卒 業 論 文

論文題目

3D モデリングシステムにおける同期編集機構の研究

指導教員

乃万 司　教授

九州工業大学情報工学部知能情報工学科

卒業年度　2016

学生番号　13231071

氏名

古城戸　隆志

題目

3D モデリングシステムにおける同期編集機構の研究

年度

2016

氏名

古城戸　隆志

卒 業 論 文

3D モデリングシステムにおける同期編集機構の研究

指導教員:　　乃万 司　教授

九州工業大学情報工学部
知能情報工学科

2016 年度

古城戸　隆志

目 次

第1章　はじめに

近年, Google ドキュメント [Google] や Microsoft Word Online[Microsoft] に代表されるテキスト編集を中心に, 同期編集システムの研究や開発が注目されている. 同期編集では, 同期する際に生じる編集の衝突を解消することが主な課題である. 編集の衝突を解消でき, 広く用いられ手法として操作変換がある. CoMaya[Agustina08] は操作変換を応用して 3D モデリングにおける同期を可能にした. しかし, CoMaya の操作変換は, 面と頂点, オブジェクトと面などの依存関係がある要素に対する同期は不可能である. そのため CoMaya ではオブジェクト単位の同期に留まっている. より柔軟に同期編集をするには, 頂点の操作も含めた 3 次元データを同期する必要がある. Differential Synchronization[Fraser09] は, 操作変換とは別の同期手法であり, 元々はテキスト編集を対象に開発されたが, 依存関係の扱いが操作変換より容易になると予想される.

本研究では, Differential Synchronization の同期手法に基づいて, 3 次元データの依存関係を考慮した同期編集機構の手法を提案する. これにより, 複数人で同一の 3 次元データを同時に編集できる 3D モデリングソフトの開発を可能にする.

本システムはオブジェクトが複数の面から構成されるサーフェスモデルを扱う. Differential Synchronization では, 同期のため, 接続されたクライアントごとに, クライアントとサーバに, データのシャドウコピーを作る. クライアントとサーバで, 差分の計算とその適用を行い同期を行う.

本システムの動作実験を行うため, サーバと 3 つのクライアントを用意し, クライアントのインターフェイスとして基本命令を実装した. また, クライアントごとに任意の基本命令 50 件をランダムなタイミングで発行した. その後, 各クライアントで同期されたデータを比較した結果, データはすべて一致しており, 同期可能であることを確認できた.

本論文の構成は次の通りである. まず, 第 2 章で本研究の背景と目的について述べる. 次に, 第 3 章でシステムで利用する手法について述べる. そして, 第 4 章で実験とその結果について述べ, 考察する. 最後に第 5 章で本研究のむすびとして今後の課題を述べる.

第2章　背景と目的

2.1　背景

近年, Google ドキュメントや Microsoft Word Online に代表されるテキスト編集を中心に, 同期編集システムの研究や開発が注目されている. 同期編集システムとは, 複数人で同一のデータを同時に編集できるシステムである. 複数人で行うため, 作業効率が上がることや, 他者が作っている部分を見ながら自分の作業している部分を修正できるメリットがある. また, 個々で作られたファイルをマージする手間も削減できる. 同期編集では, 複数のクライアントから様々な編集が行われるため, あるクライアントが最新ではないデータを編集する際に, 編集の衝突が起こる. 衝突が起こると各クライアントが扱うデータに矛盾が生じ, データの同期が失敗する可能性がある. データに矛盾が生じる例を図 2.1 に示す. 図 2.1 では, クライアント A の挿入命令をクライアント B が検知する前に, クライアント B が新たな挿入命令を送信している. よってお互いの命令を適用する際に, クライアント A が “red” を挿入した分の位置がずれてしまい, クライアント A とクライアント B の最終結果に矛盾が生じる. こういった衝突を解消することが同期編集の主な課題となっている. 衝突を解消するための同期機構にはいくつか選択肢があり, 後で変更することは困難なため, アプリケーション開発サイクルの早い段階で同期機構を選択する必要がある.

また, 3D モデリングにおける頂点や面などの, 3 次元データを扱う同期編集システムも研究や開発が行われている. 3 次元データを扱う際, 例えば, 面と頂点のデータ間にある依存関係を考慮しなければならない. 現在の研究では, オブジェクト単位の同期と, 依存関係のないデータのみの同期編集に留まっている. より柔軟に同期編集をするには, 依存関係を考慮し, 頂点の操作も含めた 3 次元データを同期する必要がある.

2.2　関連研究

2.2.1　Edit-based

Edit-based は, クライアントの全ての操作を他のクライアントに反映させる同期機構である. アルゴリズムとして操作変換 [Ellis89] を扱う. 操作変換は, ある編集操作のデータを, 以前に実行された編集操作の結果によって更新する仕組みである. これにより, 編集の衝突によるデータの矛盾を防ぐことができる. 図 2.2 は図 2.1 の矛盾を操作変換を用いて解消した例である. 図 2.1 では, クライアント A が “red” を挿入した分の位置のずれが, クライアント B の操作に対応していなかったが, 操作変換によって図 2.2 にあるように “red” の文字数分, 適用の際にクライアント B の命令のパラメータを更新する. これによりクライアント A とクライアント B の最終結果が一致する.

