Vol.2019-CG-173 No.5
2019/3/5

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

石井 翔[1,a)]　齋藤 豪[1,b)]

概要：通信速度の向上やエッジコンピューティング環境の整備などの技術的進歩に伴い、ユーザモバイル端末の多様化などの背景からサーバーサイドレンダリングが再び注目を集めている。ユーザの数によってスケーリングする構造が研究されている一方で、ユーザがモデルをアップロードするなど事前に予期できない多種のモデルが介在する環境を効率よくスケーリングする方法は提案されていない。本手法ではユーザのアップロードするコンテンツのローカル空間のみでの空間領域分割パラレルレンダリング手法おを提案し、多種のモデルに対してスケール可能なレンダリング手法および、サーバーサイドアーキテクチャを提案する。

A method to generate mixed CG images from the server side rendering architecture using parallel rendering for adopting various contents

1. 背景

現在規格化が進行中である第五世代移動通信システム”5G”の規格化によって高速、大容量、高信頼なモバイル通信が可能になる。この技術的進歩により、高度なグラフィクスを用いているゲームなどのアプリケーションでも、ユーザデバイスの性能の差に大きく影響されることなく、サーバ上で描画された結果をスマートフォンなどのモバイル端末にストリーミング配信することによって提供可能になる。

この様な状況下では、ユーザ端末の性能の差を考慮することが必要なくなるため、フレームレートの低下や局所的なフリーズなどの原因は、最早クライアントの性能の問題であり、ユーザの問題であるという認識から、描画サーバの問題であり、サービス提供者の問題であることに変化すると考えられる。すなわち、描画サーバに必要とされる性能を推定することが大きく求められてくるのである。多くのゲームではゲームの作成者はあらかじめ描画されうるモデルの量やタイミングなどを知ることが可能である。従って、サーバサイドレンダリングで配信する際には、最大に負荷が掛かるタイミングを想定することによって必要なサーバの性能をある程度は試算することが可能である。一方、ユーザ自身がモデルをアップロードする形式のアプリケーション (以降、メタバース的アプリケーションと呼称する)、参加者の数などによって大きく描画負荷が変動するようなアプリケーションでは、事前にサーバにどの程度の処理能力が存在すれば十分にユーザに配信可能であるのかの推定することが難しい。そのため、アプリケーション内で利用できるモデルの大きさや参加人数等に大きな制限を設ける必要や、必要以上に高い性能の描画サーバを確保する必要がある。特にメタバース的アプリケーションでは、モデルデータ、モーションデータなどの静的なデータのみでなく、ユーザ毎のオリジナルのシェーディングや、スクリプトなどもアップロードでき、独自の動的な処理を実行できる場合も存在する。この様な状況での描画サーバに必要な性能を推測することはさらに難しくなり、動的に必要な性能のサーバを割り当てるようなシステムを考案しなければならない。

サーバ環境においてスケーリングを考える際、1 台の性能を上げることによるスケーリングであるスケールアップよ

[1] 東京工業大学
Tokyo Institute of Technology,2-1-1 Meguro, Tokyo 152–8550, Japan
[a)] kakeru@img.cs.titech.ac.jp
[b)] suguru@c.titech.ac.jp

