Vol.2020-CG-179 No.2
2020/9/19

多種モデル描画のための階層的分散並列レンダリングサーバにおける深度情報の構成に対する検証と多数のクライアント下における負荷評価

奥村 直仁[1,a)]　齋藤 豪[1,b)]

概要：VR 上でコミュニケーションを行うアプリケーション等においてはリアルタイムで多数のモデルの描画を行う必要があるが, 低性能なユーザデバイスでは描画や通信のコスト面で問題になる. 石井らの手法では, クラウド上の複数の計算ノードを階層的に通信させ, 色情報と深度情報を描画・合成していくことで描画負荷を分散しながらクライアントへ描画結果を提供できることが提示された. 本研究では, 石井らが用いた深度情報のビット深度を実験によって検証し, さらに多数のクライアントが同時に存在する場合におけるコンテンツプロセスの送信負荷についての評価を行った.

キーワード：クラウドレンダリング, リアルタイムレンダリング

1. はじめに

VR 空間上でコミュニケーションを行うといったゲームなどにおいては, リアルタイムに様々な種類の 3D モデルを描画する必要がある. しかしながらモデルの種類や数の増加に伴って描画コストは高くなり, 一部のユーザデバイスでは性能が不足してしまうことがある. 低性能なユーザデバイスにおいてもリアルタイムに高品質な描画を行うための手段として, クラウド上のサーバが高負荷な描画を行うクラウドレンダリングの技術が注目を集めている.

クラウド上の複数の計算ノードを用いて特定の描画結果を生成するという分散並列レンダリングの手法は Molner ら [3] によって分類がなされており, これまでに巨大なモデルを事前に分割する方法 [4] や空間を直方体で分割する方法 [7] によって処理を分散させることが提案されてきた. しかし, これらはユーザが各々のアバターとしての 3D モデルをアップロードするような VR アプリケーションを想定すると, 規模を拡大させるのが容易でなくモデル情報をノード間で転送しなくてはならないといった欠点が生じる. このような背景を踏まえて新たに石井らは, 1 つのモデルに対して 1 つのサーバプロセスを割り当てた上で階層的に合成を行う分散並列レンダリング手法 [5], [6] を提案した. この手法では, 合成を行うプロセスに割り当てるコンテンツの数を調節することや合成の多段化を行うことで特定のプロセスへ負荷が集中することを減らし, 一定の台数効果を実現した.

本稿ではこの石井らの手法に対し, ノード間の通信で受け渡される色と深度についての情報のうち深度に関するデータの構成方法について, 視錐台の取り方および画素あたりのビット深度の条件を変え, 24bit 固定小数点数によるビット深度の有効性を検証し, また石井らの報告では未達であった複数のクライアントが同時に存在する場合における負荷評価を実施した.

2. 先行研究

この節では, 石井らの提案した階層的分散レンダリング手法 [5], [6] について説明する. この手法では, シーンの描画のために「コンテンツプロセス」と「マージプロセス」という 2 種類のプロセスを用意し, それらが通信を介して協調して描画を行う. クラウド上で図 1 のようにコンテンツプロセスとマージプロセスが階層的に通信を行い, ユーザデバイスに相当するクライアントはマージプロセスへ描画を要求することで描画結果をリアルタイムで取得する.

コンテンツプロセスは, 原則として一つのモデルの描画を行うプロセスであり, マージプロセスからクライアントのスクリーン情報と視点情報を受信すると所有するモデル

[1] School of Computing, Tokyo Institute of Technology
[a)] naohito@img.cs.titech.ac.jp
[b)] suguru@c.titech.ac.jp