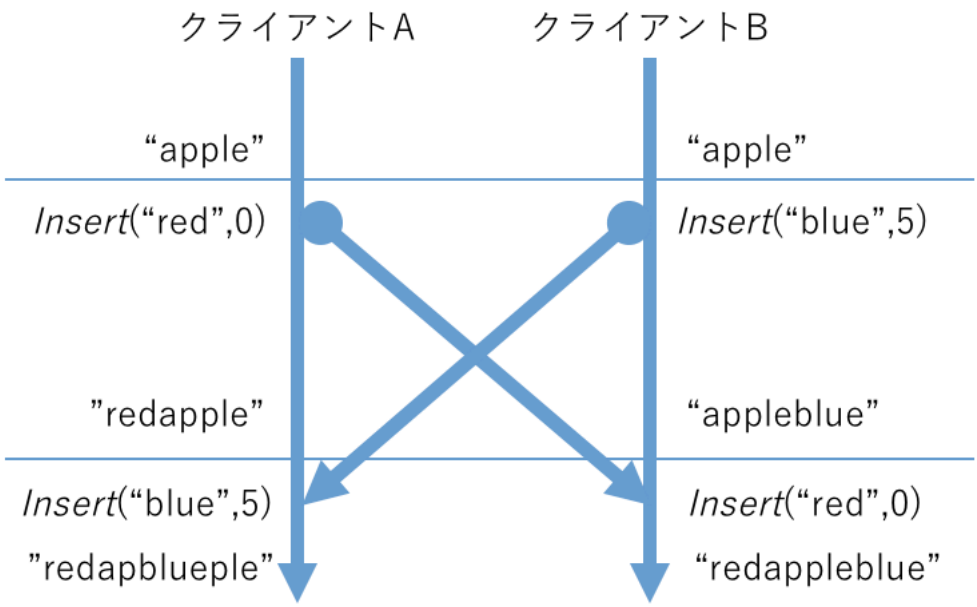

図 2.1 矛盾が生じる例

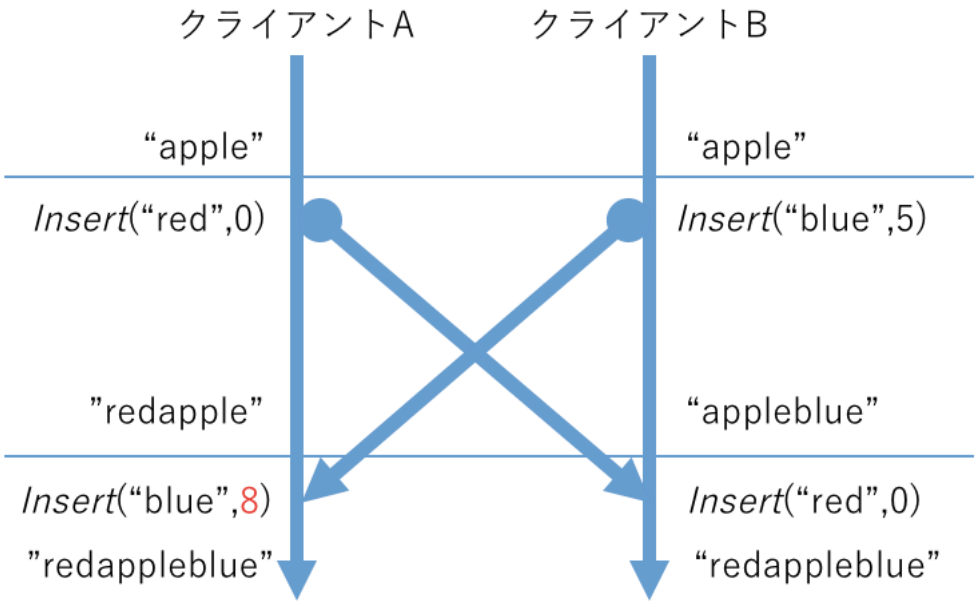

図 2.2 操作変換

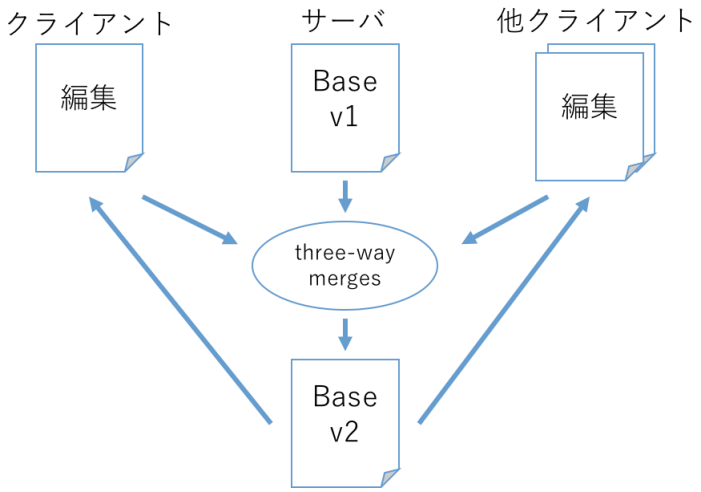

図 2.3 three-way merges

2.2.2 three-way merges

three-way merges[Lindholm04] は 3 つのデータを, サーバでマージし最新のファイルを作る同期手法である. three-way merges の流れの概略図を図 2.3 に示す. 図 2.3 のように, あるクライアントの編集と, そのクライアント以外のクライアントによる編集, さらにベースとなる元ファイルのデータを, サーバでマージし最新のファイルを作る. その最新のファイルをクライアントに送信し同期を行う. 各クライアントはサーバで作られた同一のファイルを受信するため, 同期後のファイルに各クライアント間での矛盾はない. しかし, 複数のクライアントがベースと本ファイルの同じ箇所を編集した場合, 編集の衝突が起こり, 各場合の, 衝突を回避する処理の実装が必要となる.

2.2.3 Differential Synchronization

Differential Synchronization は差分の計算とその適用によって同期を行う. 図 2.4 は Differential Synchronization の Dual Shadow Method の同期の流れである. また, 図 2.5 は Dual Shadow Method の他クライアントも含めた概要図である. Dual Shadow Method は, ClientText と ServerText の他に, クライアントとサーバに各データのシャドウコピーを用意する. 処理の流れとしては, まず ClientText と ClientShadow の差分を計算し編集としてサーバに送信する. 次に, クライアントではシャドウコピーを更新し, サーバでは受信した差分の適用処理を行う. サーバ側でも同様の手順で処理することで同期が可能となる. ServerText は複数のクライアントから利用され, ClientText と 2 つのシャドウコピーは, 接続しているクライアントごとに用意される. 2 つのシャドウコピーは各適用処理の後, 必ず一致する仕組みになっている. 編集が衝突し, データに矛盾が生まれた場合でも, それぞれ差を計算し適用することによって各データが同一のものに収束する.

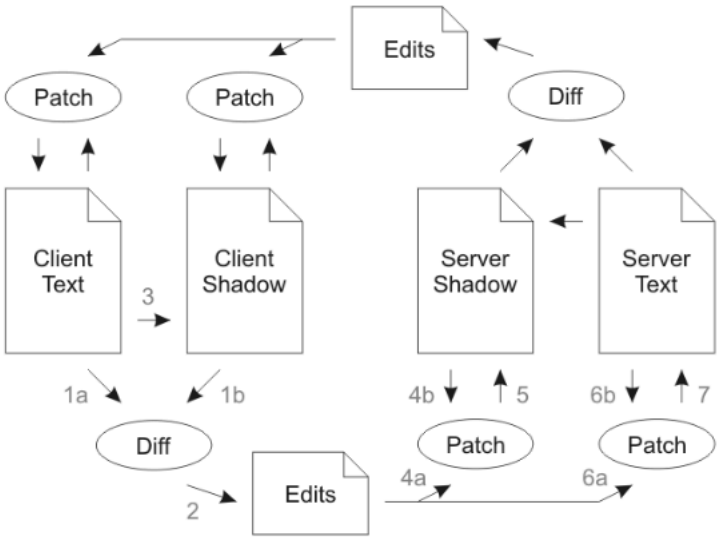

図 2.4 Dual Shadow Method[Fraser09]

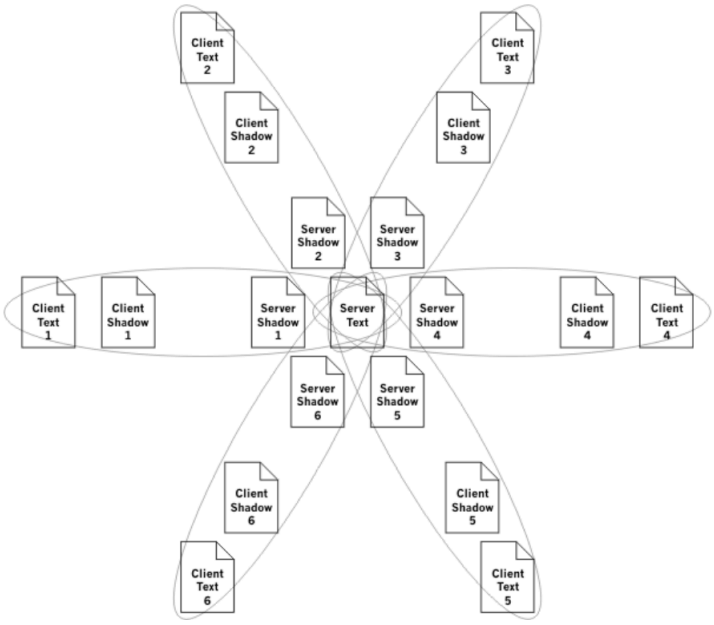

図 2.5 Dual Shadow Method の全体図

2.2.4 CoMaya

CoMaya は 3D 統合ソフト Maya の API を利用し, 3D モデリングにおける同期編集を可能にしたものである. 図 2.6 は CoMaya のシステム図であり, 図中の (a) は Maya のインターフェイス, (b) はオブジェクトと Maya の API の関係のグラフを表している. (c) は CoMaya によって拡張されたオブジェクトのデータと Maya の API を一次元で管理し, 操作変換を適用して同期を行う部分である. 一次元に管理するため, オブジェクトと面や, 面と頂点間の依存関係を解決できておらず, オブジェクトの位置やマテリアルの同期に留まっている.

