ARG WI2 No.9, 2016
WI2-2016-22

リアルタイムWeb協調作業のための多層レイヤーCanvas同期機構の実現

伊藤　栄俊[†,a]　　大囿　忠親[††,b]　　新谷　虎松[††,c]

† 名古屋工業大学情報工学科　†† 名古屋工業大学大学院情報工学専攻

a) *ashun@toralab.org*　b) *ozono@toralab.org*　c) *tora@toralab.org*

概要　リアルタイム Web 協調作業を行う上で，コンテンツ同期のリアルタイム性は重要である．描画処理において，コンテンツが増加するほどリアルタイムな同期を保つことは困難になる．本研究では，Web ブラウザ間でのリアルタイムなコンテンツ同期を実現するために，HTML5 の Canvas 要素を動的に多重レイヤー化することで，大量のオブジェクトに対しても描画コストを軽減する．また，ユーザーの特性に基づくオブジェクトの更新頻度の偏りに注目した．ユーザーの特性とオブジェクトの更新頻度に関する実験を重ね，得られた結果より，複数オブジェクトを同時に更新処理するためのアルゴリズムを提案する．提案する同時処理アルゴリズムを評価することで本システムの有用性を示す．最後に，本システムを用いて開発したリアルタイム Web 協調作業アプリケーションの紹介を行う．

キーワード　HTML5，Canvas，リアルタイム，協調作業支援

1　はじめに

オンライン型のリアルタイム Web 協調作業においてリアルタイム性は重要である．本研究では，Web 上でのリアルタイムな同期の実現における描画の高速化に注目した．一般的に，Web 上でのリアルタイムな同期の実現において，通信の遅延が重要である．しかし，Web 上のビットマップ描画機構である HTML5 の Canvas(以降 Canvas) における描画の遅延は，通信の遅延よりもリアルタイム性に悪影響をおよぼすことがある．

リアルタイム Web 協調作業では多人数での操作により大量のオブジェクトを描画されることがある．オブジェクト数の増加に伴い，リアルタイム性を保つことが困難になる．本研究では，リアルタイムな描画処理を可能にするため，片山らのレイヤー化の手法を応用する [1][2][3]．レイヤー化の手法を組み合わせることで，大量のオブジェクトに対してもリアルタイムなレスポンスが可能な，多層レイヤー Canvas 同期機構を開発した．本システムのアプリケーションを示す．

以降，本論文では，2 章で Canvas の再描画問題について述べ，3 章で不定レイヤー Canvas 技術と本システムのための予備実験の結果を示し，4 章で多層レイヤー Canvas 同期機構とオブジェクトの同時処理方法に関して説明を行い，5 章で本システムと同時処理についての評価を示し，6 章で本システムのアプリケーションの紹介，7 章で今後の研究に関連する研究の紹介，8 章で本論文をまとめる．

2　Canvas 再描画問題

HTML5 の Canvas 要素とは，Web 上で 2D グラフィックの描画を行うための機構として W3C によって仕様の策定が完了している [4]．HTML5 の Canvas 要素は 2 次元ビットマップ画像として描画される．Canvas は DOM 要素である Canvas 要素と，Canvas 要素から取得できる Canvas コンテキストから構成されている．Canvas 上にオブジェクトを描画する際は，Canvas コンテキストに対して命令を行う．

通信遅延よりも Canvas の描画の遅延がリアルタイム性に悪影響を及ぼすことがある．片山の研究では，オンライン上で PDF 文書を協調編集するために，Canvas を用いた同期機構を提案した [5]．PDF．js[6] による PDF 文書の閲覧機能と，リアルタイムなアノテーション機能による協調編集システムを開発した．そこで，Canvas での描画におけるリアルタイム性を高めるための技術を実現する必要がある．

