2011年度　卒　業　論　文

リアルタイムなエネルギー波表現の高速化に関する研究

指導教員：渡辺 大地 講師
三上 浩司 講師

メディア学部 ゲームサイエンスプロジェクト
学籍番号　M0108238
下村 匠

2011 年度　　卒　業　論　文　概　要

論文題目

リアルタイムなエネルギー波表現の高速化に関する研究

メディア学部 **学籍番号：M0108238**	**氏名**	下村 匠	**指導教員**	渡辺 大地 講師 三上 浩司 講師

キーワード	3DCG、ボリュームレンダリング、リアルタイム、 CG エフェクト、プログラマブルシェーダ

近年、リアルタイム 3DCG の分野ではボリューメトリックエフェクトの研究が盛んである。従来のポリゴンとテクスチャーを使用した手法では、煙や炎といった物体表面の存在しないエフェクトを表現することが難しく、それを代替する様々な手法が考案されている。2010 年に阿部らが発表した「エネルギー波表現のリアルタイムレンダリング」では、創作表現に見られるビーム状のエフェクトであるエネルギー波を、GPGPU を用いる事で高速にレンダリングしている。本研究では阿部らの手法を基に、これの高速化を行った。

阿部らの手法では、スクリーンピクセル毎にその視線上のエネルギー総量を求める事でエネルギー波のレンダリングを行っている。エネルギー総量の計算には初等関数に展開された線積分を用いていたが、この計算自体の重さと、計算のための GPG と CPU 間のデータ転送によるオーバーヘッドの発生が問題であった。本手法ではこのエネルギー総量の計算を事前計算データの参照へ置き換える事で計算量を減らしている。視線とエネルギー波の位置関係に対応した視線上のエネルギー総量をあらかじめ計算しておき、このデータ配列をルックアップテーブルとして利用する。描画時には計算コストの高い積分計算を行う必要が無いため、計算量を減らす事ができる。これは実行時間比較で 6 倍の高速化となった。同時にこの計算をプログラマブルシェーダー内で処理を完結させる事で不要なオーバーヘッドを解消した。これにより描画するエネルギー波が少数の場合に大幅な速度の改善ができた。

データを事前に持っておく手法上、ある程度の制約や表現誤差が発生したがこれは許容範囲に収まっている。一方で最大約 66 倍という大幅な高速化が達成でき、本手法がリアルタイムレンダリングにおいて有効であることを示した。

目 次

図 目 次

第 1 章

はじめに

1.1 背景

昨今、漫画やアニメ、映画のような創作コンテンツ上では様々な非現実的特殊効果が描かれる [1]。強く発光するエネルギー体はその中でも代表的な表現の 1 つで、アクションや格闘シーンでの特殊効果として利用例が多い。代表的な例としては、漫画「ドラゴンボール」[2] の作中における気功波、テレビアニメーション「機動戦士ガンダム」[3] の作中におけるビーム攻撃などが挙げられる。本研究ではこの特殊効果をエネルギー波と名付け、「空間中のエネルギーの密度が高い場所が強く発光する」「ある地点から広がり、また直線状に伸びる」現象と定義する。

特徴的なのは前者の点、エネルギーの分布する空間全体が発光する事である。エネルギー波のような空間全体が発光する現象は、現在のリアルタイムな 3 次元コンピュータグラフィックス (以下 3DCG) で用いるポリゴンによる描画では制限がある。ポリゴンによる描画は物体の中身という概念を持たず、物体の表面しか表現できないからである。

図 1.1 は半透明なエフェクトを描画する手法として最も一般的な手法、ビルボードテクスチャである。ビルボードテクスチャは常にカメラに向けたポリゴンの板にテクスチャーを貼り付けるもので、簡単かつ高速なためリアルタイム 3DCG の分野で利用例が多い。平面を感じないように遮蔽物との交差点を滑らかにするな