2.2.5 WFE

WFE[合田 12] は Web ページの同期編集システムである. ある時点でのサーバデータにおけるスナップショットと各クライアントによる編集データを用いる同期機構を採用している. Web ページの HTML における DOM 要素を編集単位として同期を行っている. WFE は, スナップショットがベースとなり元ファイルがあるので, three-way merges と同様に, クライアントによる最新ではないデータに対する編集が行われても, データの矛盾を起こさずクライアント間で同期を可能である.

2.3 three-way merges の 3 次元データへの適用

本研究では, まず最初に, WFE を参考に three-way merges の 3 次元データへの適用を試みた. システムは, 頂点編集が可能なクライアントと、編集のデータを管理するサーバプログラムによって構成した. クライアントの操作ごとに編集のデータをサーバに送信し, サーバはその時点でのスナップショットと各クライアントの編集をマージした. クライアントは一定時間ごとにデータをサーバから受信し適用した. これらの手順よりクライアント間でデータの同期を行なった. その結果, 各クライアントのデータは一致し同期可能であった. しかし, スナップショットをポーリングにより取得するため, 頂点が増えデータの容量が増大する可能性がある 3 次元データでは対応しきれない.

2.4 目的

本研究では, 依存関係の扱いが操作変換より容易になり, 送受信の容量が少なくて済むと予想される Differential Synchronization の同期手法に基づいて, 3 次元データの依存関係を考慮した同期編集機構の手法を提案する. サーバのデータを従来のテキストと同様に一次元で管理すると, 依存関係があった場合にデータの管理が困難となる. この問題に対して, 3D モデリングで用いるエンティティごとにデータモデルを定義し, そのデータモデル間で依存関係を持つことによって解決する. これにより, 複数人で同一の 3 次元データを同時に編集できる 3D モデリングソフトの開発を可能にする.

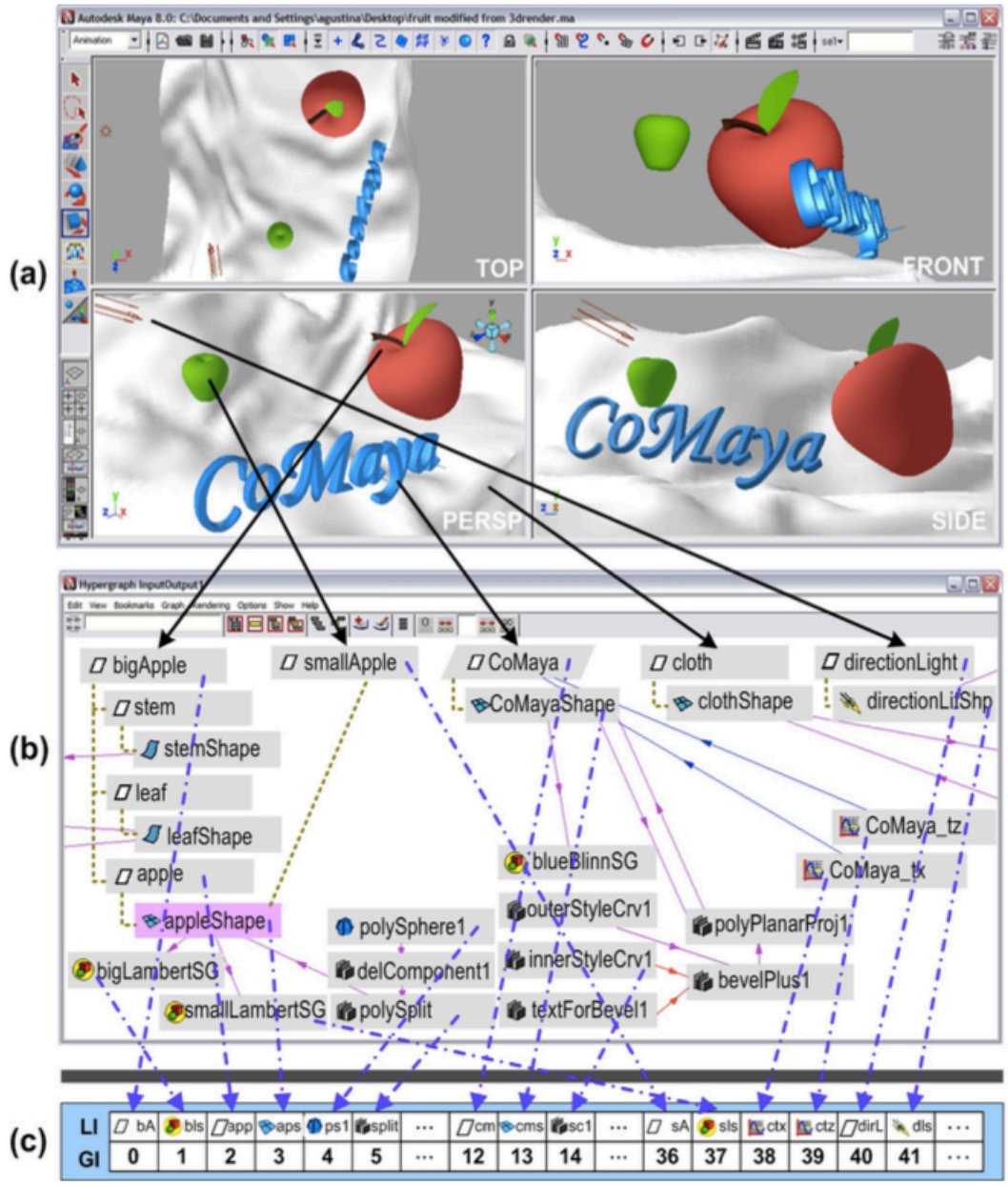

図 2.6 CoMaya のシステム図 [Agustina08]

第3章　提案手法

3.1　システムの構成

本システムは, オブジェクトが複数の面から構成されるサーフェスモデルを扱う. また, 依存関係を考慮したデータを管理するサーバと, データを操作するクライアントによって構成する. クライアントは, クライアントのデータのシャドウコピーを作成しておき, 一定時間ごとにサーバに対して同期を行う. 同期は, ユーザによって更新されているデータとシャドウコピーの差分をサーバに送信する. また, その際にクライアントはシャドウコピーを更新する. 一方, サーバでは受信した差分をサーバのデータに適用する. サーバでもシャドウコピーをクライアントごとに用意し, 差分をクライアントに送信する. クライアントではその差分をクライアントのデータとそのシャドウコピーに適用することで同期を完了する.

3.2　データ構造

本システムでは 3D モデリングで用いるオブジェクト, 面, 頂点の 3 つのデータモデルを定義する. また, サーバデータとそのシャドウコピーであるサーバシャドウを区別するために, シーンというデータモデルを定義する. 新しくデータを作成する場合, データベースにデータモデルを元に作成したデータを登録する.