Canvas 要素をリアルタイム Web 協調作業の Web アプリケーションに適応するためには，再描画問題を解決する必要がある．再描画問題とは，オブジェクトの更新時に再描画する必要のないオブジェクトも再描画してしまう問題である．再描画時の流れを説明する．Canvas 上に，オブジェクト A とオブジェクト B が存在していたとする．オブジェクト B はオブジェクト A の上に重なるように描画されていたとする．オブジェクト B の座標を移動させる場合を考える．Canvas 上のオブジェクト B は移動により位置情報が更新されるため，元々描画されていた部分を削除する必要がある．この際，オブジェクト B によって隠れていたオブジェクト A の一部がオブジェクト B の移動に伴い，顕在化する．Canvas は 2

次元ビットマップ画像表現されているため，オブジェクト A を適切に表示させるためには，オブジェクト A に対して，再描画処理を行う必要性がある．オブジェクト B の更新によって，オブジェクト A も再描画する，無駄な再描画処理が発生する．Canvas 上のオブジェクト数が増加するに従い，無駄な再描画処理数が増加する．

3 不定レイヤー Canvas

片山らは，再描画問題を解決するために Canvas のレイヤー化を行った．オブジェクトはレイヤー化された Canvas 上に描画される．レイヤー当たりに描画されるオブジェクト数を減らせるので，無駄な再描画が減少する．片山らの提案した Canvas のレイヤー化は 3 種類存在する．

Drawing-Frequency based Layered Canvas(DFLC) という手法である [1]．更新頻度既知のオブジェクトを更新頻度ごとに分類し，レイヤーに割り当てる．頻繁に更新されるリアルタイム（Realtime）レイヤー，更新される可能性のある可変（Mutable）レイヤー，更新される可能性が低い，もしくは更新されない不変（Immutable）レイヤーの 3 種類である．本システムでも，更新頻度が既知のオブジェクトであるときは更新頻度に応じて分類する．

DFLC には問題点がある．DFLC は更新頻度が既知の場合には有効であるが，未知の場合にはレイヤー分割を行うことができない．そこで，片山はオブジェクトの描画履歴から更新頻度を予測し，動的に最適なレイヤーへと割り当てる Generational Layered Canvas (GLC) を提案した [2]．過去に更新頻度が高かったオブジェクトは，以降も更新頻度が高い，という予測にもとづきレイヤー分割を行う．更新頻度が低いと判断されたオブジェクトは，描画先のレイヤーを変更する Promotion という概念を用いている．これにより，更新頻度が未知のレイヤーに対しても適用することができ，動的なレイヤー生成を行い，過去の更新頻度から大量のオブジェクトに対しても描画遅延を抑えることが可能となる．しかし，Promotion によりオブジェクトのレイヤーが入れ替わるため，オブジェクトの重なり順序が入れ替わってしまう．そこで，大囿らは不定レイヤー Canvas を導入することで解決した [3]．不定レイヤー Canvas とは，レイヤー l の描画コストが閾値 θ を超えた時，動的に生成されるレイヤーのことである．この処理を不定レイヤー分割という．不定レイヤー分割では，描画コストが一定になるようオブジェクトが分配される．GLC とは違い，動的にレイヤーが生成されるため，メモリ使用量を抑えつつ，更新頻度が未知のレイヤーに対しても有効となる．