どといった改良も存在する [4] が、その実体は板であるため厚みを表現する事はできない。

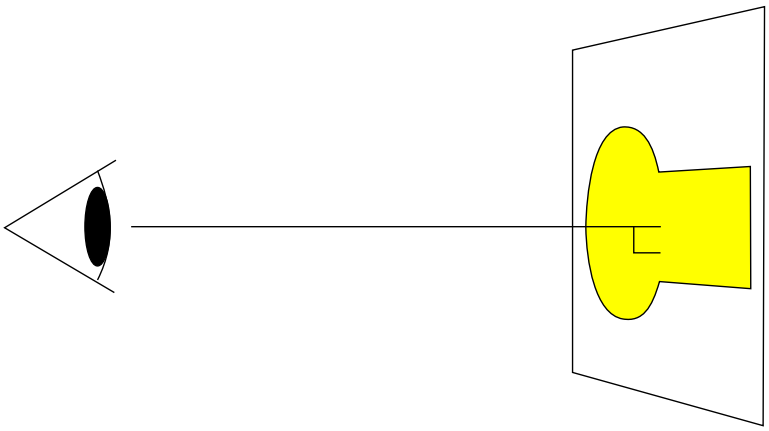

図 1.1: ビルボードテクスチャ

図 1.2 は物体の厚みを利用したボリュームライト [5] という手法である。メッシュオブジェクトの裏面と表面の深度差から厚みを計算し、それを輝度に利用する。図の A が表面の深度、B が裏面の深度で、B から A を引く事で部分的な厚みを得る事ができる。しかしこの手法ではオブジェクト内の輝度分布をおおよそ一様にしか設定する事ができず、そのままではエネルギー波の表現に利用できない。

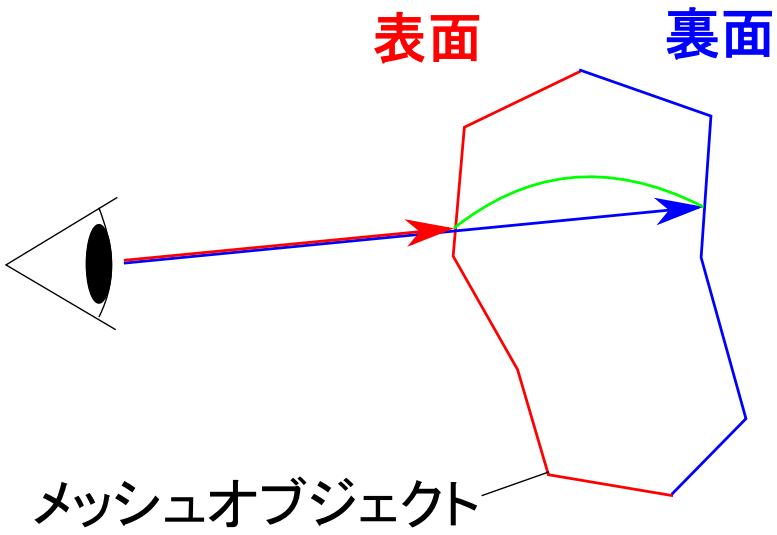

図 1.2: ボリュームライト

3 次元空間中の分布データを可視化する技術に、Levoy[6]、Drebin ら [7]、Saito[8] などの研究、ボリュームレンダリングがある。

この手法は CT スキャンや MRI によって測定した人体の内部構造の描画や、Zhao ら [9] の炎や Umenhoffer ら [10] の煙のような物体の表面が存在しない現象のシミュレーション、描画に用いられる。ボリュームレンダリングでは空間中の分布データであるボリュームデータを用いて描画を行う。ボリュームデータとは 3 次元空間を一定間隔で区切り、分布をサンプリングしたものである。このボリュームデータをレンダリングする手法にはレイキャスティング法 [11]、Marching cubes 法 [12]、テクスチャベース法 [13] などの種類がある。図 1.3 はレイキャスティング法を用いた場合の模式図である。レイキャスティング法では視線上を一定間隔に区切り、各ポイントから周囲のボリュームデータをサンプリングする。