図 3.1 にデータモデル間の依存関係を表した図を示す. 図 3.1 のようにシーンのデータモデルはオブジェクトを, オブジェクトは面を, 面は頂点をそれぞれ子にもつ. また, 子のみが親の参照先をもつ. 各データモデルのパラメータを図 3.2 に示す. シーンはクライアント識別子というパラメータを持ちサーバシャドウの役割を担う. またシーンのクライアント識別子に 0 を与えサーバデータとする. これらのデータ構造によって, 子を削除した場合に親との依存関係も削除できる. 親を新しく参照する場合, 子のデータを複製しながら新しく参照する親を参照先に追加する. 子のデータを複製していくことによって, 関係を削除した際も複製元のデータは残る.

3.3　固有 ID の付与

各データには, 複数クライアント間で ID の衝突が起こるのを防ぐため, そのデータを作成したクライアント識別子を組み込んだ固有 ID を与える. 本システムでは図 3.3 のように, クライアント識別子およびデータモデル識別子, インクリメント番号, ランダム文字列の順につなげた固有 ID を与える. クライアント識別子は 1 から順に振られたユーザ ID を用いてクライアントが付与する. また, データモデル識別子は, オブジェクトが 0, 面は 1, 頂点は 2 と

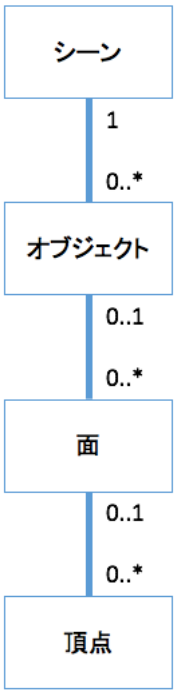

図 3.1 データモデル間の依存関係

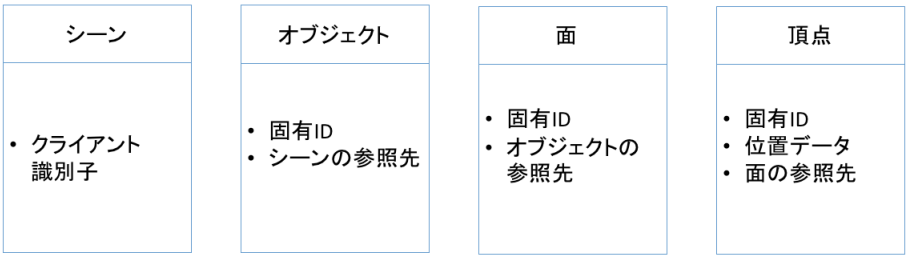

図 3.2 データモデルがもつパラメータ

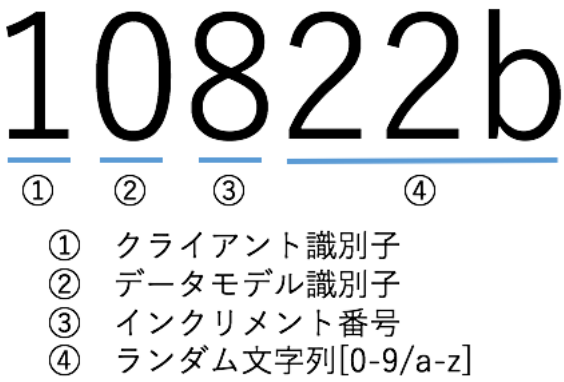

図 3.3 本システムの固有 ID

なるように与える．インクリメント番号は, サーバでもクライアントでも作ったデータの数を記憶しておき, 作るごとにインクリメントする番号である．ランダム文字列は 3 桁の数字またはアルファベットからなる文字列であり, 固有 ID の競合の発生を抑制する．

3.4 同期の手順

同期は Differential Synchronization に基づいて行う．同期の手順を図 3.4 を用いて説明する．なお, 図 3.4 中の各番号は処理の順番を表している．まず (1) クライアントデータとそのシャドウコピーの差分を計算し, (2) サーバに送信する．次に, クライアントでは (3) シャドウコピーを更新し, サーバで (4) 適用処理を行う．サーバ側でも同様の手順で処理することで同期が可能となる．サーバデータは複数のクライアントから利用され, クライアントデータと 2 つのシャドウコピーは接続しているクライアントごとに用意される．

サーバデータを全てサーバシャドウとしてコピーするにはデータ量の負担が大きいため, サーバでのコピーは, サーバデータとサーバシャドウの差分を取り, サーバシャドウに適用することによって解決する．

1 つのサーバに対し 2 つのクライアントの同期の具体的な例を図 3.5 から図 3.7 を用いて説明する．まずクライアント A の同期の手順を図 3.5 に示す．クライアント A が頂点を作成する編集を行うと, クライアント A のクライアントデータに “122xyz” が作成されたとする (図 3.5(a))．クライアント A では, クライアントデータとクライアントシャドウの差分を計算し, [+ “122xyz”] となる．次に, 計算した差分をサーバに送信し, [+ “122xyz”] をクライアント A のサーバシャドウと, サーバデータに適用する (図 3.5(b))．この時にクライアント A では, クライアントデータをクライアントシャドウにコピーする (図 3.5(c))．サーバでは受信したデータの適用後, サーバデータとサーバシャドウとの差分を計算し, 差分なし [ ] となる (図 3.5(d))．この時にサーバではサーバデータをクライアント A のサーバシャドウにコピーする (図 3.5(e))．クライアント B の同期の手順を図 3.6 に示す．クライアント B がすでにある頂点を削除する編集を行うと, クライアント B のクライアントデータにある “121abc” が削除されたとする (図 3.6(a))．クライアント B はクライアントデータとクライアントシャドウの

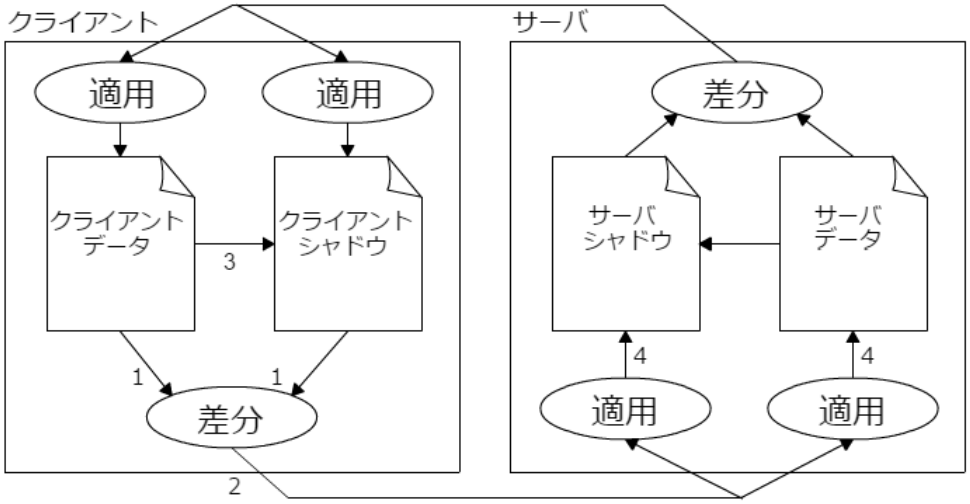

図 3.4 Differential Synchronization

差分を計算し, [– “121abc”] となる. 次に, 計算した差分をサーバに送信し, [– “121abc”] をクライアント B のサーバシャドウと, サーバデータに適用する (図 3.6(b)). その時クライアント B ではシャドウコピーを作成する (図 3.6(c)). サーバでは差分を計算し, [+ “122xyz”] となる. [+ “122xyz”] をクライアント B のクライアントデータとクライアントシャドウに適用する (図 3.6(d)). この時にサーバではシャドウコピーを作成する (図 3.6(e)). またクライアント B の編集をクライアント A が検知して, 各クライアント間でデータが一致するまでの手順を図 3.7 に示す. まず, クライアント A が差分を計算し差分なし [ ] となる (図 3.7(a)). 差分なし [ ] なので, サーバデータとサーバシャドウに適用するものはない (図 3.7(b)). その時にクライアント A ではシャドウコピーを作成する (図 3.6(c)). 次に, サーバデータとサーバシャドウの差分を計算し, [– “121abc”] となる 3.7(d)). 計算した差分をクライアントに送信し, [– “121abc”] をクライアント A のクライアントデータとクライアントシャドウに適用する. この手順により各クライアントのクライアントデータ, クライアントシャドウ, サーバのサーバデータとサーバシャドウのデータが一致し, 同期が可能となる.