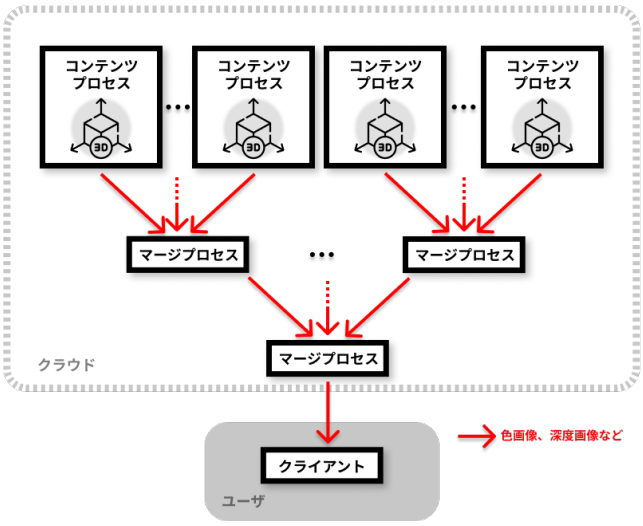


図 1　階層的なプロセスの通信

についてのみの描画を行い, その色画像 (図 3) と深度画像 (図 4) を送信する. 一方マージプロセスは他のプロセスに描画を依頼して, 受け取った描画結果を深度を考慮しながら合成するプロセスであり, 依頼先として予め複数のコンテンツプロセスまたは複数のマージプロセスを割り当てておく. マージプロセスは他のマージプロセスもしくはクライアントからスクリーン情報と視点情報を受信する.

このように 2 つの役割のプロセスがクラウド上で相互的に接続しており, クライアントはこの内のもっとも上流のマージプロセスへ描画を要求することで結果的に画面描画を得ることができる.

3. 実験と結果

3.1 TSUBAME3.0 を用いた実験環境

多数のプロセスを必要とする検証では, 東京工業大学の分散並列型スーパーコンピュータ TSUBAME3.0[2] を用いてモデルの描画を行った. TSUBAME3.0 の計算ノード 1 つあたりの構成は表 1 の通りで, 今回は 1 つの GPU に対して 1 つのプロセスを割り当てて実験を行った (すなわち, 各計算ノードは 4 つのプロセスをそれぞれ異なる GPU を指定して起動させた).

表 1　TSUBAME3.0 の計算ノード 1 つあたりの構成

OS	SUSE Linux Enterprise Server
Network	Intel Omni-Path HFI 100Gbps x4
CPU	Intel Xeon E5-2680 V4(14 core) x2
GPU	NVIDIA TESLA P100 (16GB,SXM2) x4

3.2 視錐台・深度情報の違いによる送信量の検証

石井らの構築したシステムにおいてコンテンツプロセスにて描画を行う際に用いる投影変換には, クライアントのカメラが定義する視錐台 (図 2 灰色の台形) から求める投影

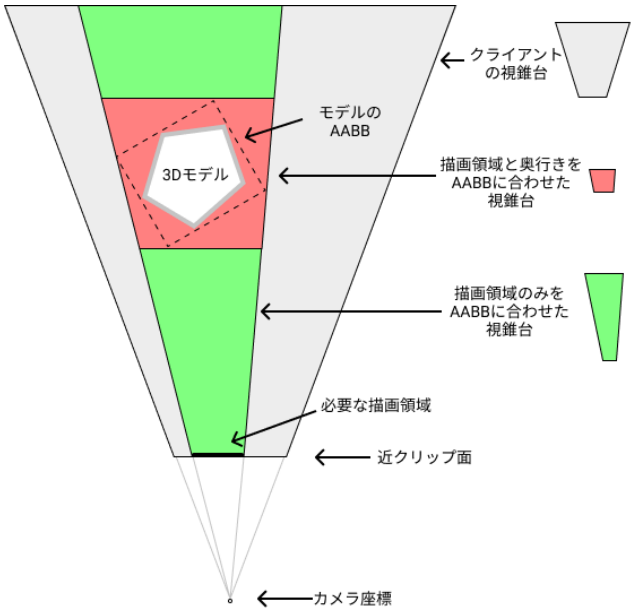
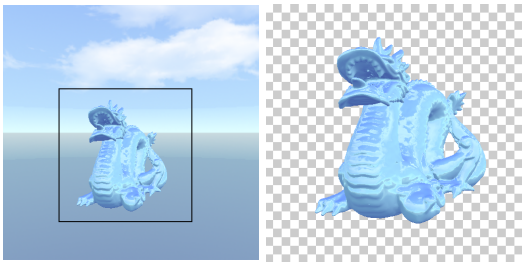


図 2　視錐台の上面図



(a)　(b)