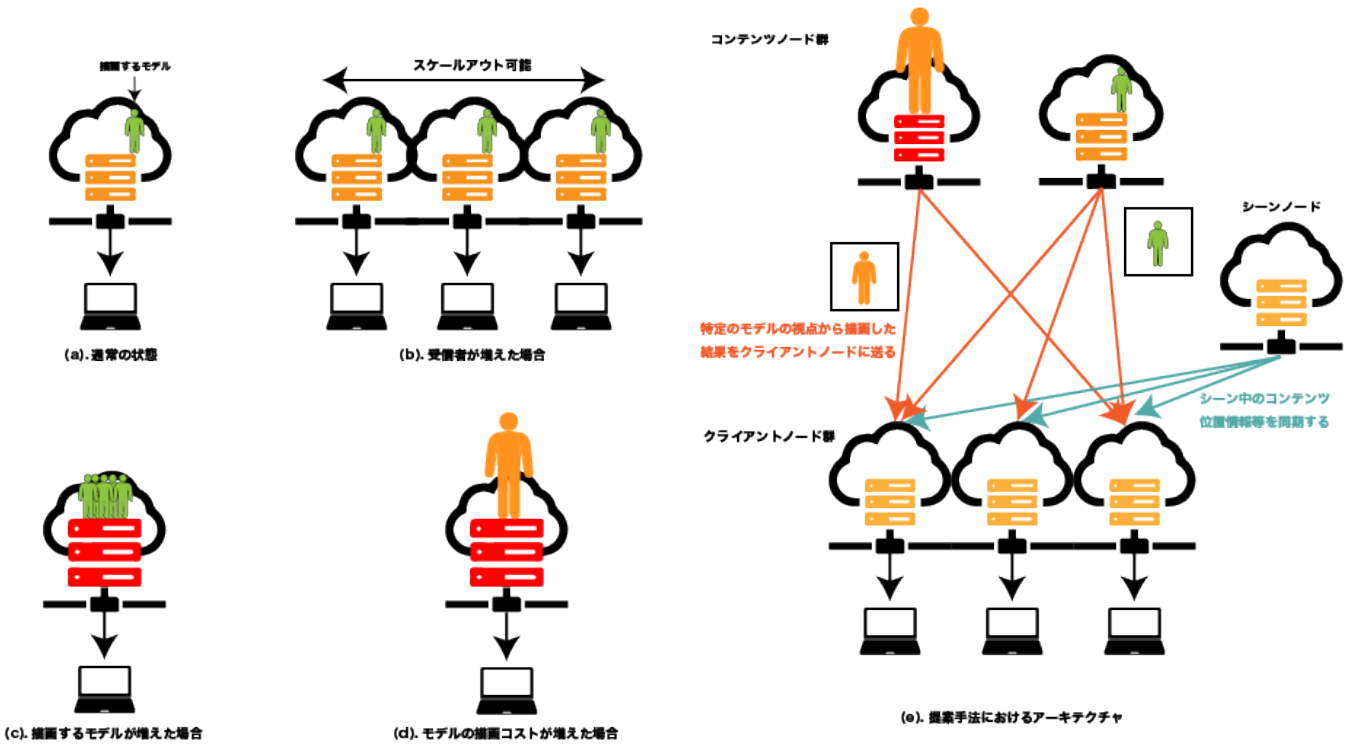

図 1 状況に応じた動的スケーリングの例 (左) と本提案のシステム俯瞰図 (右)

りも、台数を増やすスケーリングであるスケールアウトであることの方が望ましい。特に描画処理を実行中のサーバをスケールアップすることは多くの場合ダウンタイムを伴うため動的にスケーリングの手段として不適である。一方でスケールアウトの場合は、追加のサーバを適切なユーザに割り付けることが出来ることから、特にサーバサイドレンダリングを行う際に動的にスケーリングを行うことができる。

図 1 左はサーバサイドレンダリングでスケーリングが必要な主な場合の図である。クライアントが複数増えた場合その分の描画サーバを増やしてスケールアウト可能である (図 1(b))。逆にユーザがログアウト等を経て映像を配信する必要のなくなった際にはサーバを減らす (スケールイン) ことによってコストを削減する事が可能である。描画すべきモデルが増えた場合、簡易的な方法では単に 1 描画サーバあたりの性能を上げてスケールアップせざるを得ない (図 1(c))。複雑で描画負荷の高いモデルの描画が新たに必要になった場合、簡易的な手法でスケーリングさせるには図 1(c) と同様に 1 描画サーバあたりの性能を上げてスケールアップしなければならない (図 1(d))。従って、図 1(c) や図 1(d) のような状況が起き得ると考えられる場合には実際に必要なサーバよりも性能のよい描画サーバを設けなければならない。

本提案では、以上で挙げたようなサーバサイドレンダリングを行う際のスケールアウト・スケールアップの問題に着目し、ユーザごとに描画を分散するという図 1(b) ナイーブなサーバーサイドレンダリングの構成とは異なる、3 種類のノードからなるサーバサイドレンダリングの手法を提案する。特に、コンテンツ毎の空間を考えてコンテンツ毎にサーバを割り付けるつけることのできる「コンテンツノード」、これらの結果を深度を考慮し合成する「クライアントノード」に分割することによって、スケールアウトによる対処で多くの場合スケーリング可能であり、事前に必要となる性能が分かる対象に対してはスケールアップによって対処可能な構成となる。

2. 関連研究

2.1 パラレルレンダリング

計算量が非常に大きくなりやすいコンピュータ・グラフィクスの分野ではその演算に並列計算の概念を導入すること [1][2][3] によりリアルタイムな描画やより高精細な画像を現実的な時間で描画することを実現してきた。パラレルレンダリングの研究では、主にどのような単位で分散を行うのかといった点が問題となる。

パラレルレンダリングの研究の初期においては、1 台のコンピュータ内で複数個のコアを用いて最終的な生成画像を高速に生成する手法が研究された [4][5]。一台あたりの描画能力が向上すると、次第に複数台の汎用コンピュータを用いてさらに大規模な描画能力が求められるような対象を描画することが求められるようになった。このように、ネットワーク上の複数台のコンピュータを用いて大規模なデータを描画する手法もパラレルレンダリングの一種とし

て知られている。

スクリーン単位での分割を主として、その他にもサブピクセル単位での分割 [6] や時間単位での分割 [7] など分割方法によって大きくその構成が異なる。ここでは、本手法に比較的近いと思われるスクリーン単位での分割方法の例及び、オブジェクト単位での分割の例を挙げる。