3.5 基本命令

オブジェクト, 面, 頂点の各データモデルに対して, 作成, 親への参照の追加, 削除の 3 つの基本命令を定義する. 基本命令はシステムに対する最低限の命令であり, これらの命令を組み合わせることで, 3D モデリングで使われる面の分割や押し出しなど, より高度な命令を実現できる. 基本命令を適用する際のデータの変化を図 3.8 から図 3.13 を用いて説明する. まず最初のデータとして図 3.8 のように, 面が 1 つ, 頂点が 2 つあり, 一方の頂点が面を参照しているとする. 図 3.8 のデータに対して, 頂点の作成命令を適用した場合のデータの変化を図 3.9 に示す. 図 3.9 は頂点 “123ddd” を作成した. 図 3.9 の頂点 “522ccc” に対して,

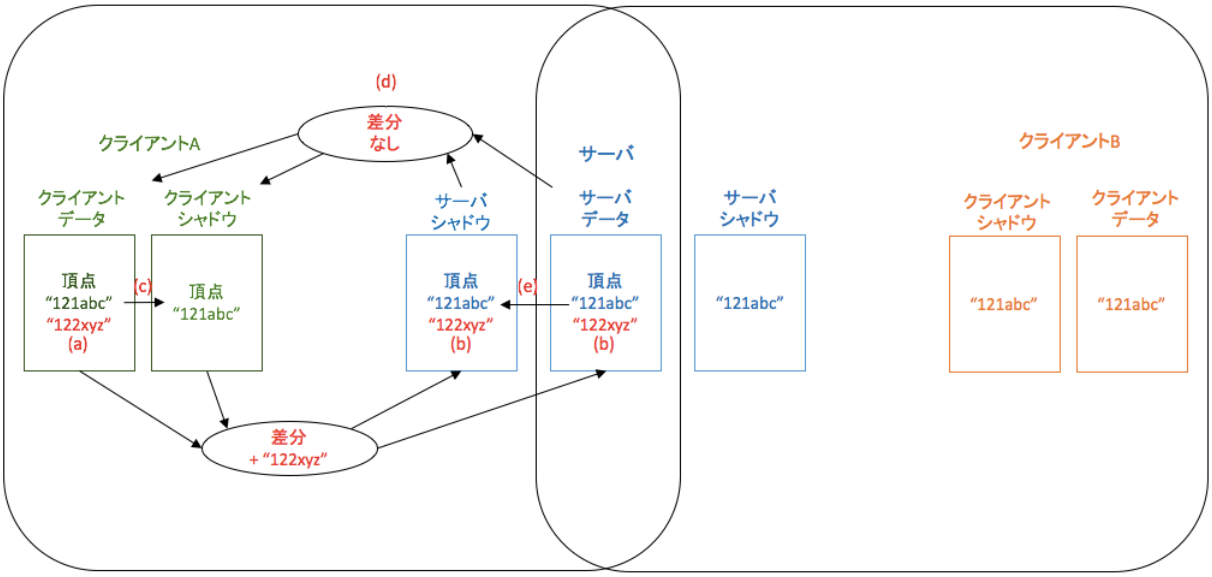

図 3.5 クライアント A の同期の手順

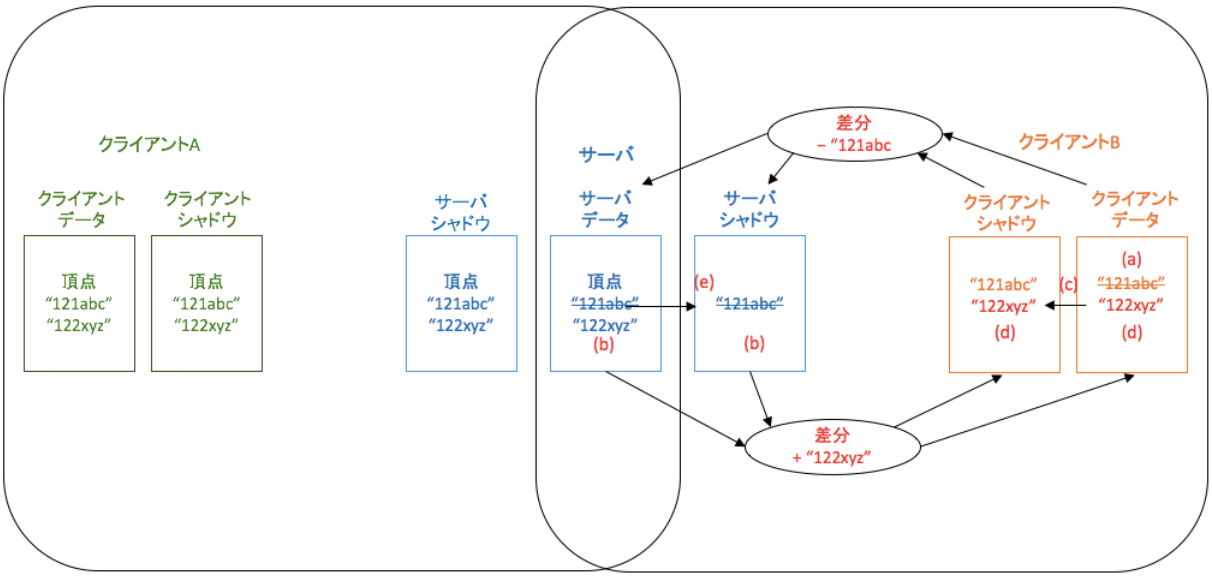

図 3.6 クライアント B の同期の手順

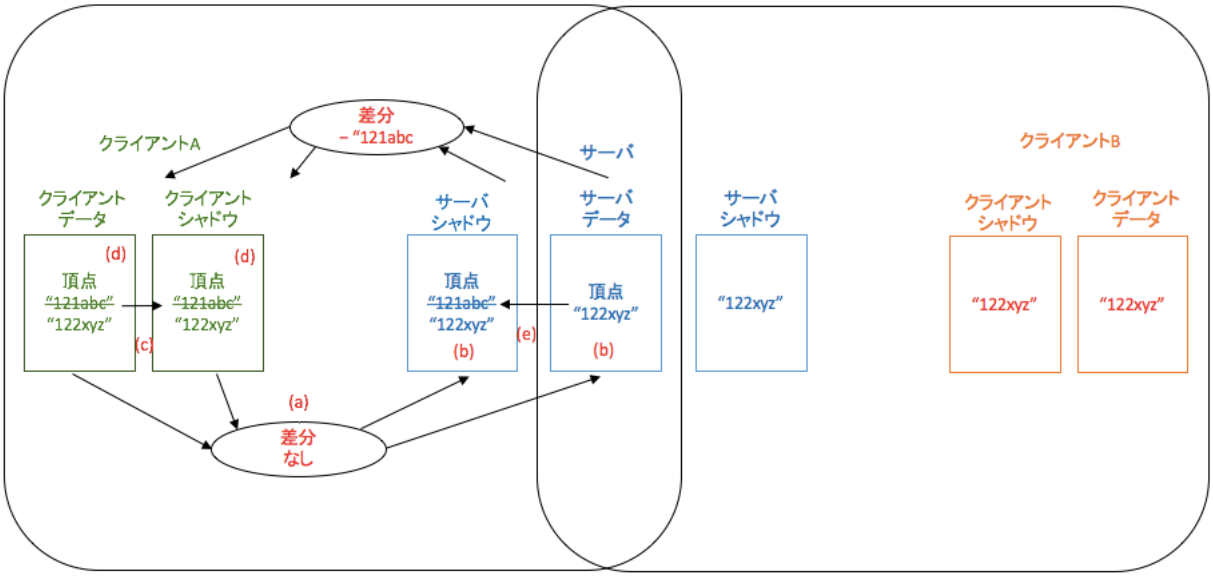

図 3.7 2 回目のクライアント A の同期の手順

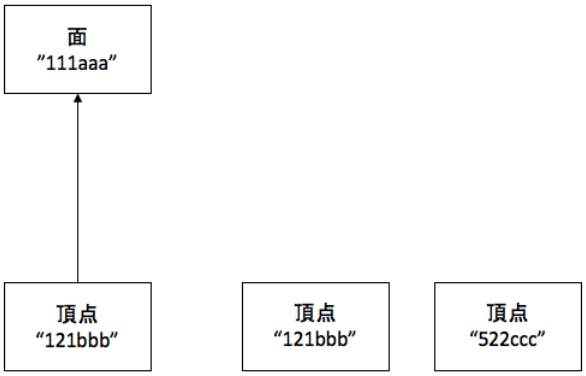

図 3.8 最初のデータ

図 3.9 作成命令適用後

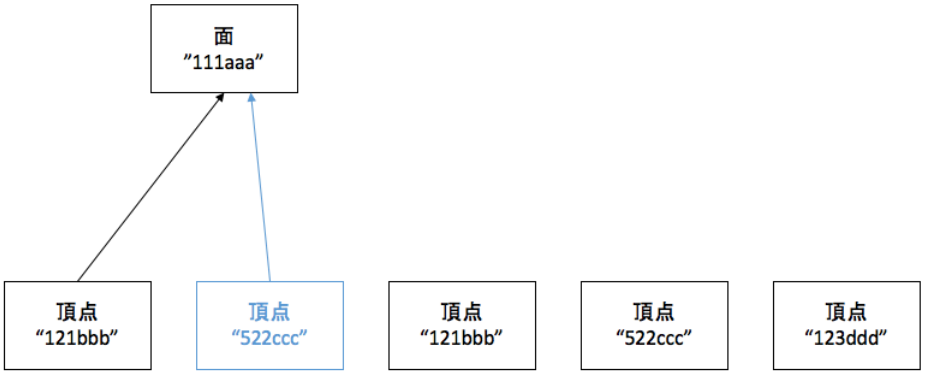

図 3.10 親への参照追加命令適用後