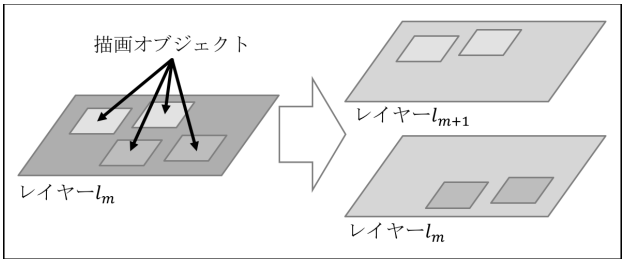

図 1 不定レイヤー分割

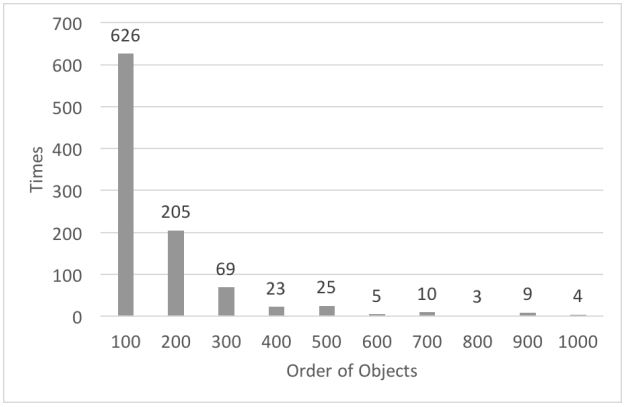

図 2 オブジェクトサイズが 1% 時の表示順序と更新頻度

3.1 オブジェクトの更新頻度の偏り

不定レイヤー分割にも課題がある．オブジェクトの更新頻度は開発するアプリケーションに対する依存が大きいことである．そこで，オブジェクトの更新されるタイミングに注目した．オブジェクトが更新されるタイミングには 3 種類ある．システムからの選択による更新，ユーザーからの選択による更新，レイヤー更新時の無駄な再描画である．システムからの選択による更新は設計時に予測できる．よって予測できない更新はユーザーからの選択になる．ユーザーからの選択による更新に重きをおく．ユーザーがオブジェクトを選択する際に重要になるのは，画面上でのオブジェクトの描画状態である．ユーザーからの選択による更新頻度の偏りについて予備実験を行った．予備実験時の実行環境は，ブラウザが GoogleChrome Version 54．0．2840．71(64-bit) で，OS X ElCapitan Vesrsion10.11.6，2.7GHz Intel Core i5，8GB 1867MHz DDR3 である．Canvas 上に 10000 個のオブジェクトを描画し，Canvas 上の任意の位置を指定する．指定した位置にオブジェクトが存在するならばオブジェクトを更新する，というヒットテストを 1000 回行い，オブジェクトの更新回数の偏りを調べた．以降よりオブジェクトサイズはオブジェクトの面積のことを指すこととする．図 2 は，オブジェクトサイズが Canvas サイズの 1% のときの結果である．横軸がオブジェクトの表示順序，縦軸が更新回数である．

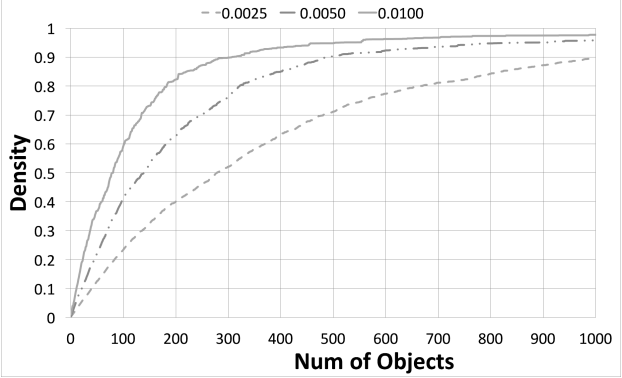

図 3 オブジェクトの表示順序と描画密度

結果より，更新処理のおよそ 90% が 300 番目までのオブジェクトに対するものであることがわかった．画面に表示されているオブジェクトに対して，ユーザーからの操作によって選択されたオブジェクトの更新頻度には偏りが生じると分かる．オブジェクトサイズが大きいほど，更新頻度の偏りは大きくなる．逆に，オブジェクトサイズが小さい場合，偏りが小さくなる．オブジェクトが更新される時，オブジェクトが割り当てられているレイヤーも更新されるため，オブジェクトの更新頻度の偏りは，レイヤーの更新頻度の偏りに対応する．予備実験 2 では，オブジェクトサイズと Canvas 全体の大きさに注目した．