図 3　(a) マージプロセスが合成した色画像, (b) コンテンツプロセスが送信する色画像

変換ではなく, 3D モデルの AABB から近似して描画領域と奥行きを小さくした部分視錐台 (図 2 赤色の台形) による投影変換を用いる. これにより, コンテンツプロセスが描き出す必要のある画像のサイズをスクリーンのサイズよりも小さくすることができるとともに, マージプロセスに送る深度情報の解像度を上げている. 今回の実験では, 奥行き方向はクライアントと一致させ画角の縮小のみを行った部分視錐台 (図 2 緑色の台形) による投影変換で描画と送信量の比較を行った. 以降単純のため, 図 2 の赤の視錐台を用いた投影変換を「奥行きを合わせた投影変換」, 緑の視錐台を用いた投影変換を「奥行きを合わせない投影変換」と呼ぶ.

また, マージプロセスでの画像の合成に用いる深度画像の値の表現には 32bit 浮動小数点数 [1] と 24bit 固定小数点数がある. 通常, 合成に必要な深度情報は「投影変換後に −1 から 1 の範囲で表される各ピクセルごとの深度値」であり, 図 5 に示した 32bit 浮動小数点数で表現すると図 4 の (c-1) および (c-2) のような深度画像が得られる. しかし石井らの手法では図 6 に示した 24bit の固定小数点数を用いて深度情報を構成し, RGB 各 8bit ずつの色画像と同様の

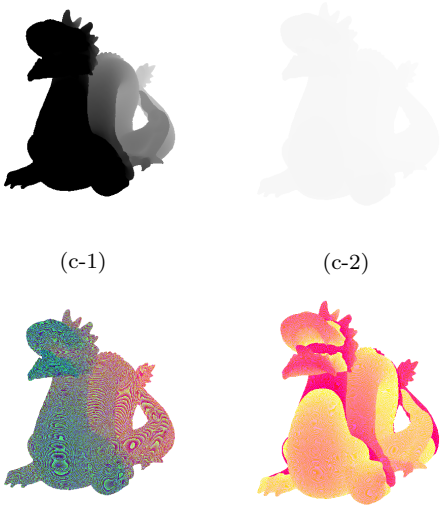


図 4 各条件でコンテンツプロセスが送信する深度画像. 条件と記号の対応は以下の通り.
(c-1) 奥行きを合わせた投影変換+32bit 浮動小数点数
(c-2) 奥行きを合わせない投影変換+32bit 浮動小数点数
(c-3) 奥行きを合わせた投影変換+24bit 固定小数点数
(c-4) 奥行きを合わせない投影変換+24bit 固定小数点数

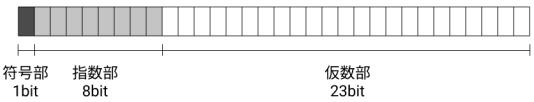


図 5 32bit 浮動小数点数による深度の表現

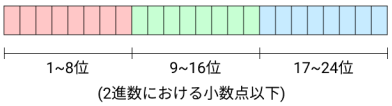


図 6 24bit 固定小数点数による深度の表現

フォーマットで送信を行っていた (ここで深度値の −1 から 1 の範囲を縮小して, 0 以上 1 未満の範囲に変換を行っている). この場合の深度画像は図 4(c-3) および (c-4) であり, 小数点以下 17 位から 24 位までの範囲が 11111111 から 00000000 に桁上がりする際などに生じる独特な縞模様が見て取れる. この深度情報の表現では, 深度画像の送信が普通の色画像として扱うことができるという利点があるが, 32bit 浮動小数点数と 24bit 固定小数点数の間には値の表現に差があるため, 深度の比較において間違いが生じるパターンが異なると考えられる.

このような背景のもと, 一つ目の検証として 64 種のモデル**??**の同時描画をそれぞれ視錐台の取り方と深度情報のエンコードの条件を変え, 通信量を評価した. 描画を行ったシーンは図 7 の通りであり, 原点を中心とした格子のように 64 体を並べた. この検証では TSUBAME3.0(表 1) を用いて実施し, 稼働させたプロセスの数はコンテンツプロセスがモデルと同数の 64 個, コンテンツプロセス 8 個ずつを割り当てられたマージプロセスが 8 個, それらのマージプロセスの出力をさらに合成する多段マージプロセスが 1 個, クライアント用のプロセスが 1 個である.