面 “111aaa” を親とする参照の追加命令を適用した場合のデータの変化を図 3.10 に示す. 図 3.10 は子である頂点 “522ccc” を複製しながら面への参照をする. 図 3.10 の頂点 “123ddd” に対して, 削除命令を適用した場合のデータの変化を図 3.11 に示す. 図 3.11 は, 頂点 “123ddd” が削除される. 図 3.11 の頂点 “522ccc” に対して, 削除命令を適用した場合のデータの変化を図 3.12 に示す. 図 3.12 は, 頂点 “522ccc” であるデータが複数あるが, 全ての 頂点 “522ccc” に関するデータを削除する. この際に, 子が親の参照先を持っているので, 参照関係も同時に消える. 図 3.12 のデータである面 “111aaa” に対して, 削除命令を適用した場合のデータの変化を図 3.13 に示す. 図 3.13 は, 面 “111aaa” を削除し, それを参照している子のデータも一緒に削除する. この際, 参照がないデータは削除しない. 削除命令や親への参照の追加をする命令を適用する際, 命令の対象がない場合は適用を無効とする.

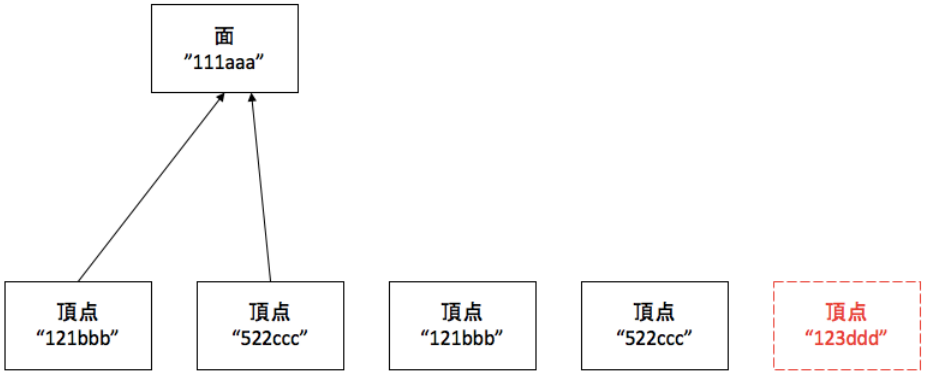

図 3.11 削除命令適用後

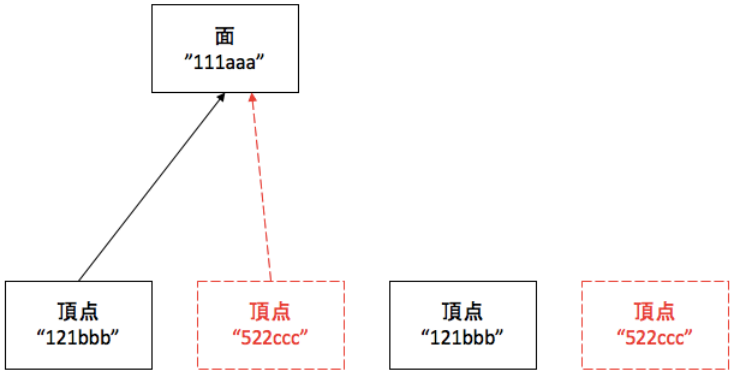

図 3.12 参照関係を含んだデータの削除命令適用後

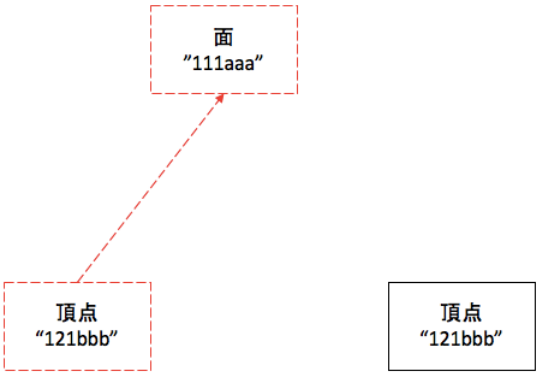

図 3.13 親のデータの削除命令適用後

第4章　実験と考察

4.1　動作実験

4.1.1　実験の目的

Differential Synchronization に基づき, 本研究での提案手法による同期機構で 3 次元データは同期可能であるかを確かめる.