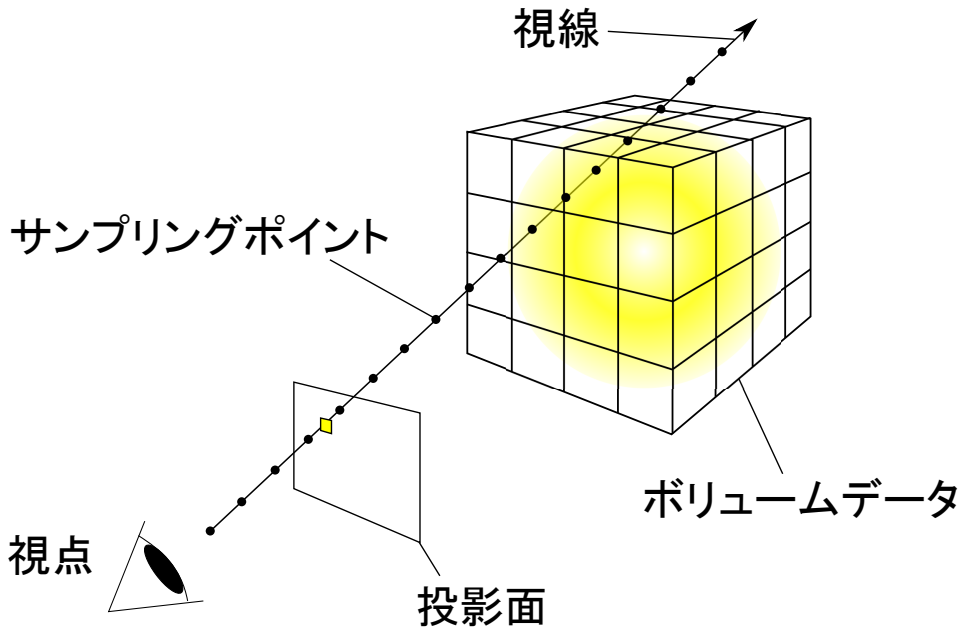

図 1.3: ボリュームレンダリング

従来のボリュームレンダリングは計算量が多くリアルタイムレンダリングには向かない手法であり、リアルタイムレンダリングには専用ハードウェアを用いる事もあった [14]。近年では GPU の著しい発達もあり、Stuart ら [15]、Viola ら [16]、Kruger ら [17]、Hadwiger ら [18] などが GPU を用いた高速なレンダリング手法を発表している。

ボリュームレンダリングにおける画質はボリュームデータの解像度に依存するが、３次元データであるボリュームデータは解像度に合わせてデータ量も爆発的に増加する。これはメモリを圧迫するだけでなく、計算コストの増加にもつながる。また、形状が変化する際にはボリュームデータの再生成を行う必要があり、これをリアルタイムに行うのは難しい。

2010 年に阿部ら [19] が発表した「エネルギー波表現のリアルタイムレンダリング」では、エネルギー波の表現に前節のボリュームレンダリングに近い手法を採用しつつその弱点を解消している。

まずエネルギー波の形状生成だが、ボクセルデータやポリゴンモデルを使わず

に数学的手法を用いている。エネルギー密度の分布を関数の形で定義し、それを基に各スクリーンピクセルの視線上のエネルギー総量を線積分によって求める。そしてエネルギー総量を輝度としてスクリーンに加算合成する事でエネルギー波のレンダリングを行う。エネルギー総量を求める線積分処理は初等関数の組み合わせによって行う事ができる。

図 1.4 と図 1.5 は先行研究における阿部らの手法とボリュームレンダリングとの画質の比較である。阿部らの手法はボリュームレンダリングと同様の形状であり、エネルギー波が上手く表現できている。