3.2 Canvas の描画密度

Canvas 上に大量のオブジェクトが描画されている時を考える．オブジェクト数が増えるに従い，Canvas はオブジェクトにより被覆される．レイヤー全体の面積に対するオブジェクトの占める面積を描画密度 d と定義し，以下の式のように求める．ただし，$\mathbb{O}$ は全描画オブジェクト集合，$VS(o_i)$ をユーザーが視認可能なオブジェクト o_i の面積，S をオブジェクトの面積とする．

$$d = \frac{\sum_{o_i \in \mathbb{O}} VS(o_i)}{S_{layer}} \tag{1}$$

ランダムにオブジェクト 10000 個を描画した．図 3 は，横軸がオブジェクトの表示数，縦軸が描画密度を表したグラフである．グラフ上の曲線は，上から順にレイヤー l_j 全体の面積に対するオブジェクトの面積が，1%，0.5%，0.25% のときのものを示している．

オブジェクト数が増加するに伴い，描画密度が大きくなり，値が 1 へと収束していることが分かる．この結果より，レイヤー l_j に対するオブジェクト o_i の重み w を次のように定める．

$$w(o_i) = \frac{S_{o_i}}{S_{l_j}} \tag{2}$$

レイヤー l_j に割り当てられたオブジェクトの集合を $\mathbb{O}_j$ としたとき，レイヤーの重み $w(l_j)$ は次のようになる．

$$w(l_j) = \sum_{o_i \in \mathbb{O}_j} w(o_i) \tag{3}$$

レイヤーの上層のレイヤーからの累積の重みを考える．$l_0, l_1, \cdots l_k$ までのレイヤーの集合を $\mathbb{L}_k$ とすると，$\mathbb{L}_k$ の重み W は次のようになる．

$$W(\mathbb{L}_k) = \sum_{i=0}^{N} w(o_i) \tag{4}$$

$$N = \sum_{j=0}^{k} |\mathbb{O}_j| \tag{5}$$

図 3 より，$W(\mathbb{L}_k)$ が 1.0 を超えると，描画密度が 0.6 を上回っていることが分かる．2.5 を超えると，描画密度が 0.9 を上回っていることが分かる．つまり，上層レイヤーからの累積重みから描画密度が予測できる．また，累積重みが 1.0 超えているということは，オブジェクトに重なりが発生していることになる．重なりが大きくなるほど，ユーザーから視認できないオブジェクトが増える．視認できないオブジェクトは更新頻度が低くなる．ただし，オブジェクトの局所性が著しい場合は描画密度はオブジェクト数の増加の影響を受けなくなる．

3.3 レイヤー数と描画処理

再描画問題より，1 レイヤーに描画されるオブジェクト数が少ないほど無駄な再描画処理の回数は減少する．1 レイヤー当たり 1 オブジェクトを割り当てることで，無駄な再描画は発生しないことになる．しかし，レイヤー数の増加に伴うメモリ使用量の増加が発生する．メモリ使用量の増加はメモリリークを引き起こす可能性も高め，描画処理時間の低下を引き起こす恐れがある．そこで，1 レイヤー当たりのオブジェクト数と描画処理時間の関係を評価した．オブジェクト数は 10000 で，レイヤーには等しくオブジェクトを割り当てた．図 4 はレイヤー数と描画処理時間の関係を示した図と表である．図 4 の横軸は，1 レイヤー当たりのオブジェクト数を表し，縦軸は描画処理時間を表している．縦軸横軸どちらも対数尺度になっている．1 レイヤーあたりのオブジェクト数が 1 のときと 100 の時を比較したところ，描画処理時間に大きな差は見られなかった．これは，オブジェクト数が少ない時は，無駄な再描画によるコストより，レイヤーの更新処理のコストが大きいためだと考えられる．また，1 レイヤーあたりのオブジェクト数とレイヤー数は反比例の関係にある．メモリーあたりの描画処理時間は 1 レイヤーあたりのオブジェクト数が 100 のときの方が小さいと言える．本システムでも，1 レイヤーあたりのオブジェクト数が 1000 以下になるよう設計した．