表 2 実験に用いたモデルと描画形式

モデル (1 体あたり)	
ポリゴン数	61666 ポリゴン
関節数	59 関節
アニメーション	57 の関節駆動
描画方式	
描画形式	Torrance-Sparrow model + IBL
アンチエイリアス	4xMSAA
解像度	512x512

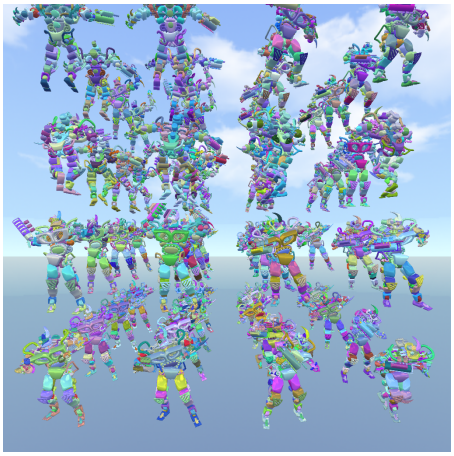


図 7 描画を行った 64 種のモデルを描画したシーン

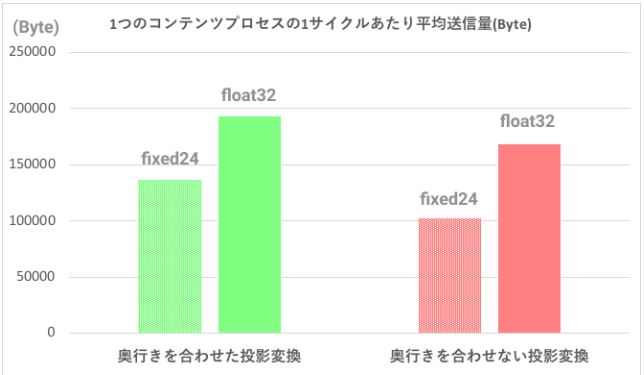


図 8 視錐台の取り方および深度情報の構成の異なる条件下によるコンテンツプロセスの送信量の比較

(視錐台の取り方・2 通り)×(深度情報の構成方法・2 通り) の条件間で比較を行ったところ, 図 8 のような結果が得られた. これは自明の範囲ではあるが, 24bit の固定小数

点数と 32bit の浮動小数点数では使用する bit 数が異なるため, 24bit 固定長の場合にはより少ない送信量に抑えることができることが示された. 奥行きに合わせた投影変換を行った場合と奥行きを合わせない投影変換を行った場合の間にはわずかな差が見られるが, 今回の実装ではプロセス間で通信される画像は非圧縮で送信されるためサイズは同一になると考えられ, これは実装上の問題と見るべきだろう.

続いて二つ目の検証として, 深度を考慮した合成が間違いを起こす例について具体的な場合を取り上げ比較した. 描画するシーンとして, 図 9 に示すような緑と青の 2 枚の板状のモデルが非常に近接して並んでいる様子を斜めからカメラを向けるというものを用意する. 描画にあたり, モデルそれぞれを担当する 2 つのコンテンツプロセス, それらをマージするマージプロセスが 1 つ, そしてマージプロセスへ要求を行うクライアント用のプロセスを用意し, 計 4 つのプロセスを起動した. なお, 2 つのコンテンツプロセスの深度画像の表現は共に同一のものを用いた. このようなシーンを用いた理由としては, 与えられた深度の値の精度が不十分であるとき, 画素ごとの深度値が衝突してしまい (すなわち 2 枚の板の間の深度差が無視され), 出力される画素の色を取り違えてしまうからである. 図 9 のシーンにおいて期待される正しい出力画像は, 今回カメラは緑の板の側に配置したため, 緑の配色が前面に現れるというものである.