Smanta 等の研究 (2000 年)[8] ではスクリーンを数個の領域に分割して複数台の描画サーバにそれぞれの領域を描画させる手法を提案した。特にこの手法では、画面上の領域分割を、その領域を占める物体のポリゴンの多さなどを推定することにより動的に領域の大きさを定めることを提案した。これにより複数台の描画サーバで均一に描画負荷を担当することが出来るが、事前にモデルを分解して描画負荷を見積もらなければならない

Correa らの研究 (2002 年)[9] では非常に大規模な発電プラントの 3D 図面を十分にインタラクティヴな速度で描画するためにパラレルレンダリングを用いている。Correa らはあらかじめ描画対象となる画面を複数の領域に分割し、それぞれに対応した描画サーバを割り当てることにより高速化を行っている。

これらのスクリーン画面を分割して複数台の描画サーバを用いて描画する方法は描画時に映る可能性のあるデータは全ての描画サーバがアクセス可能でなければならない。そのため、あらかじめ描画されるモデルが全て決まっていないために、モデルのデータを各描画サーバーにコピーしておくことなどが行えない場合には活用が難しい。さらに、これらの手法は描画時にどのスクリーンに対応するか高速に判定を行うために事前処理を行っている。例えば Correa らの例では八分木を用いて空間をあらかじめ分割し、対応するメッシュや頂点を定めている。多くの場合静的なモデルにたいしてこのような八分木を割り当てるアプローチは動作するが、細かい部品が大きく動くようなシーンに対しては動的に八分木を更新しなければならずオーバーヘッドが大きい。

スクリーン単位での分割ではない手法の一つとして、三次元空間での分割を行うことによって、空間毎に担当する描画サーバを分割する手法がある。渡辺らの研究 (2008 年)[10] では、描画対象となる空間を分割し複数台のサーバに分散を行うことによって、事前処理なく多量のモデルを描画することを達成した。この手法では、ユーザ視点から撮影した各空間の表面の画像を再帰的に取得することによって結果となる画像を得ている。ある空間中に存在するモデルの量が一定の場合はユーザの動きに問わず、描画にかかるコストが一定である。しかし、この場合でもある空間で表示されるモデルが増えた場合に描画を間に合わせることが出来なくなった場合には、対応した空間の描画サーバーをスケールアップする必要性が生じる。

以上のように、突然描画負荷の高いモデルが出現した場合や、参加者のモデルが増えた場合に対応できるパラレルレンダリング手法は未だ提案されていない。我々の手法はこのような負荷の突然の変化に対して適切なノードを追加することによって対応することができ、描画に必要なモデルの量などが大きい場合においても十分に台数効果が発揮し続けることができるような構造のパラレルレンダリングを提案する。

3. 提案手法

3.1 概要

図 1 に示す構造が我々の提案手法のパラレルレンダリングアーキテクチャである。主に 3 種類のノードによって構成されるパラレルレンダリングアーキテクチャであり、各ノードの予め読み込みを行うべきデータと毎フレーム発生する入出力及び計算内容を表 1 に示す。各モデル毎に与えられた「コンテンツノード」がそのコンテンツだけを効率的に求められた視点から描画を行い、各ユーザ毎に与えられた「クライアントノード」がこれらコンテンツノードより描画結果画像及び深度画像を取得し深度を考慮した合成を行うことによりユーザに配信する最終的な映像を送信し、「シーンノード」が仮想環境全体の管理を行う。

3.2 コンテンツノード

コンテンツノードはクライアントノードから送られる描画リクエストによって描画を行う。この時、クライアントノードからユーザのカメラの姿勢行列及び射影行列とユーザデバイスでの表示解像度を受け取る。これらの入力から、描画結果及び、深度を出力としてクライアントノードに返信する (表 1)。

このために最初のモデルが映りうるユーザのカメラ視点から見たときの視錐台を求める。この視錐台の中でコンテンツが十分に収まる部分空間 (図 2、以後「コンテンツ空間」と呼ぶ) を以下のように求める。コンテンツノードはそのローカル座標系においてモデルを包含する最小の直方体を保持しており、直方体を構成する各頂点の座標を、ユーザ視点のカメラ行列によってカメラ座標系に変換された点の座標の集合を A と置く。式 (1) から式 (6) によりこの A から 6 頂点を選ぶ。

$$\mathbf{f}' = \arg\max_{\mathbf{p}\in A} \mathbf{p}.z \tag{1}$$

$$\mathbf{n}' = \arg\min_{\mathbf{p}\in A} \mathbf{p}.z \tag{2}$$

$$\mathbf{l}' = \arg\min_{\mathbf{p}\in A} \frac{\mathbf{p}.x}{\sqrt{\mathbf{p}.x^2 + \mathbf{p}.z^2}} \tag{3}$$

$$\mathbf{r}' = \arg\max_{\mathbf{p}\in A} \frac{\mathbf{p}.x}{\sqrt{\mathbf{p}.x^2 + \mathbf{p}.z^2}} \tag{4}$$

$$\mathbf{b}' = \arg\min_{\mathbf{p}\in A} \frac{\mathbf{p}.y}{\sqrt{\mathbf{p}.y^2 + \mathbf{p}.z^2}} \tag{5}$$