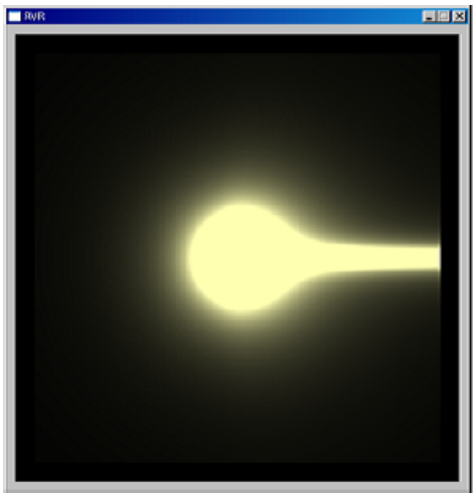

図 1.4: ボリュームレンダリング

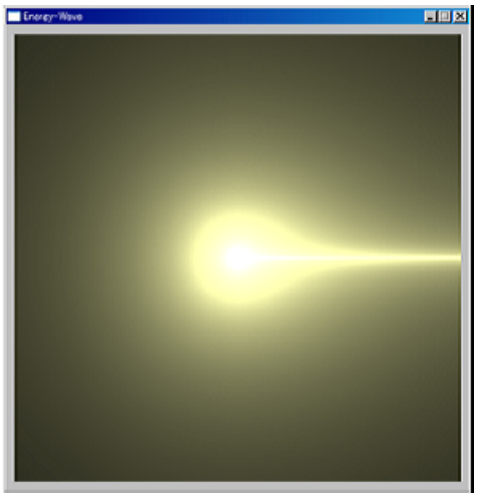

図 1.5: 阿部らの手法

スクリーンピクセル毎に積分計算を行うため計算量は多いが、阿部らの手法では GPU による汎用計算 (General-purpose computing on graphics processing units、以下 GPGPU) 技術の 1 つである CUDA[20] を用いてリアルタイム性を保っている。

本研究ではこの阿部らの手法を基とし、これの高速化を行った。まず、スクリーンピクセル毎に行っていた計算量の多い積分計算を取り除くため、事前計算データからこれを読み出す手法を取った。一定の条件下での視線上のエネルギー総量をあらかじめ計算し、この配列を保存しておく。描画時にはエネルギー波と視線の位置関係からこのデータを読み出すだけとなり、計算量を大幅に削減する事がで

きた。同時に阿部らの手法では計算不可能なエネルギー密度分布を指定する事が可能になり、表現の幅が一層増した。また阿部らの手法で発生していたオーバーヘッドを解消する事で、エネルギー波の数が少ない状況下でのパフォーマンスが劇的に改善した。検証の結果では阿部らの手法に比べ最大約 66 倍の高速化となり、計算部分だけでも約 6 倍高速な結果を示した。一方でエネルギー波の表現力はほぼ劣らず、一部の状況で発生する画質の劣化も多少のコストを支払うことで緩和することが可能である。同等の表現のより高速なレンダリングが達成できたことをもって、本手法がリアルタイムレンダリングにおいて有効であることを示した。

1.2 論文構成

2 章では本研究で提案する手法について述べる。3 章では本研究の実行結果を用いて、先行研究との比較、評価を行う。

1.3 数式の定義

本論文で用いる数式を以下で定義する。

- $\mathbf{v} \cdot \mathbf{w}$ はベクトル $\mathbf{v}$、$\mathbf{w}$ の内積を表す。
- $\mathbf{v} \times \mathbf{w}$ はベクトル $\mathbf{v}$、$\mathbf{w}$ の外積を表す。
- $|\mathbf{v}|$ はベクトル $\mathbf{v}$ のノルムを表す。

第 2 章

提案手法

本章では本研究のエネルギー波の描画手法について述べる。

まずはじめにエネルギー波の形状を定義する。本手法では阿部らの手法を踏襲し、エネルギー波の形状を数式によって表わす。エネルギー波は球状のエネルギー体と円柱状のエネルギー体を組み合わせて生成し、このエネルギー体内のエネルギー密度分布を決めるエネルギー密度分布関数を定義する。ここまでは阿部らの手法と同様である。

次にこのエネルギー密度分布関数を用いて事前計算データとなるルックアップテーブルを生成する。ルックアップテーブルはあらかじめ生成しておき、毎フレームの描画では再生成を行わない。ルックアップテーブルとは、複雑な計算処理を単純な配列の参照処理で置き換えて効率化を図るためのデータ構造である。

最後の描画でも阿部らの手法と同様に視線上のエネルギー総量を用いてピクセルの輝度を計算するが、このエネルギー総量はその場で計算するのではなくルックアップテーブルから取得する。エネルギー総量を求めるための積分計算を省く事で計算量を減らす。