二つ目の検証の結果を図 10 に示す. (X)-color は合成によって得られた出力の色画像であり, (X)-depth は緑の板を担当するコンテンツプロセスにおける深度画像である. 図に示す通り, (B) 奥行きを合わせない投影変換を用い 32bit 浮動小数点数で深度画像を構成した場合と, (D) 奥行きを合わせない投影変換を用い 24bit 固定小数点数で深度画像を構成した場合の 2 通りにおいて, 青色の板が緑色の板の前に出現してしまっている画素が見られた. これは, 画素の深度値が双方のモデルの間で差が消失してしまったために, 深度値の比較に失敗している現象である. 投影変換の違いに関してこの結果からわかることは, 奥行き方向の大きさが適切に縮小されていることが深度合成を安定させるために必要であり, AABB を描画領域の方向と奥行きの方向の両方に縮小する投影変換には一定の効果がみられるということである. また深度情報の表現については, (A) と (C), もしくは (B) と (D) の間に顕著な差が見られなかったためどちらの手法を用いても深度合成に差はあまり見られないといえる ((B) と (D) の間には深度比較を誤った画素の数に差が見られるが, これは角度をわずかにずらすとそのような画素の位置や数が変動するため, どちらの方が間違いが多いということは伺えなかった). 以上の結果を鑑みると, 深度情報の表現には 32bit 浮動小数点数と 24bit 固定小数点数の間に差は見られず, 投影変換には奥行き方向を合わせた図 2 の赤の視錐台を用いる方が通信コストの面で有利であると結論できる.

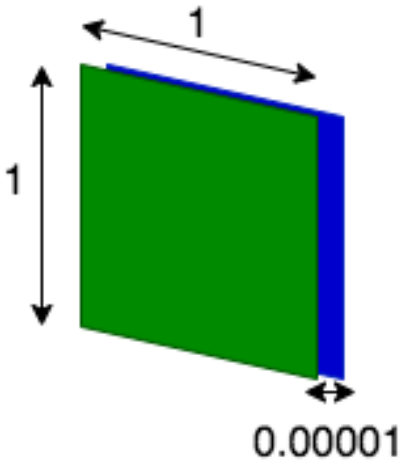


図 9 板状のモデルが近接し, 深度比較の誤りが生じやすいシーン

3.3 多数クライアントにおける負荷評価

表 3 実験に用いたモデルと描画形式

モデル (1 体あたり)	
ポリゴン数	4672 ポリゴン
描画方式	
描画形式	Torrance-Sparrow model + IBL
アンチエイリアス	4xMSAA
解像度	1024x1024

石井らの報告した検証では全て単一のクライアント, すなわち単一の視点についての描画を評価していた. しかし, 多人数が同時に同じシーンに存在するアプリケーションの実現に向けてはより多くのクライアントにリアルタイムで対応できる必要がある. 今回の検証では, 複数のクライアントが同時に接続した際にコンテンツプロセスを共有するアーキテクチャを構成し, その負荷評価を行った. 描画したシーンは図 12 のように 16 個の人型モデル 11 が並べられ, クライアント数が 1, 2, 4, 8, 16 と増えていくにしたがって外側のモデルからクライアントとカメラ座標を割り当てるように設定した. 各モデルは移動を行わないが, 視点のみゆっくり回転させ, 時間とともに視界に映るモデル数が変化するようにした. マージプロセスにおける遅延が顕著に現れないように, 石井らの報告 [5] にてマージプロセスの 1 フレームあたりの処理時間が $\frac{1}{60}$ 秒を上回らないとされる 1 つのマージプロセスあたり 4 つのコンテンツプロセス (もしくはマージプロセス) の割り当てを設定して実験を行った. 実験に必要なプロセスの数は, コンテンツプロセスがモデルと同数の 16 個, マージプロセスが 1 クライアントごとに 5 個, クライアント用のプロセスがクライアントと同