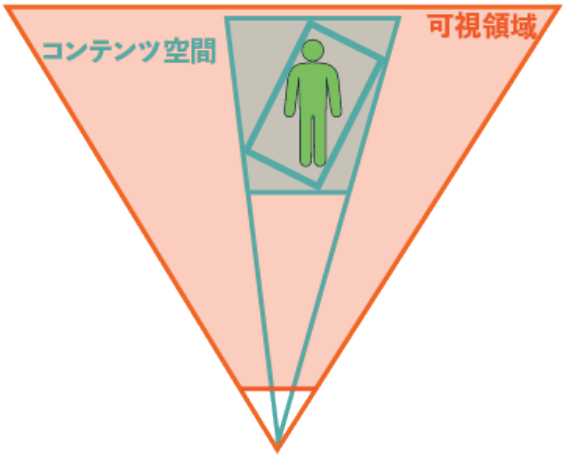

図 2　コンテンツ空間のイメージ

$$\mathbf{t}' = \arg\max_{\mathbf{p}\in A} \frac{\mathbf{p}.y}{\sqrt{\mathbf{p}.y^2 + \mathbf{p}.z^2}} \tag{6}$$

(ここで、arg min, arg max は最大もしくは最小を取るような点が複数点存在した場合にはその値を取る引数のうち任意の一つを取るものとする)

次に 6 頂点から、以下のパラメータを求め実際に用いるコンテンツ空間の射影行列 P_c を求める。

$$l = \frac{\mathbf{n}'.z}{\mathbf{l}'.z}\mathbf{l}'.x \tag{7}$$

$$r = \frac{\mathbf{n}'.z}{\mathbf{r}'.z}\mathbf{r}'.x \tag{8}$$

$$b = \frac{\mathbf{n}'.z}{\mathbf{b}'.z}\mathbf{b}'.y \tag{9}$$

$$t = \frac{\mathbf{n}'.z}{\mathbf{t}'.z}\mathbf{t}'.y \tag{10}$$

$$P_c = \begin{pmatrix} \frac{2\mathbf{n}'.z}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2\mathbf{n}'.z}{t-b} & \frac{b+t}{t-b} & 0 \\ 0 & 0 & \frac{\mathbf{f}'.z+\mathbf{n}'.z}{\mathbf{f}'.z-\mathbf{n}'.z} & \frac{2\mathbf{n}'.z\cdot\mathbf{f}'.z}{\mathbf{f}'.z-\mathbf{n}'.z} \\ 0 & 0 & -1 & 0 \end{pmatrix} \tag{11}$$

さらに、実際に転送するテクスチャの大きさを求めるために、スクリーン空間上で射影面の大きさを求める。クライアントノードで用いるユーザのカメラの用いる射影行列を P とすると、

$$\mathbf{u} \in \{(-1,1,1,1)^T, (1,1,1,1)^T, (-1,-1,1,1)^T, (1,-1,1,1)^T\} \tag{12}$$

$$\mathbf{v} = PP_c^{-1}\mathbf{u} \tag{13}$$

によって求まる $\mathbf{v}$ が実際にクライアントノードで描画される射影面となるので、画面解像度と掛け合わせることによって描画結果の中での画像の大きさがわかる。クライアントノードから受信したカメラ行列とコンテンツ描画用の射影行列である P_c を用いて通常のレンダリング、深度マップの出力を行い、クライアントノードに送信する。

以上のような仕組みに則ってコンテンツの映像、深度を配信することによって、担当するモデル以外のモデルを保持する必要がない。

3.3　クライアントノード

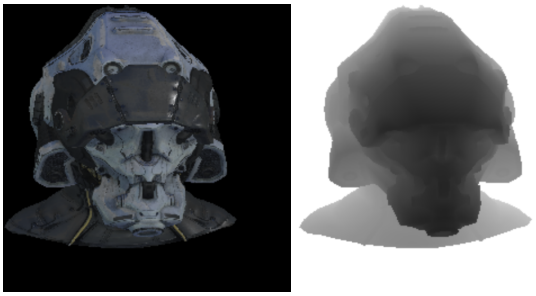

図 3　送られるコンテンツ空間の部分画像と深度画像

クライアントノードは、キャラクターの操作などのユーザ入力を受信し、これに合わせてカメラやコンテンツの位置情報を更新する。各コンテンツに対応するコンテンツサーバに対して、カメラの姿勢行列、射影行列、描画解像度を送信し、描画結果及び深度画像を受信する。受信した画像はコンテンツ空間の近クリップ面である射影面に置いた平面に対して貼り付け表示する。この際、この平面を構成する頂点はクライアントノード上での射影行列を施したあとの座標は式 (13) によって求めたものと等しい。

