A Robust Collaborative 3D Editing Tool Utilizing Distributed Consensus Protocol

Yeoun-Ui Ha¹, Jae-Hwan Jin¹ and Myung-Joon Lee²

Department of Electrical/Electronic and Computer Engineering, University of Ulsan, 93, Daehak-ro, Nam-gu, Ulsan 680-749, Republic of Korea
¹{gkdus23, jjhok2000}@gmail.com, ²mjlee@ulsan.ac.kr

Abstract. Many 3D editors have been developed as 3D modeling has been popular. Naturally, collaboration support for 3D editing has appeared in those editors. As one of such editors, Lets3D singularly enables a group of users to edit 3D scenes in real time. In this paper, we present a different version of the Lets3D editor named Lets3D-C, which does not require the centralized middlewares for collaboration. Lets3D-C adopts distributed consensus model based on the raft protocol, ensuring the robustness of collaboration in case of network failure and providing the similar functionalities for collaborative 3D editing.

Keywords: Lets3D, Consensus algorithm, Raft, 3D collaborative editor

1 Introduction

Recently, 3D-related user activity has become richer and many useful 3D editors such as blender[1], FreeCAD[2] and Google’s sketch up[3] are being used popularly. Since collaboration in 3D modeling is also getting more attention, collaborative environments for 3D modeling [4][5] are provided as commercial Web services and the Lets3D [5] editor is introduced as an extension of Three.js with the functionality of collaborative real-time editing and sharing of 3D scenes for a group of users.

In this paper, we present a different version of the Lets3D editor named Lets3D-C, which does not require the centralized coordination for collaborative editing. Lets3D relies on the whiteboard service of C3ware[6] to provide collaborative functionality and uses the Openfire XMPP messaging system to create and maintain user/group information, which might be sources of the single point of failure. To address this issue, Lets3D-C adopts distributed consensus model based on the Raft[7] protocol, which is recently introduced as a consensus protocol easy to understand in comparison with the famous Paxos [8] protocol. Lets3D-C internally uses Skiff [9] which is an open source JavaScript implementation of the raft protocol, ensuring the robustness of collaboration in case of network failure or node failure. Through Lets3D-C, collaborative editing can progress when a majority of participating users are available. As in Lets3D, participating users share their editing screen through Web browsers in Lets3D-C with being aware of which objects are being edited by which...
users. Any editing operations are coordinated by the distributed locking mechanism supported in Lets3D-C. Unlike Lets3D, the users to participate in collaborative editing are invited by a conventional email.

2 System Structure

A Raft cluster is composed of multiple nodes, which are supposed to achieve consensus via a distinguished leader. Each node in the cluster acts as one of the three roles: leader, follower or candidate. The leader plays a crucial role both in handling requests to the cluster and in log replication. In case that the leader fails or becomes disconnected from other nodes, the leader election process is triggered as described in Fig. 2. To achieve consensus, the following tasks are performed in the cluster:

**Leader election:** The leader periodically sends a heartbeat message to followers for testing connection. If there is no heartbeat message within timeout (each follower has different waiting time), followers have an opportunity to become a leader as candidate. At this moment, it starts the election. When a candidate gets agreement from majority of servers, the candidate becomes a new leader.

**Log replication:** A log entry is committed only when the entry has been replicated on a majority of nodes in the cluster by the leader that created the entry.

![Role Transition in Raft Cluster](image)

Fig. 2. Role Transition in Raft Cluster

 Lets3D-C consists of two modules: editing module and consensus module. The consensus module in each Lets3D-C application participating in collaborative editing becomes a member of a Raft cluster as shown in Fig. 3.
3 Implementation

As is the case with the Three.js editor, the Lets3D-C editing screen is composed of 4 parts: Viewport, Menu bar, Side bar and Tool bar as shown in Fig.5.

Users perform the following procedure for collaborative 3D editing at Lets3D-C.

1. To form a collaborative editing group, a user sends the URL (IP, port number) of the consensus module to other users through email.

2. Through the URL, each invited member connects to the consensus server. When all members connecting the consensus module, a collaborative group is formed. The consensus modules of all members build a Raft cluster, being connected to each other.

3. Members of the group can edit 3D objects, sharing the editing screen as shown in Fig.6. With the help of distributed locking mechanism supported by the consensus modules in the cluster, any conflicting edit operations on the same objects are automatically coordinated.
4 Conclusion

Distributed consensus is fundamental and useful for building robust distributed applications even in the presence of the failures of some participating nodes. In this paper, we applied distributed consensus to collaborative 3D editing, presenting Lets3D-C as an outcome. For replication and synchronization of 3D data needed to participants in collaborative editing, Lets3D-C utilizes the Raft consensus module instead of XMPP messaging and the whiteboard service of C3ware in case of Lets3D. We believe that Lets3D-C and the associated techniques would be meaningful and useful for large-scale collaborative editing under widely distributed environment.

Acknowledgments. This research was partially supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. 2013R1A1A4A01004459)

References