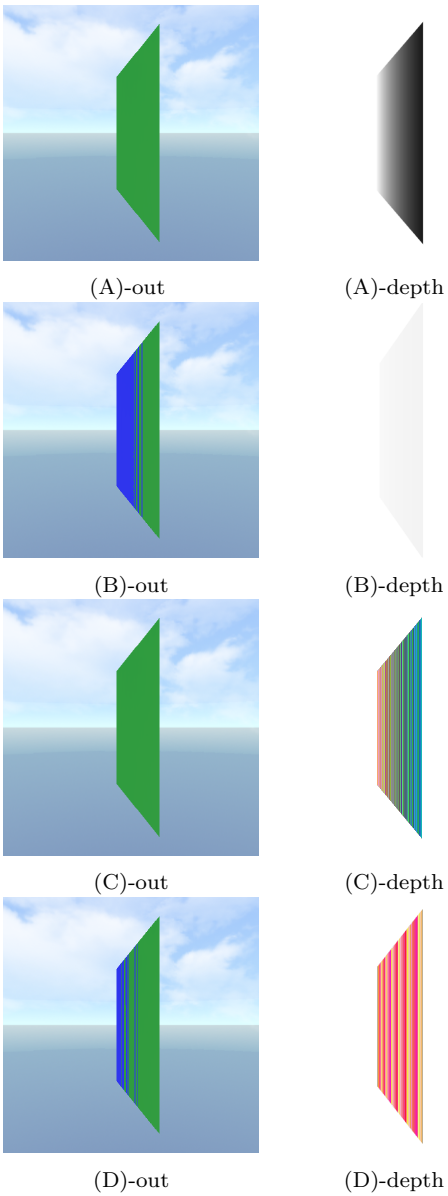


図 10 (A) 奥行きを合わせた投影変換+32bit 浮動小数点数
(B) 奥行きを合わせない投影変換+32bit 浮動小数点数
(C) 奥行きを合わせた投影変換+24bit 固定小数点数
(D) 奥行きを合わせない投影変換+24bit 固定小数点数

じ個数である.

コンテンツプロセスは内部的なループにおいて, 接続されている全てのプロセスからの描画要求を 1 回ずつ処理して結果を送り返すということを 1 サイクルで行っている. そのため, 1 サイクルあたりの平均の送信量・処理時間を評価することで, 通信コストおよび時間的なコストを評価することができる. コンテンツプロセスは複数のクライアント間で共有されるため, コンテンツプロセスは 1 サイクルにクライアントの数だけ描画と送信を実行することとなる.

図 13, 図 14 はそれぞれクライアント数と 1 つのコンテンツプロセスの 1 サイクルあたりの「送信量」「送信処理時間」のグラフを示している. これらのデータは, いずれも図 12 の四隅のうちの一箇所に位置するモデルに割り当てたコンテンツプロセスについて, 時間平均をとったものである. ここからまず, クライアント数が倍々に増加するにしたがって, コンテンツプロセスが配信する必要のあるデータの量はおよそ倍々に増えてゆくといえる. また, それらの送信を行うのにかかる時間は, クライアント数が 4 程度まではおよそ一定だがそれ以降は大きく時間がかかるようになることがわかる. さらに描画処理が占める時間を別途調べたところ, 描画処理は 16 個のクライアントについてのレンダリングでも十分に実行が可能であり, より多くのモデルやより複雑なモデルに関して対応する余地があるとわかった.

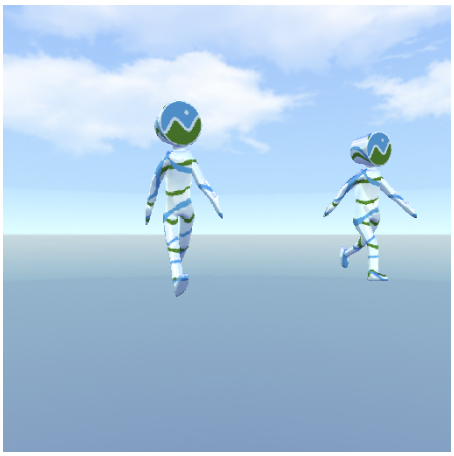


図 11 描画に用いた人型モデル

コンテンツプロセスの 1 サイクルあたりの「送信量」を時系列でグラフ化したものが図 15, 図 16 である. これらは, 図 12 の四隅のうちの一箇所に位置するモデルに割り当てたコンテンツプロセスについての測定結果である. 整列しているモデルの中でも端に位置するモデルを担当するコンテンツプロセスは, 各カメラが一斉に回転する (図 12 参照) ことによって, 一定の周期で多くのクライアントの視界に映り込む時間とほとんどのクライアントで映らない時間が交互に発生する. 図 15, 図 16 の周期性はそのことに起因しており, 送信量が急激に上昇する期間とほとんど送信量が低い状態で抑えられる期間が現れる. この 2 つのグラフの 1 周期にかかるサイクル数が異なるのは, 1 サイクルにかかる時間が長くなってしまっているためである. 長くかかる原因は送信量の増加が原因と考えられ, クライアントの FPS に間に合うように送信量の削減が今後の課題である.