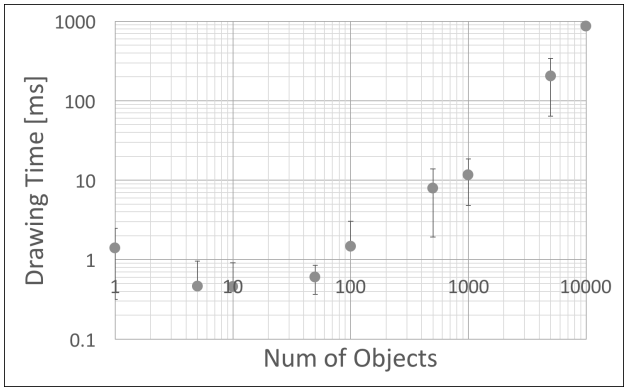

図 4　各レイヤーにおけるオブジェクト数と描画処理時間（対数尺度）

3.4　予備実験のまとめと考察

オブジェクトの更新頻度の偏りを調べた．ヒットテストを行ったところ，更新頻度の偏りが確認され， オブジェクトの表示順序と面積の影響を受けていた．オブジェクトの面積と描画密度を調べたところ，レイヤーの累積重みから描画密度が推定できた．1 レイヤーあたりのオブジェクト数と描画処理時間の関係を調べた．1 レイヤー 1 オブジェクトの場合が，無駄な再描画処理がなく理想的である．実際は，無駄な再描画よりも他の処理のコストが大きく，メモリー使用量も大きくなってしまうため効果的ではないとわかった．

これらの予備実験を踏まえ，多層レイヤー Canvas 同期機構を開発した．多層レイヤー Canvas 同期機構では，オブジェクトサイズに応じてレイヤーに割り当てるオブジェクト数を動的に変更することにした．さらに，更新頻度の偏りとオブジェクトサイズの考察より，同時処理機構を提案する．

4　多層レイヤー Canvas 同期機構

4.1　システム構成図

図 5 に開発した多層レイヤー Canvas 同期機構の描画処理機構のシステム構成図を示す．多層レイヤー Canvas 同期機構はオブジェクト順序管理，レイヤー優先度管理，レイヤー順序管理，不定レイヤー分割からなる．各レイヤーは DOM 上の Canvas 要素である．ユーザーは操作画面から更新するオブジェクトを選択する．ユーザーから選択されたオブジェクトは，一度オブジェクトスタックに蓄えられる．オブジェクトスタックのサイズは，レイヤー描画コストと開放するオブジェ平均オブジェクトサイズから動的に変更する．オブジェクトスタックから開放されたオブジェクトは，レイヤー管理，オブジェクト管理に渡され描画処理が行われる．