図 4 は、射影面がわかりやすくなるように赤く色を付けた結果である。

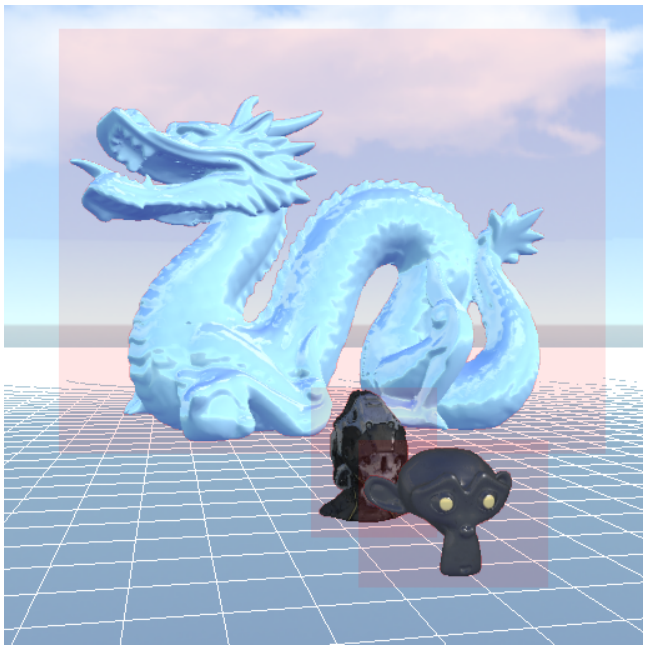

図 4　実際に合成されたコンテンツノードから送られた結果

他のコンテンツノードから受信した部分画像とそれぞれ

の深さを考慮した合成を行うために各コンテンツノードから受信した深度画像を合成する。式 (13) の座標 $\mathbf{v}.x, \mathbf{v}.y$ において深度値が d である場合以下の式によってクライアントノード上の深度に変換することが可能である。

$$\mathbf{v}_d = \{\mathbf{v}.x, \mathbf{v}.y, d, 1\} \tag{14}$$

$$\mathbf{d} = PVP_c^{-1}\mathbf{v}_d \tag{15}$$

$$depth = \mathbf{d}.z/\mathbf{d}.w \tag{16}$$

実際の実装ではこのような値の計算を結果画像を合成するフラグメントシェーダで行うことによって、深度値の更新及び結果画像の書き込みを同時に行い、深さを考慮したコンテンツの結果画像の合成が可能となる。AABB の深さ順にソートして描画するのみでは、コンテンツ同士の AABB が重なってしまうような場合は単に遠景からテクスチャを配置するだけでは描画することができない。それぞれが描画結果以外に深度 (図 3) を受取り、クライアントノード上の深度バッファにマージすることによって、AABB が重なっている状況下でも正しい結果画像を得ることができる。

4. 実験結果

今回は実際に想定するような複数台におけるパラレルレンダリングではなく、同一 PC 上にコンテンツノードを 3 つ、シーンノードを兼ねたクライアントノードを 1 つ建てることにより実験を行った。それぞれはプロセスとして独立しており、ソケット通信を経てテクスチャのデータをやりとりすることによって最終的な画像を得ている。コンテンツそれぞれは環境マップ及び予め方向が決定されている方向ライトによって照らされ、Cook-Torrance モデル [11] によってシェーディングされている。1 回反射のみが考慮されており、複数モデルにより生まれる反射や大域照明などは含まれていない。また、ドロップシャドウは考慮されていない。以上のように、今回の実験ではシェーディングに必要な入力は予めコンテンツノードが知りうる入力のみによって完結している。

図 6 及び 8 に 3 つのモデルが存在する空間をクライアントノードが描画した結果を示す。また、この映像の合成に使われた、コンテンツノードからクライアントノードに送信された画像を図 5、7 に示す。図 5 の黒い枠の横幅が最終的に生成される図 6 の横幅と同一であり、実際には結果画像の色の占める領域だけのデータが送信されている。図 8 に対する図 7 も同様である。コンテンツ空間の計算を適切に行うことにより、図 5、7 に示すように実際にシーンノードの必要とするデータのみの転送でよく、転送にかかる通信帯域量が遠くに行けば行くほど小さくなり、斜めにみえるような場合には適切な長方形によって描画領域が決定されることがわかる。