全体を通して, 60fps のゲームを想定した際の 1 サイクルあたり $\frac{1}{60}$ 秒以内に収めるという目標を達成できたのは現在のところ 4 個のクライアントの場合までであり, 16 個のクライアントの場合は 1 サイクルに 200ms 程度かかってしまうこともあった. したがって, 現在の実装によってリアルタイムのクラウドレンダリングを阻害する主な原因となりうるものは, クライアント間で共有されるコンテンツプロセスの通信処理である. 石井らの設計としては, コンテ

ンツプロセスはモデルの数で用意し, マージプロセスは適切に数を増やしたり階層を増やすことで負荷分散効果を得るというものであったが, コンテンツプロセスへの負荷の集中は複数のクライアントの同時接続を前提とすると避けることができないため, 今後新たにコンテンツプロセスへの最適化や複数プロセスへの分割を考える必要がある. また, プロセス間で送受信する色画像と深度画像は, 適切な圧縮処理をかけることでよりコンパクトに伝送することができると考えられるが, 深度画像には通常の画像に施されるような色の周波数成分を考慮した不可逆圧縮をかけることは適切ではなく, よりこの問題に適した圧縮法の検討が有効である.

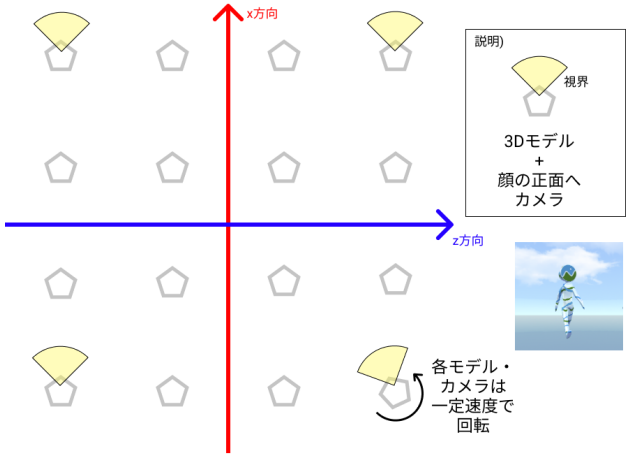


図 12 複数のクライアントにおけるシーンの設定
(例:4 クライアントの場合)

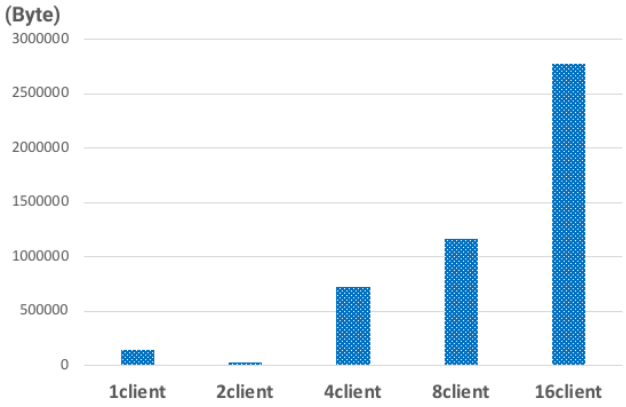


図 13 クライアント数とコンテンツプロセスの 1 サイクルあたり送信量 (Byte)

4. 結論と今後の課題

本稿では, 石井らが用いた「奥行きを合わせた投影変換」と 24bit 固定小数点数による深度表現について検証的な評価を与え, 同時に複数のクライアントがコンテンツプロセスを共有するように拡張したアーキテクチャにおいてデータ転送の負荷上昇によってフレーム更新が大きく遅れてしまうという現時点での問題点を示した.

今後の課題としては, プロセス間を送受信されるデータをより小さくするために深度情報に対応した圧縮や不要なオーバーヘッドの削除を施し, また同時にコンテンツプロセスについてもクラウドの台数効果が得られるような設計を行うことが挙げられる.