本システムのレイヤー構造には，DWLC と不定レイヤー Canvas を組み合わせた多層レイヤー構造を採用し

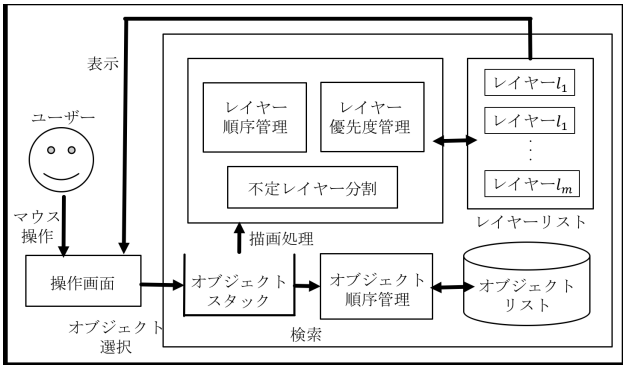

図 5 システム構成図

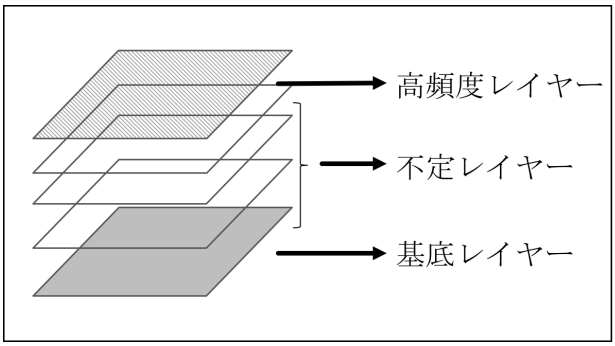

図 6 多層レイヤー構造

た．図 6 に多層レイヤー構造を示す．まず，更新頻度が高いオブジェクト，低いオブジェクトをそれぞれリアルタイムレイヤーと不変レイヤーに割り当てる．更新頻度が未知なオブジェクトは不定レイヤーに割り当てられる．

4.2　同時処理

予備実験から，ユーザーからの操作に基づくオブジェクトの更新には偏りがあり，更新すべきレイヤー数は限定される．また，あるオブジェクトの更新から他のオブジェクトが波及的に更新されることが考えられる．そのため，本システムでは複数オブジェクトの同時処理機構を開発した．同時処理方法を 2 種類提案する．一つ目は，更新されるオブジェクトを，割り当てられているレイヤー毎にまとめ，レイヤー単位で更新する方法である (バッチ処理方式)．レイヤー毎に更新するオブジェクトをまとめるため，無駄な再描画回数が減る．同時処理するオブジェクト数を限定しなければ，最悪時には全てのレイヤーを更新することになる．予備実験より，ユーザーからの操作に基づく更新の場合は，再描画されるレイヤー数が限定されるためこの方式が可能となる．Algorithm1 にアルゴリズムを示す．

もう一つは，描画密度に基づくレイヤーの更新である．予備実験より，大量のオブジェクトが描画されている場合は描画密度が高くなることがわかった．ユーザーは全ての更新に関係しない全てのオブジェクトを把握す

Algorithm 1 同時処理（バッチ処理方式）

Require: *Stack*：オブジェクトスタック，*map*：レイヤーの *id* をキーとした描画オブジェクトの配列の連想配列

Ensure: オブジェクトスタック内のオブジェクトの再描画処理

```
for o in Stack do
    l ← Layer(o)
    key ← l.id
    if map[key].isEmpty() then
        map[key] ← newArray()
    end if
    map[key].add(o)
    update(o)
end for
for key in map.keys() do
    l ← getLayer(key)
    update(l)
end for
```

