11]. It is also a kind of optimistic concurrency control approaches regarding data structure and can maintain eventual consistency of shared data. This approach assumes a data structure and commutative operations on it. Thus, every site executes local and remote operations without any transformation of a sequence of operations and mutual exclusion. As a result, the approach improves the scalability in large-scale distributed systems such as cloud systems.

© Springer International Publishing AG 2018
L. Barolli et al. (eds.), Advances on Broad-Band Wireless Computing, Communication and Applications, Lecture Notes on Data Engineering and Communications Technologies 12, https://doi.org/10.1007/978-3-319-69811-3_58
Towards a Collaborative Editing System on 3D Space

There are several CRDT-based systems for shared documents such as Telex [2], Treedoc [10], a collaborative editing in MANET [4], bounded counter [1], and eventually consistent register [14].

Now, we focus on a real-time collaborative editing for 3D models. Various applications which use 3D models (e.g., virtual reality, distance learning, city planning [8], etc.) have been developed in recent years. Imae and Hayashibara proposed a conflict-free data type for 3D models, called ChainVoxel [5]. It guarantees the eventual consistency property of a 3D space without a mutual exclusion mechanism. Thus, it solves the performance bottleneck caused by the mechanism for a consistent update of collaborative editing systems with lots of users.

It is a simple mechanism, which has only two operations; insert and delete. However, most of 3D models have some logical structure. For instance, an anatomical model of the human body consists of a head, a torso, arms, and legs. They should be constructed by several voxels because the size of them is not the same. ChainVoxel can not be used for such a case.

In this paper, we propose a layered structure of ChainVoxel to extend its notion for a practical collaborative editing for shared 3D models. We also propose the upper layer of ChainVoxel and three operations on it associated with groups of voxels.

2 Related Work

We introduce some of related work on collaborative editing.

For realizing real-time collaborative editing systems for documents, lots of algorithms and data structure have been proposed and discussed so far.

The operational transformation approach is a major solution for consistency maintenance. Sun et al. proposed the causality preservation transformations: the inclusion transformation and the exclusion transformation [12] for collaborative editing of documents.

In SOCT algorithm proposed by Vidot et al. [13], operations are ordered globally using a timestamp given by a sequencer. The sequencer generates positive integer values which monotonically increase. As a result, these operations are delivered on each site in the same order based on timestamps.

State difference based transformation (SDT) approach proposed by Li and Li [7] guarantees the convergence in arbitrary transformation paths. SDT is the first operation transformation algorithm for collaborative editors in peer-to-peer systems. The worst-case on time complexity is $O(n^3)$ and the average on it is $O(n^2)$.

In contrast, Shapiro et al. proposed a conflict-free replicated data type called CRDT for real-time collaborative editing [11]. It is an optimistic concurrency control approach based on a commutative data structure. For this reason, a shared data is ensured to converge a consistent state among sites when operations in multiple sites occur asynchronously. Letia et al. gave an example of a CRDT-based system and showed its performance and scalability [6].
3 ChainVoxel

ChainVoxel [5] proposed by Imae and Hayashibara is a data structure based on CRDT [11] represented by a chained hash structure for collaborative editing for 3D models. It makes a sequence of operations commute and guarantees eventual consistency of shared data among sites. A site (e.g., computer) is assigned unique identifier (sid), siteId = \{id ∈ ℤ|id ≥ 0\} is represented by an integer.

3.1 Voxels

In the original definition of CRDT, an atom is defined as a minimum unit in a document which is shared and modified by multiple users. On the other hand, a voxel is equivalent to an atom in ChainVoxel since we assume that a 3D model consists of a set of voxels.

Each voxel has sid and a timestamp ts and is represented as voxeltsid. A timestamp ts ∈ ℤ is represented by an integer value like POSIX time, but it can be a negative integer value. The latest timestamp is the biggest value among existing timestamps.

Then, we consider a location of a voxel in a 3D space. We assume a 3D space where each position is shown as a tuple \((x, y, z) \in PI \subset ℤ^3\), called posID and shared 3D models are located in it. Thus, a position of each voxel is represented as posID and a voxel is a cube of size 1. posID is associated with a mapping \(V \xrightarrow{f_{pos}} PI\) where \(V\) is a set of voxels allocated in the space and \(PI\) is a set of posIDs of voxels. Note that more than two different voxels in the same position cause contention and only one of them can appear in the space. It means that ChainVoxel plays an important role to resolve contention of voxels in the same location.