4.1.2　実験環境

本手法を実現するサーバとクライアントを実装した. 本実験で使用したサーバの計算機環境を表 4.1 に示す. また 3 つのクライアントの計算機環境を表 4.2, 表 4.3, 表 4.4 に示す. サーバとクライアントは異なるネットーワーク上に配置し, 通信でのロスも発生する.

4.1.3　実験方法

クライアントを 3 つ用意し, 各クライアントごとに任意の基本命令 50 件を 5 分以内のランダムなタイミングで発行した. 初期データとして図 4.1 のように頂点 2 つ, 面 1 つ, オブジェクト 1 つ準備し, それぞれのクライアント識別子はシステムが生成したとして 0 を与えた. また, クライアントは 4 秒ごとに差分を計算しサーバからの返信が着き次第適用処理を行う. 最後のクライアントが実験を開始して 5 分 10 秒後に各クライアントのデータが一致しているか確かめた.

表 4.1 使用するサーバのスペック

OS	ubuntu
CPU	Intel(R) Core(TM) i5–2520M CPU 2.50GHz
メモリ	2144MB
開発言語	Ruby
データベース	MySQL
Web サーバ	Puma

表 4.2 使用するクライアント 1 のスペック

OS	Microsoft Windows 10 Pro
CPU	Intel(R) Core(TM) i7–2700K CPU 3.50GHz
メモリ	8.00GB
開発言語	JavaScript

表 4.3 使用するクライアント 2 のスペック

OS	Microsoft Windows 10 Pro
CPU	Intel(R) Core(TM) i7–2700K CPU 3.50GHz
メモリ	8.00GB
開発言語	JavaScript

表 4.4 使用するクライアント 3 のスペック

OS	macOS Sierra
CPU	Intel(R) Core(TM) i5-5250U CPU 1.6GHz
メモリ	8GB
開発言語	JavaScript

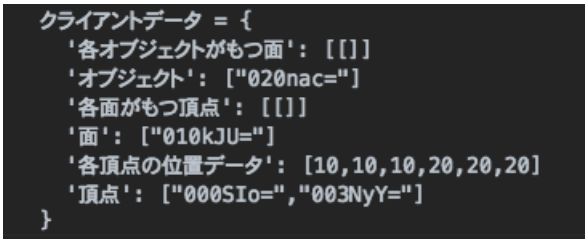

図 4.1 初期データ

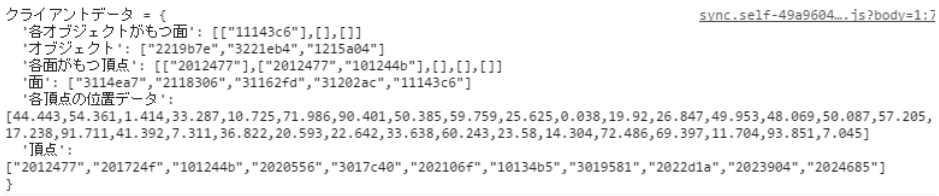

図 4.2 クライアント 1 の結果

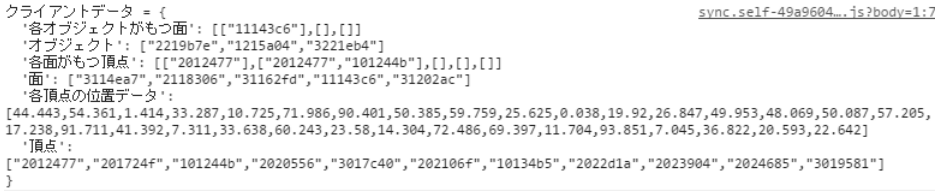

図 4.3 クライアント 2 の結果

4.1.4 実験結果

用意した全ての基本命令を処理した後のクライアント 1 の結果を図 4.2 に, クライアント 2 の結果を図 4.3 に, クライアント 3 の結果を図 4.4 に示す. 各図に示す通り, 各クライアントのデータは一致した.

4.1.5 考察

各クライアントのデータが一致したので, 本研究での提案手法による同期機構で 3 次元データは同期可能であった.

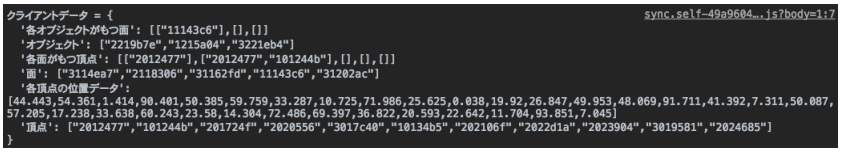

図 4.4 クライアント 3 の結果

表 4.5 使用するサーバのスペック

OS	macOS Sierra
CPU	Intel(R) Core(TM) i5-5250U CPU 1.6GHz
メモリ	8GB
開発言語	Ruby
データベース	MySQL
Web サーバ	Puma

表 4.6 使用するクライアント 1 のスペック

OS	macOS Sierra
CPU	Intel(R) Core(TM) i5-5250U CPU 1.6GHz
メモリ	8GB
開発言語	JavaScript

4.2 負荷実験

4.2.1 実験の目的

Differential Syncronization に基づいた, 本研究の同期機構が大量のリクエストに対して, どの程度影響が出るかを確かめる.

4.2.2 実験環境

本手法を実現するサーバとクライアントを実装した. 本実験で使用したサーバの計算機環境を表 4.5 に示す. また 2 つのクライアントの計算機環境を表 4.6, 表 4.7 に示す. サーバとクライアントは同一のネットワーク上に配置し, 通信においてのロスはない.

4.2.3 実験方法

クライアントを 2 つ用意した. クライアントは 4 秒ごとに差分を計算しサーバに差分のリクエストを送信した. クライアント 1 がサーバに差分をリクエストした時刻からレスポン

表 4.7 使用するクライアント 2 のスペック

OS	macOS Sierra
CPU	Intel(R) Core(TM) i5-5250U CPU 1.6GHz
メモリ	8GB
開発言語	JavaScript

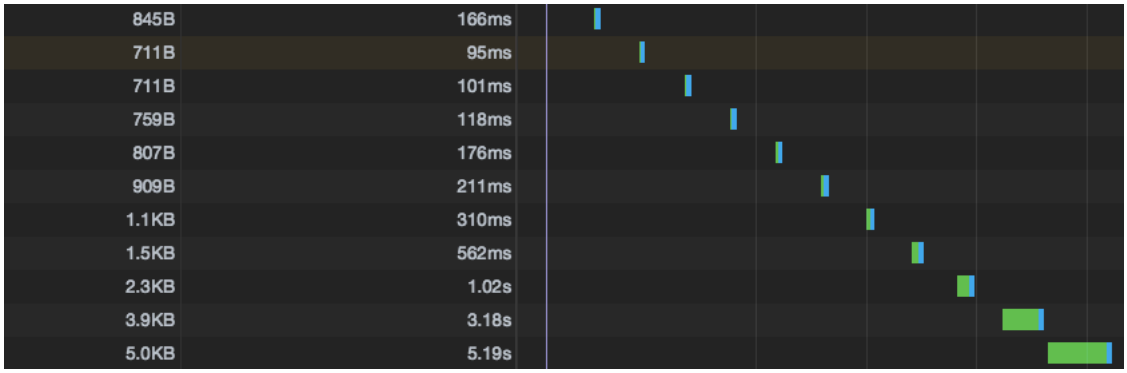

図 4.5 (1) の計測による同期レスポンス時間

スが届いた時刻までを同期レスポンス時間と定義した. 2 パターンの同期レスポンス時間の計測を行った. まず, (1) クライアント 1 は 4 秒ごとに面を作る命令を送信した. 面を作る命令の数を 4 秒ごとに倍にしていき, 同期レスポンス時間の計測を行った. 最初の面を作成する数は 1 個とした. 同期レスポンス時間が 5 秒を超え同期に遅延が生じたと判断できるところで計測を終了した. 次に, (2) クライアント 1 は 4 秒ごとに, 20 個の面を作成する命令を, (1) で計測不能になるまで作成した面の数になるまで送信した. それまでのクライアント 1 の同期レスポンス時間を計測した.