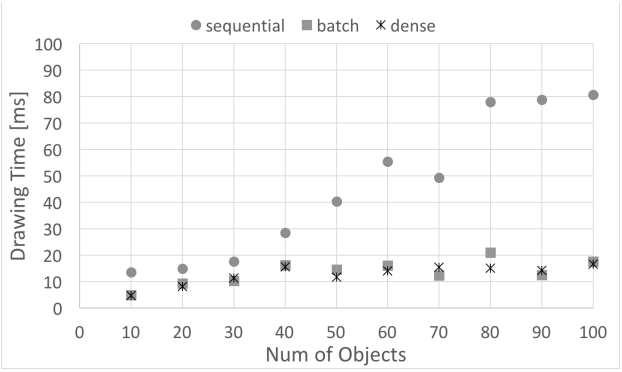

図 7 逐次処理と同時処理時の描画処理時間

ることは難しい．なので，全体の 90% 程度を占めるレイヤーを更新することで，オブジェクトが更新されたように見せる．描画密度は，累積重みから推測する．

5 評価と考察

3 種類の方法で複数オブジェクトの処理を行った．オブジェクトを一つ一つ再描画処理を行う逐次処理方式 (sequential)，更新オブジェクトを更新レイヤーごとにまとめ更新するバッチ処理方式 (batch)，描画密度が 0.9 以上のレイヤー集合のみを更新する描画密度方式 (dense)，以上の 3 種類である．平均オブジェクトサイズが Canvas サイズの 1% である 10000 オブジェクトに対して，同時処理するオブジェクト数を変化させながら描画処理時間を測定した．結果のグラフを図 7 に示す．横軸が同時処理するオブジェクト数，縦軸が描画処理時間である．逐次処理方式の場合，処理オブジェクト数が増加するに従い，線形に近い形で描画処理時間が上昇している．それに対して，バッチ処理方式，描画密度方式は変化が少ない．予備実験で確認されたように，レイヤーの更新頻度の偏りにより更新されるレイヤー数が限定されているためであると考える．バッチ処理方式と描画密度方式で処理時間に差が少ないのは更新されるレイヤー数の差が少ないためと推測される．バッチ処理方式，描画密度方式のそれぞれで更新されたレイヤー数を比較したところ，同時処理オブジェクト数が全体のオブジェクト数の 1% の時で，高々10 レイヤー程度であった．同時処理数を 100 より大きくしても差が出なかった．描画処理時間よりも同時処理オブジェクトの情報の更新処理時間が支配的になるためである．

オブジェクトを更新した際に，更新オブジェクトを元のレイヤーから最前面のレイヤーに移動させることで更新頻度の偏りを活かすことができる．更新頻度が高いオブジェクトほど，更新頻度が高いレイヤーに集まり，更新頻度が低いオブジェクトほど，更新頻度が低いレイヤーに集めることができる．

本システムでは，バッチ処理方式を採用した．採用した理由は以下のとおりである．バッチ密度処理方式と描画密度方式で大きな差は出なかったが，描画密度方式は描画されないオブジェクトが存在し，推測も不安定となる．バッチ処理方式は，必ず再描画に関連するオブジェクトとレイヤーが更新される．

ユーザーからの選択に基づく更新頻度を，描画されるオブジェクトの面積から推測し，再描画処理を高速化することができた．システムからの更新は設計時に推測でき，他のオブジェクトの更新に付随した更新が発生しても，複数オブジェクトの同時処理技術によってリアルタイムに処理できる．DWLC により，更新頻度が既知のオブジェクトに対しても有効である．よって，多層レイヤー Canvas 同期機構は，更新頻度によらずに大量のオブジェクトに対してリアルタイムな描画が可能であることがわかった．

6 アプリケーション

本システムを応用して開発したシステムを紹介する．一つ目は，発表時の状況に応じて，コンテンツを即応的に取捨選択するためのプレゼンテーション支援システムを開発である．プレゼンテーションにおいて聴講者とのインタラクションは重要である．聴講者の反応に応じた内容にスライドを変化させるために，発表中のスライド文章の情報を取得し，コンテンツを動的に呼び出すシステムを開発した．

二つ目は，画像共有アプリケーションである．多層レ

図 8 画像共有アプリケーション

イヤー Canvas 同期機構により大量の画像をアップロードしてもリアルタイム性が保たれる．また，自動タグ付け機能により円滑な画像共有が可能となった．

三つ目は，付箋を用いた会議を支援するための電子付箋会議システムである．Post-it などの物理世界の付箋を電子化し，スクリーン上に投影することで，物理付箋と電子付箋を用いた会議を実現する．リアルタイム性から遠隔地間との会議への利用が考えられる．