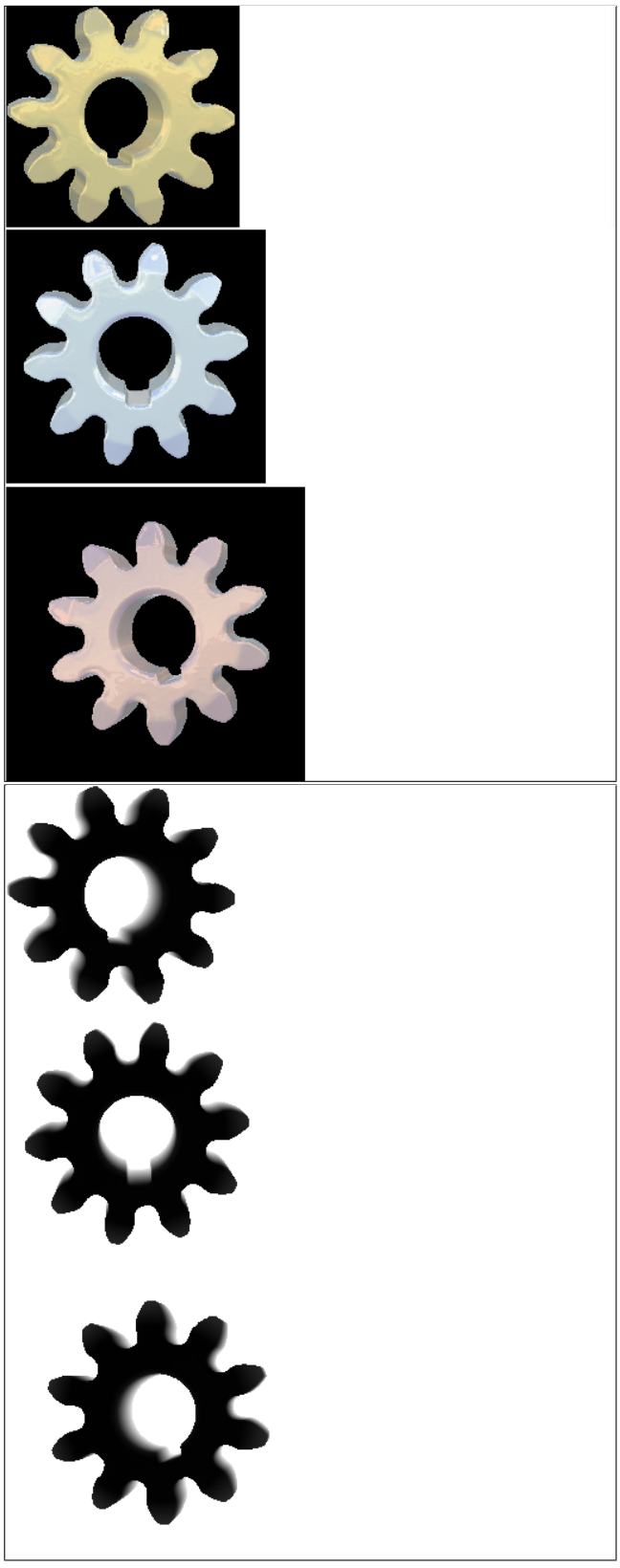

図 5 コンテンツノードが生成する描画結果画像と深度画像

また、図 8 のようにコンテンツの AABB に対して入り組んだ前後関係がある場合にも、深度バッファを合成していることから適切な結果画像が生成されていることが確かめられる。図 7 に示されている深度は今回の実験では 8bit の

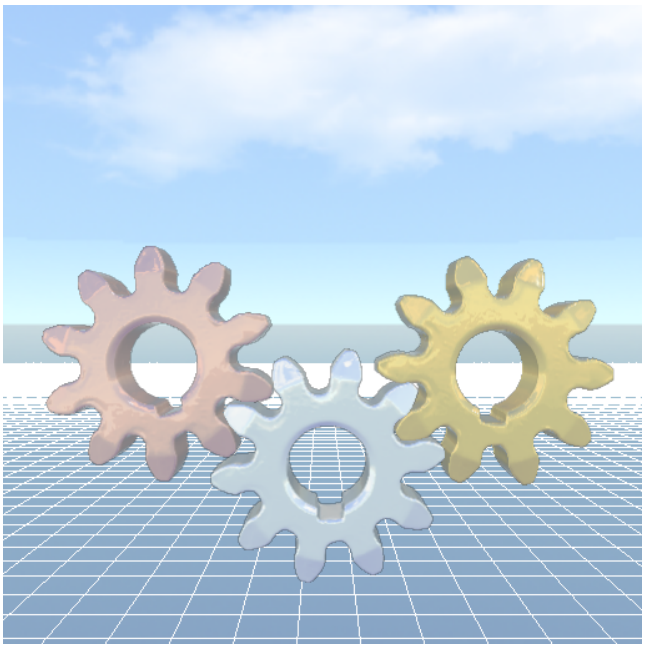

図 6　合成後の映像

ビット長でデータを保持している。これは、深度バッファのテクスチャとしてはかなり短いビット長であると言えるが、今回の実験では十分正確に描画されている。深度をコンテンツ空間からクライアントノードの空間に変換することによって、コンテンツノードの最大厚みが大きくない場合にはビット長の短いテクスチャを深度バッファとして十分に用いることができる。また、モデルに対して正面でない場合や、斜めに見ている状況下であっても適切な視点、射影行列における描画が行われており AABB に対して十分に小さい領域が描写されていることがわかる。