3.2 Hash Structure of ChainVoxel

ChainVoxel represents voxels in a chained hash structure, which holds ordered sets of voxels corresponding to posID (see Fig. 1). ChainVoxel is generated for each shared space; thus, it includes an identifier of a shared space.

The purpose of managing the operations as ordered sets of voxels is to avoid contention of voxels. These voxels are sorted according to ts in ascending order. However, multiple sites can execute operations at the same position simultaneously. We call this situation a collision.

In the case of collisions, there are several possible solutions such as the intention preservation ordering [12]. To make the explanation of the proposed data structure simple, we suppose voxels are ordered by sid in ascending order when a collision occurs. It is quite a simple solution, but it does not impose any impact (i.e., overhead) on the performance of the collaborative editing system.

As we stated above, voxels allocated in the 3D space are stored as ordered sets by its position (see Fig. 1). We call the head voxel of each chain, the primary voxel, and the others are known as sub-voxels.
A primary voxel is the voxel corresponding to the head in the chain of voxels for each posID. In other words, the oldest voxel is treated as a primary voxel. All other voxels are classified in sub-voxels. Primary voxels are only voxels that are displayed in the shared 3D space. The negativeVoxel in Fig. 1 is a special voxel which is added by a delete operation. It is like a tombstone to remove voxels. The next voxel to negativeVoxel becomes a primary voxel in the case that negativeVoxel exists in the chain.

Voxels are inserted by sites asynchronously. By doing so, the shape of a shared 3D model might be different in each site. However, the chain of voxels at each position is rearranged in total order. It means that voxels in ChainVoxel in every site eventually become consistent.

### 3.3 Operations on ChainVoxel

We now describe about operations on ChainVoxel. In this paper, we define two operations insert and delete. Each operation is identified by sid and ts. We show these operations in Algorithms 1 and 2.

#### 3.3.1 Insert Operation

The insert operation requires sid, ts and posID. sid is the identifier of the site. We assume that there is a function which can get current time like the time function in C. ts is obtained by the function. posID is the value obtained by user’s input. This operation substitutes the tuple (sid, ts, posID) for insertVoxel which has been initialized. We introduce a special voxel called negativeVoxel including sid, ts and posID. It is added by the delete operation for the first time. It is at most one in a chain of voxels for each posID. An insert operation checks ts of negativeVoxel by using the function $f_{ts}$ if negativeVoxel already exists at posID. An insert operation fails in the case of $f_{ts}(\text{negativeVoxel}) \geq ts$. 

![Fig. 1. Hash structure of ChainVoxel](image)
3.3.2 Delete Operation

Semantically speaking, the delete operation removes voxels which have been inserted at a certain \( \text{posID} \). By contrast, it puts a particular voxel \( \text{negativeVoxel} \) at the end of the current chain of voxels. It is also called a tombstone and means that voxels in front of the negativeVoxel are no longer valid. This operation adds \( \text{negativeVoxel} \) for the first time and just updates \( \text{ts} \) for the second time or later (see line 5 in Algorithm 2). It means that \( \text{negativeVoxel} \) exists at most one in a chain of voxels for each \( \text{posID} \). Note that \( \text{sid} \) is a positive integer but is assigned a negative value for \( \text{negativeVoxel} \). We assume a garbage collection mechanism to remove voxels, which are not valid, from the chains of voxels.

So, operations (i.e., insert and delete) which have been executed at a site \( p \) are sent to all sites in \( S \) except \( p \). These operations are called local operations at \( p \) and remote operations at a site \( q \) in \( S \setminus \{p\} \). Suppose an operation \( \text{insert}_{p}^{ts} \). It is a local operation on \( p \) while it is a remote operation on \( q \) because \( \text{sid} \) is different from \( q \).

Local operations are immediately executed at every site. On the other hand, remote operations are executed as soon as they are delivered.