7 関連研究

本章では，本 Canvas 同期機構の改良に関する関連研究について議論する．VR 技術の発達により，3D モデルでのリアルタイム処理の研究が盛んである．Kapetanakis らは，WebSocket と X3Dom での 3 次元表現でのリアルタイム性を実現している [7]．X3Dom とは XML を 3D に対応させたフォーマットである．Canvas には WebGL という 3 次元モデルを描画するための機構があり，X3Dom に記述された 3 次元モデルを Canvas に描画している [8]．Andrioti らは WebRTC と X3Dom での研究を行っている [9]．WebGL では GPU による処理により，2Dcontext 高速な描画処理が可能となる反面，2Dcontext での実装に比べ開発コストが高くなる．多層レイヤー Canvas 同期機構は、描画処理を分割し，無駄な再描画処理を少なくすることが重要である．3D モデルの描画処理時間に応じて Canvas を分割することで，2 次元表現と同様に高速化が期待できる．

松村らは，HTML5 の SVG 要素を用いて描画の高速化を行った [10]．Canvas が 2 次元ビットマップ画像を描画するのに対して，SVG は 2 次元ベクター画像を描画する．Web アプリケーションのパフォーマンスの向上のためには，再配置、再描画回数が重要と述べられており，本研究においても描画に影響しない部分の最適化を行った．

8 おわりに

先行研究である DWLC と不定レイヤー Canvas を組み合わせた多層レイヤー構造のシステムを開発した．不定レイヤー分割に関する予備実験を行い，ユーザーからの操作に基づく描画処理の高速化と同時処理を実現した．これにより，更新頻度に依らず大量のオブジェクトに対してリアルタイムな描画を実現した．本稿では，描画処理におけるリアルタイム性に焦点を当てているが，同期処理には通信技術が欠かせない．本システムには，SocketIO を用いた非同期通信を用いている．レイヤー化の機構により描画処理のレスポンス速度が保証されているので，今後は通信処理の向上を検討している．

謝辞

本研究の一部は JSPS 科研費 JP15K00422，JP16K00420 の助成を受けたものです．

参考文献

[1] Shinya Katayama, Takushi Goda, Shun Shiramatsu, Tadachika Ozono, and Tora- matsu Shintani: On a Drawing-Frequency based Layered Canvas Mechanism for Collaborative Paper Editing Support Systems. International Journal of Networked and Distributed Computing, Vol.2, No.2, pp.9199, 2014.

[2] Shinya Katayama, Shun Shiramatsu, Tadachika Ozono, and Toramatsu Shintani: Generational Layered Canvas Mechanism for Collaborative Web Applications. In IIAI 3rd International Conference on Advanced Applied Informatics, pp.7075, 2014.

[3] Tadachika Ozono, Shun Shiramatsu, and Toramatsu Shintani: A Stable Layered Canvas Mechanism for Collaborative Web Applications. The 2015 IEEE/WIC/ACM International Conference on Web Intelligence(WI'15), pp. .101-106, 2015.

[4] W3C: HTML Canvas 2D Context. https://www.w3.org/TR/2dcontext/.

[5] Katayama S Y, Goda T, Shiramatsu S, et al: A Fast Synchronization Mechanism for Collaborative Web Applications Based on HTML5. Proceedings of the 14th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2013), pp.663-668, 2013.

[6] PDF. js: https://mozilla.github.io/pdf.js/.

[7] Kostas Kapetanakis, Spyros Panagiotakis, Athanasios G. Malamos: HTML5 and WebSockets; challenges in network 3D collaboration. Proceedings of the 17th Panhellenic Conference on Informatics(PCI'13), pp.33-38. 2013.

[8] WebGL: https://www.khronos.org/webgl/.

[9] Haroula Andrioti, Andreas Stamoulias, Kostas Kapetanakis, Spyros Panagiotakis, Athanasios G. Malamos: Integrating WebRTC and X3DOM; Bridging the Gap between Communications and Graphics. Proceedings of the 20th International Conference on 3D Web Technology (Web3D'15), pp.9-15. 2015.

[10] 松村 哲郎, 倉光 君郎: Web ブラウザにおけるビジュアルモデルの高速描画法の開発. 情報処理学会論文誌, vol.56, No.3, pp. 1039 - 1048, 2015.