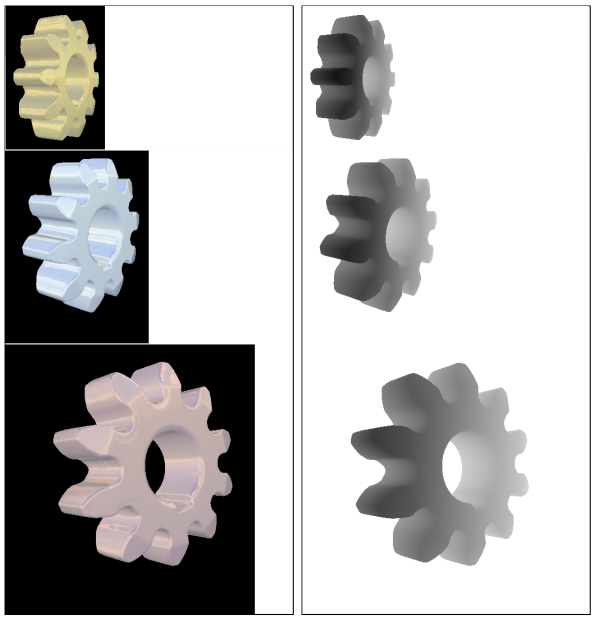

図 7　コンテンツノードが生成する描画結果画像と深度画像

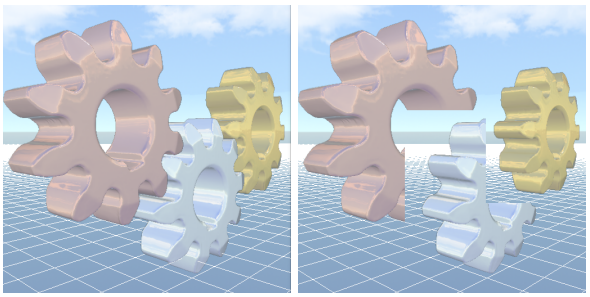

図 8　深度バッファを適切にマージした場合 (左)、深度バッファへの書き込みを行わない場合 (右)

5. 考察

モデル側の処理と、シーン全体での処理で分割できる事に関して計算コストの観点から俯瞰する。図 9 の左側は一般的なゲームなどの 3DCG を扱うアプリケーションの 1 フレームあたりの手順を示している。これらの処理は、通常、各モデル毎に対して実行されている処理と、モデルに関係しない全体の処理が存在する。例えば、一つのキャラクタの服の揺れといった物理演算処理は他のモデルの情報を必要とせず、コンテンツノードで実行可能な処理となる。一方、モデル同士の衝突処理などは、複数個のモデルの情報が必要ではあるが、例えばバウンディングボックスの情報のみをやりとりしてシーン全体の衝突判定などを行うシーン全体の処理となる。このように、モデル単体のデータで完結可能なローカルな処理、複数個のモデルのデータが必要なマクロな処理に分割して考えることによって図 9 の右側のように本提案のスケーラビリティを計算量の観点から議論することができる。

N 個のモデルが存在する空間を描画する事を考える。k 番目のモデル C_k を描画する際にかかる描画コスト (ローカルな処理) を L_k とする。画面上をモデルが占める領域の大きさを s_k としたとき、転送に掛かる処理を送受信問わず、$T \cdot s_k$ であるとする。この時、コンテンツノードの計算量は以下のようになり、s_k はモデルの複雑度によらずカメラからの遠近によってのみ増減するため、主な計算量の大きさは L_k によって定まる。