4.2.4 実験結果

(1) の計測による, クライアント 1 での同期レスポンス時間の計測結果を図 4.5 に示す. 項目は左から送信するデータの大きさ, 同期レスポンス時間, 同期レスポンス時間を表したタイムラインである. 計測の結果, 命令の数が増えるとデータの大きさが大きくなり, 同期レスポンス時間が増えた. (1) の計測で, 面が 400 個作成されたところで終了した. よって (2) は面が 400 を超えるまで計測を続けた. (2) の計測による, クライアント 1 での同期レスポンス時間の計測した結果を図 4.6 に示す. 計測の結果, 徐々に同期レスポンス時間が増えているが, (1) のように同期に支障が出るほどではなかった.

4.2.5 考察

4.2.4 項の結果より, 面を作る数が増え, 一度に大量の編集命令がサーバにリクエストされた場合, サーバにとって負荷となる. また, 命令を分散させて送信すれば, 徐々に同期にかかる時間は増えるが, 本システムは問題なく同期サイクルを回すことができる. 以上より, サーバでの適用や差分の計算に要する時間が多くかかっていることがわかり, 原因として, データベースでの挿入や検索がボトルネックになっていると考えられる.

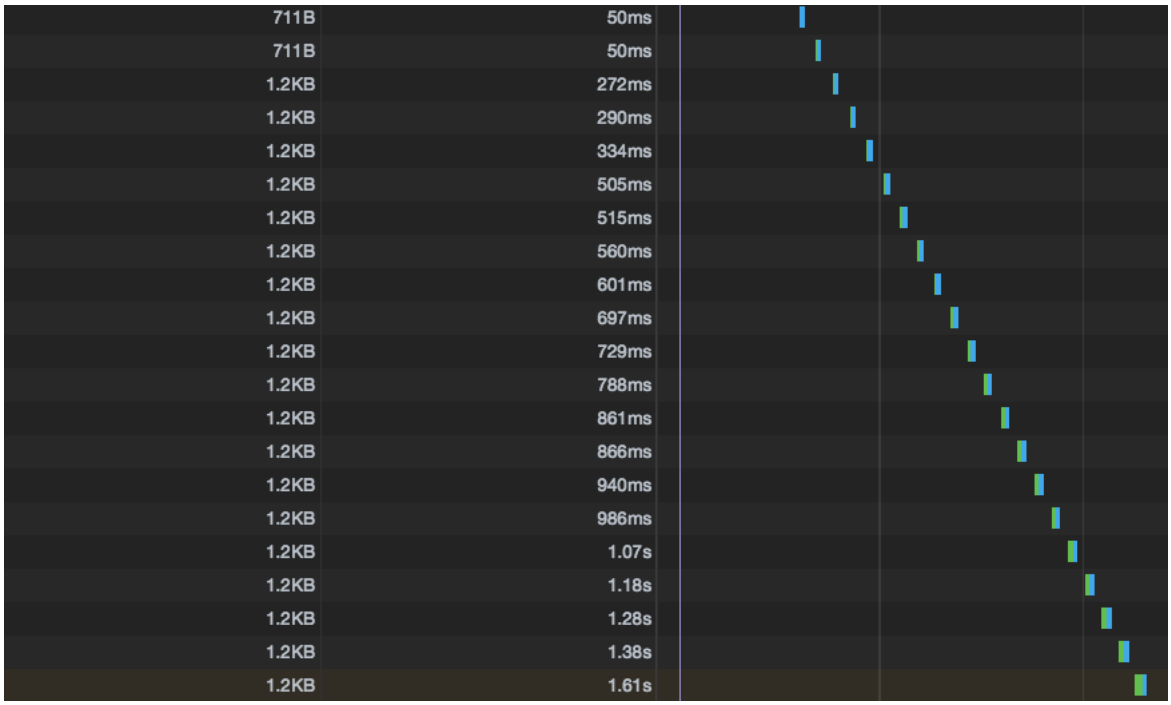

図 4.6 (2) の計測による同期レスポンス時間

第5章　むすび

一般的なサーフェスモデルの 3D モデリングソフトでは面の表と裏を区別する仕様がほとんどである. 現在の実装では, 面を構成する頂点の順番を扱っておらず, 面の法線方向を表現できていないため, 3D モデリングの同期機構としては不十分である. また, オブジェクトのマテリアル情報やテクスチャ座標情報も扱っていない. これらを実装することで, クライアントがデータから一意なオブジェクトを描画することができ, 同期編集可能な 3D モデリングを実現できる. さらに, 3D モデリングではオブジェクトを移動し, 複数の頂点が一度に更新される場合がある. 本システムでは, データベースのデータの挿入と検索がボトルネックであり, 大量のリクエスト時の同期の成否はサーバのスペックに依存してしまう. CoMaya のようにオブジェクトの状態を操作する API を同期する仕組みを併用すると, オブジェクト単位で同期でき, 複数の頂点の更新も可能となると考えられる.

謝辞

本研究を進めるあたり, 終始御指導下さいました乃万司教授および山本邦雄助教に心から御礼申し上げます. そして, 様々な点で研究をご援助下さいました松元隆二技術職員をはじめ知能情報メディア部門のみなさまに感謝の意を表します. 最後に, 常日頃よりご指導を頂き, 研究を見守ってくださった乃万研究室の皆様に心より感謝の念を表し, 謝辞とさせて頂きます.

参考文献

[Google] Google ドキュメント, `https://www.google.com/intl/ja_jp/docs/about/`, Google Inc.

[Microsoft] Microsoft Word Online, `https://office.live.com/start/Word.aspx`, Microsoft Corporation.

[Agustina08] Agustina, Fei Liu, Steven Xia, Haifeng Shen, and Chengzheng Sun, "CoMaya: Incorporating Advanced Collaboration Capabilities into 3D Digital Media Design Tools," *Proc. Computer Supported Cooperative Work '08*, pp. 5–8, 2008.

[Fraser09] Nail Fraser, "Differential Synchronization," *Proc. 9th ACM Symposium on Document Engineering '09*, pp. 13–20, 2009.

[Ellis89] Clarence Ellis, Simon John D. Gibbs, "Concurrency Control in Groupware Systems," *Proc. International Conference on Management of Data '89*, pp. 399–407, 1989.

[Lindholm04] Tancred Lindholm, "A Three-Way Merge for XML Documents," *Proc. ACM Symposium on Document Engineering '04*, pp. 1–10, 2004.

[合田 12] 合田拓史, 井上良太, 加藤雄大, 白松俊, 大囿忠親, 新谷虎松, "既存 Web ページ同期編集システム WFE-S における差分同期機構の試作," 情報科学技術フォーラム講演論文集, Vol. 11, pp. 31–37, 2012.

署名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