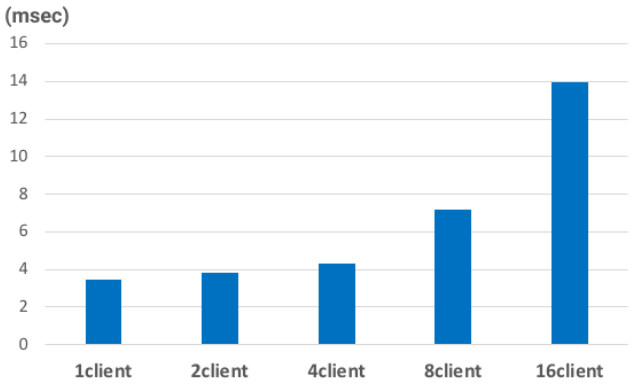


図 14 クライアント数とコンテンツプロセスの 1 サイクルあたり送信処理時間

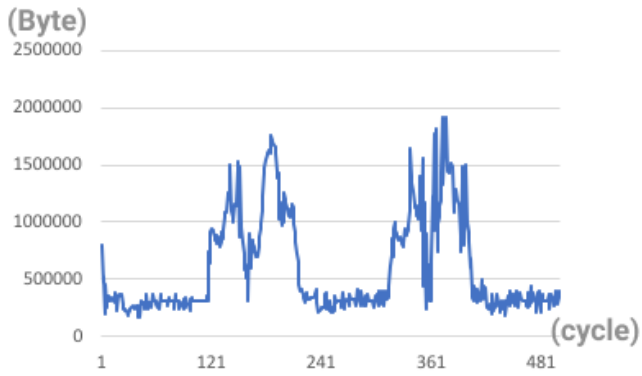


図 15 8 クライアントの場合におけるコンテンツプロセスの送信量の時間変化

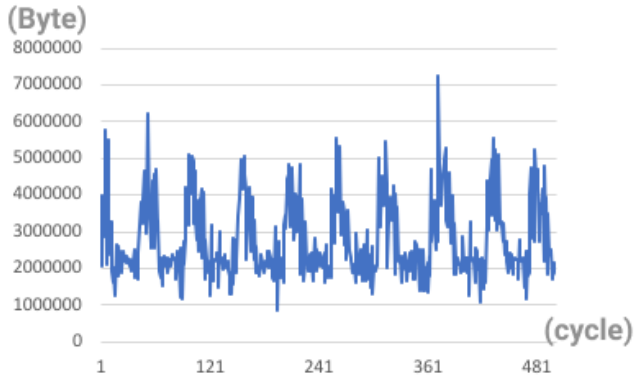


図 16 16 クライアントの場合におけるコンテンツプロセスの送信量の時間変化

参考文献

[1] : IEEE Standard for Floating-Point Arithmetic, *IEEE Std 754-2008*, pp. 1–70 (2008).
[2] Matsuoka, S., Endo, T., Nukada, A., Miura, S., Nomura, A., Sato, H., Jitsumoto, H. and Drozd, A.: Overview of TSUBAME3.0 Green Cloud Supercomputer for Convergence of HPC AI and Big-Data, *e-Science Journal*, Vol. 16, pp. 2–9 (2017).
[3] Molnar, S., Cox, M., Ellsworth, D. and Fuchs, H.: A sorting classification of parallel rendering, *IEEE CG&A*, Vol. 14, No. 4, pp. 23–32 (online), DOI: 10.1109/38.291528 (1994).
[4] Samanta, R., Funkhouser, T., Li, K. and Singh, J. P.: Hybrid Sort-First and Sort-Last Parallel Rendering with a Cluster of PCs, *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, New York, NY, USA, Association for Computing Machinery, pp. 97–108 (online), DOI: 10.1145/346876.348237 (2000).
[5] 石井翔, 齋藤豪: 多種モデルの描画に特化したスケーラブルな複数ノードによる並列レンダリングの台数効果の検証，情報処理学会第 82 回全国大会, 1ZC-04, 2pages (2020).
[6] 石井翔, 齊藤豪: 多種多様なコンテンツへのスケーリングに特化したパラレルレンダリングを用いたサーバーサイドレンダリングアーキテクチャによる合成 CG 作成法，情報処理学会研究報告（CG）2019-CG-173,5，pp. 1–7 (2019).
[7] 渡部雅人, 齋藤豪，中嶋正之: 空間領域分割による分散レンダリングにおける画像合成並列化手法，電子情報通信学会 2008 総合大会 D-11-104 (2008).