Algorithm 1. Insert operation

1: Inputs:
   \( \text{sid} \leftarrow \text{own site ID} \)
   \( \text{ts} \leftarrow \text{current time ts}_{\text{now}} \)
   \( \text{posID} \leftarrow \text{the given position from users} \)
2: Initialize:
   \( \text{insertVoxel} \leftarrow \bot \)
3: \( \text{insertVoxel} \leftarrow (\text{sid}, \text{ts}, \text{posID}) \)
4: if \( \text{negativeVoxel} \in \text{V}_{\text{posID}} \) then
5:   if \( f_{\text{ts}}(\text{negativeVoxel}) \geq \text{ts} \) then
6:      return /* do nothing */
7:   end if
8: end if
9: \( \text{V}_{\text{posID}} \cup \{\text{insertVoxel}\} \)
10: broadcast \( \text{insert}(\text{sid}, \text{ts}, \text{posID}) \) to every other sites

4 Hierarchical Structure of ChainVoxel

The notion of ChainVoxel [5] is a very simple, but it is not practical because it has only two operations (i.e., add and delete) for a voxel. We now extend the notion to handle a 3D model consists of multiple voxels.

A 3D model or a part of it has some logical structure, for instance, there are head, torso, arms, and legs in an anatomical model of the human body. They may be represented by multiple voxels in ChainVoxel. It means that ChainVoxel should handle a group of voxels.
Algorithm 2. Delete operation

1: Inputs:
   
   \( \textit{sid} \leftarrow \text{a negative integer} \)
   \( \textit{ts} \leftarrow \text{current time } t_{\text{now}} \)
   \( \textit{posID} \leftarrow \text{the given position from users} \)

2: Initialize:
   
   \( \textit{negativeVoxel} \leftarrow \perp \)

3: if \( \textit{negativeVoxel} \in V_{\textit{posID}} \) then
4:   if \( f_{ts}(\textit{negativeVoxel}) < \textit{ts} \) then
5:     update(\( \textit{negativeVoxel}, \textit{ts} \)) /* update the timestamp */
6:   end if
7: else
8:   \( \textit{negativeVoxel} \leftarrow (\textit{ts}, \textit{posID}) \)
9:   \( V_{\textit{posID}} \cup \{\textit{negativeVoxel}\} /* \text{add a } \textit{negativeVoxel} */ \)
10: end if
11: broadcast delete(\( \textit{ts}, \textit{posID} \)) to every other sites

4.1 The Hierarchical Structure Model

To extend the existing ChainVoxel, we introduce a hierarchical structure. Now we define two layers shown in Fig. 2. The primary layer is for basic operations of ChainVoxels (i.e., insert and delete). The grouping layer is for operations on groups of voxels.

There is the total order of priority among layers. An upper layer has higher priority than a lower one. In this case, the grouping layer takes priority over the primary layer. When a group is modified by using operations on the grouping layer, the primary layer is completely locked. It means that operations on the primary layer are prohibited.

![Fig. 2. Hierarchical structure of ChainVoxel](image-url)
As shown in Fig. 2, the primary layer has two operations associated with voxels; insert and delete, and the grouping layer has three operations associated with groups; create, join and leave.

4.2 Group of Voxels

To operate a set of voxels, we define a group of voxels. We use such groups to express a logical structure of 3D models. First, we define a universal set of groups \( U = \{ G_{gid}^{ts_i}, G_{gid}^{ts_j}, \ldots \} \). Each group in \( U \) satisfies the following conditions.

- It is defined as a set of voxels \( G_{gid}^{ts} = \{ V_{pos1D_1}, V_{pos1D_2}, \ldots \} \).
- It has a unique identifier \( gid \in GI \) and a timestamp that shows the time at which it has been created.
- It allows overlapping with another group, in a word, they may satisfy \( G_i \cap G_j \neq \emptyset (i \neq j) \).

There is a group lookup table, which shows groups that certain voxels belong to.

A group \( G_{gid}^{ts} = \emptyset \) will be deleted from \( U \) after a given time \( \Delta U \) elapsed. In the grouping layer, we express a logical structure of 3D models which consists of voxels.

4.3 Operations on the Grouping Layer

We now define three operations associated with groups of voxels on the grouping layer.