$$T \cdot s_k + L_k \tag{17}$$

一方、クライアントノードにおける計算量は

$$\sum_{k=1}^{N} T \cdot s_k \tag{18}$$

となり、モデルの描画負荷によらないと分かり、ユーザ数だけ想定すれば必要な計算量を求められる。

6. 結論と今後の課題

本論文では事前に全体での描画負荷が推測できない状況下で、動的なスケーリング可能なパラレルレンダリングの

表 1　提案手法の各ノードの入出力

提案手法		
コンテンツノード	保持しておく情報	担当する 1 モデルあたりのデータ、アニメーション、スクリプト等
	各フレームの入力	ユーザカメラの姿勢、射影行列、ユーザ環境の解像度
	各フレームの演算	担当するモデルのみの姿勢更新、描画、コンテンツ固有の特殊な処理
	各フレームの出力	適切なサイズ、位置、射影変換での描画結果画像及び深度画像
クライアントノード	保持しておく情報	遠景の描画に用いるテクスチャ等
	各フレームの入力	ユーザの操作情報、各コンテンツの直近のフレームの描画結果画像及び深度画像
	各フレームの演算	コンテンツ全体の大まかな位置の更新
	各フレームの出力	最終的な描画結果
シーンノード	保持しておく情報	シーンを構成するコンテンツの姿勢、位置等
	各フレームの入力	各ユーザの自身の位置の更新リクエスト
	各フレームの演算	保持しているコンテンツの位置の更新
	各フレームの出力	リクエストされた位置の近傍のコンテンツのリストとそれぞれの位置
一般的な手法		
描画ノード	保持しておく情報	全てのモデルデータ、アニメーション、スクリプト等
	各フレームの入力	ユーザの操作情報
	各フレームの演算	ゲームの描画に必要な全ての処理
	各フレームの出力	最終的な描画結果

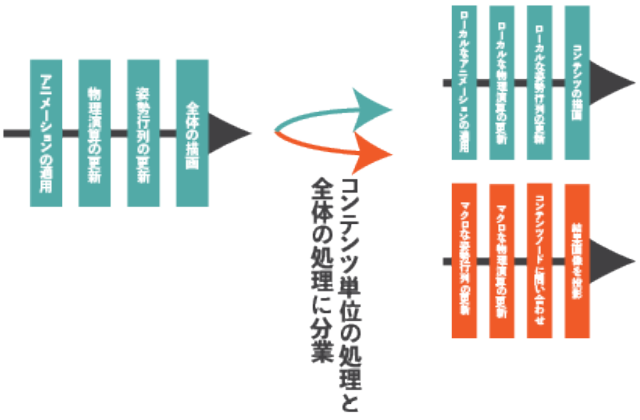

図 9　レンダリング処理の分離

手法をコンテンツ空間単位で分割することにより提案した。各コンテンツノードとして用意されるサーバには担当するコンテンツの情報のみを保持するだけでよく、またクライアントノードでは各モデルを保持する必要がない。これによって、全体のモデルを保持しなければならないような、サーバの上限の性能に如実に制限されるような構成を避けることが可能となる。また、この結果として生成される映像は通常の手法と比べても目に見える差が起きていない。今回の実験では計算機 1 台内での通信のみで分散処理を行い正しい絵が描画できるかを示すことに留まった。今後は実際に複数台の計算機を用いた多ノードにより、モデルの数や種類、描画負荷に対してどの程度の台数効果を発揮できるかを測定する予定である。

参考文献

[1] Pineda, J.: A Parallel Algorithm for Polygon Rasterization, *SIGGRAPH Comput. Graph.*, Vol. 22, No. 4, pp. 17–20 (online), DOI: 10.1145/378456.378457 (1988).
[2] Akeley, K.: Reality Engine graphics, pp. 109–116 (online), DOI: 10.1145/166117.166131 (1993).
[3] Green, S.: *Parallel Processing for Computer Graphics*, MIT Press, Cambridge, MA, USA (1991).
[4] Kobayashi, H., Nakamura, T. and Shigei, Y.: Parallel processing of an object space for image synthesis using ray tracing, *The Visual Computer*, Vol. 3, No. 1, pp. 13–22 (online), DOI: 10.1007/BF02153647 (1987).
[5] Cox, M. and Hanrahan, P.: Pixel Merging for Object-parallel Rendering: A Distributed Snooping Algorithm, *Proceedings of the 1993 Symposium on Parallel Rendering*, PRS '93, New York, NY, USA, ACM, pp. 49–56 (online), DOI: 10.1145/166181.166188 (1993).
[6] Eilemann, S., Steiner, D. and Pajarola, R.: Equalizer 2.0 - Convergence of a Parallel Rendering Framework (2018).
[7] Ahrens, J., Law, C., Schroeder, W., Martin, K., Inc, K. and Papka, M.: A Parallel Approach for Efficiently Visualizing Extremely Large, Time-Varying Datasets (2000).
[8] Samanta, R., Funkhouser, T., Li, K. and Singh, J. P.: Sort-First Parallel Rendering with a Cluster of PCs (2000).
[9] Corrêa, W. T., Klosowski, J. T. and Silva, C. T.: Out-of-core Sort-first Parallel Rendering for Cluster-based Tiled Displays, *Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization*, EGPGV '02, Aire-la-Ville, Switzerland, Switzerland, Eurographics Association, pp. 89–96 (2002).
[10] 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) (2008).
[11] Cook, R. L. and Torrance, K. E.: A Reflectance Model for Computer Graphics, *ACM Trans. Graph.*, Vol. 1, No. 1, pp. 7–24 (online), DOI: 10.1145/357290.357293 (1982).