4.3.1 Create Operation

It can create a new group of voxels which have been already inserted. It adds a unique identifier (\( gid \)) to the group. The created group by this operation is initially empty. It can be added and removed voxels by using join and leave operations mentioned below. The procedure of the create operation is described in Algorithm 3.

**Algorithm 3.** Create operation

1: if \( G_{gid} \notin U \) then
2: \( \text{obtain } gid \notin GI \)
3: \( U' \leftarrow U' = U \cup \{ G_{gid} \} \)
4: end if

The following condition has to be satisfied after the operation executed.

\[
G_{gid} \in U' \land G_{gid} = \emptyset \land U \subset U'
\]
4.3.2 Join Operation
It adds voxels into a group created by the create operation mentioned above.

We now define a function $f_{\text{state}} : PI \times GI \mapsto \mathbb{Z}$ to obtain the status of a voxel $V_{posID}$ in a group $G_{gid}$.

$$f_{\text{state}}(posID, gid) \begin{cases} 
0 & \text{if } V_{posID} \not\in G_{gid} \\
\text{fts}(G_{gid}(posID)) & \text{otherwise}
\end{cases}$$

The function $\text{fts}(G_{gid}(posID))$ obtains the timestamp of $V_{posID} \in G_{gid}$. The function $f_{\text{state}}$ returns the latest timestamp of $V_{posID}$ if $V_{posID} \in G_{gid}$, otherwise it returns 0.

Join operation requires the current time $ts_{now}$, $posID$ that shows the location of the voxel $V_{posID}$ that is added, and $gid$ that represents the group to which $V_{posID}$ is added. It is described in Algorithm 4.

**Algorithm 4. Join operation**

1: Inputs:
   $V_{posID}$ ⊳ a voxel to be added.
   $gid$ ⊳ a group that the voxel is added to.
2: if $G_{gid} \in U \land |f_{\text{state}}(posID, gid)| < ts_{now}$ then
3:   $G'_{gid} = G_{gid} \cup \{V_{posID}\}$
4:   update $ts$ of $V_{posID} \in G_{gid}$ as $\max(ts_{now}, f_{\text{state}}(posID, gid))$
5: end if

This operation prevents duplicated groups by the precondition. Moreover, it can converge to the latest join operation by checking the timestamp of voxels.

4.3.3 Leave Operation
It is executed for deleting a voxel $V_{posID}$ from a group $G_{gid}$. The procedure of the leave operation is described in Algorithm 5.

**Algorithm 5. Leave operation**

1: Inputs:
   $V_{posID}$ ⊳ a voxel to be added.
   $gid$ ⊳ a group that the voxel is added to.
2: if $V_{posID} \in G_{gid} \land |f_{\text{state}}(posID, gid)| < ts_{now}$ then
3:   $G'_{gid} = G_{gid} \setminus \{V_{posID}\}$
4:   update $ts$ of $V_{posID} \in G_{gid}$ as $\min(-ts_{now}, f_{\text{state}}(posID, gid))$
5: end if

It removes $V_{posID}$ from $G_{gid}$, updates the timestamp for $V_{posID}$, and finally executes the insert operation on the primary layer. Note that the timestamp is
updated as a negative value. It means that the voxel is not deleted immediately. It plays a role of a tombstone which intends a deleted voxel by someone. We assume a garbage collection mechanism such that voxels, each of which has a negative integer as its timestamp, will be removed from the storage after a certain period.

In the multi-users environment, it is possible that the voxel $V_{posID}$ has been already deleted by someone. For this reason, the join operation executes the insert operation on the primary layer to guarantee that the voxel remains in the 3D space.

5 Conclusion and Future Work

We proposed a hierarchical structure to extend the notion of ChainVoxel. We also proposed three operations associated with groups of voxels on the grouping layer on top of the primary layer. The proposed approach enables to handle groups of voxels for expressing a logical structure of a 3D model.

The hierarchical structure can extend ChainVoxel depending on an application. For example, you can add a layer associated with annotation if you want to add some annotation on a 3D model.

We are now implementing a prototype of the proposed layered structure of ChainVoxel. So, we evaluate the prototype regarding throughput, latency and convergence time for maintaining consistency in the future.

Acknowledgement. This work was supported by JSPS KAKENHI Grant Number JP16K00449.

References