計算はプログラマブルシェーダのみで行う。プログラマブルシェーダは GPU が描画の際に行うシェーディングを自由にプログラミングできるようにしたものである。プログラマブルシェーダの中でもピクセルシェーダはスクリーンピクセル単位で計算を行う部分であり、ピクセル毎の計算を並列に行う。図 2.1 は阿部らの

手法と本手法の描画におけるデータフローの模式図である。

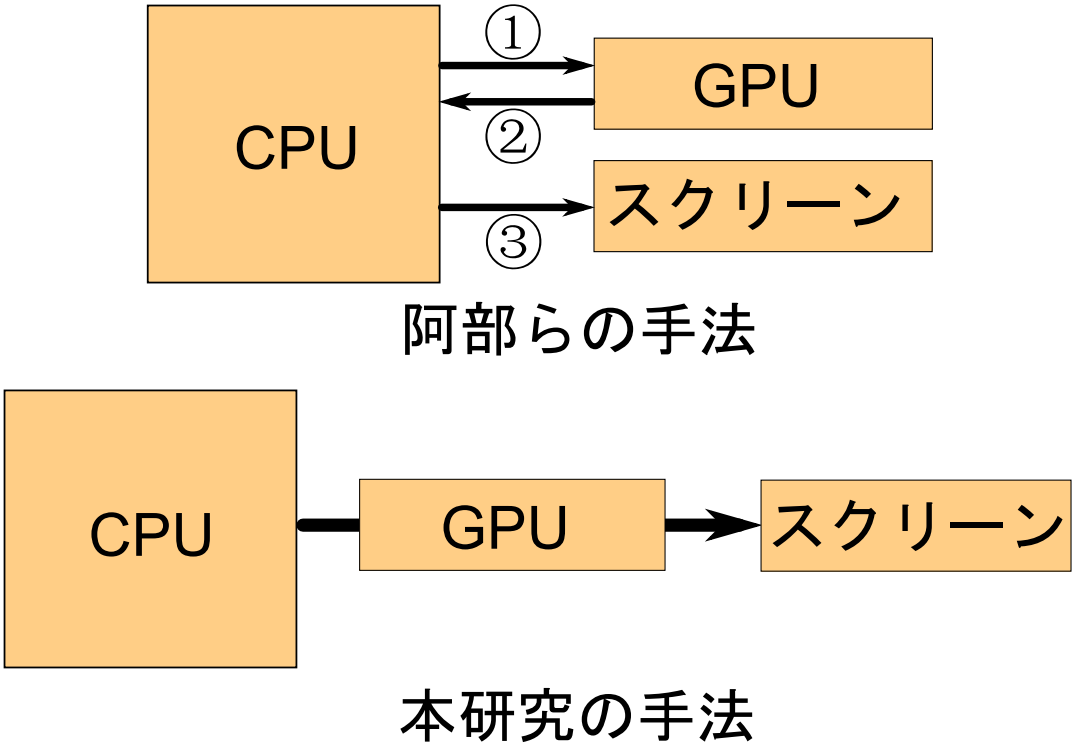

図 2.1: 描画におけるデータフロー

阿部らの手法では、GPU と CPU の間でデータを往復する必要があり、これが過大なオーバーヘッドとなっている。本手法では描画計算がピクセルシェーダ内で完結するため、阿部らの手法にみられるようなオーバーヘッドがなく高速である。

ルックアップテーブルは 2 次元テクスチャーとして保存し、ピクセルシェーダで利用する。テクスチャーのデータ構造はただの配列と同じだが、後述する中間値の線形補間やクリッピングといった機能を利用する事ができるため、ルックアップテーブルとして利用するのに適している。

2.1 エネルギー波形状の生成

エネルギー波形状の生成は、阿部らの手法を踏襲し、プリミティブ形状の組み合わせを用いる。図 2.2 はプリミティブ形状の組み合わせによる、典型的なエネル

ギー波の形状生成の模式図である。

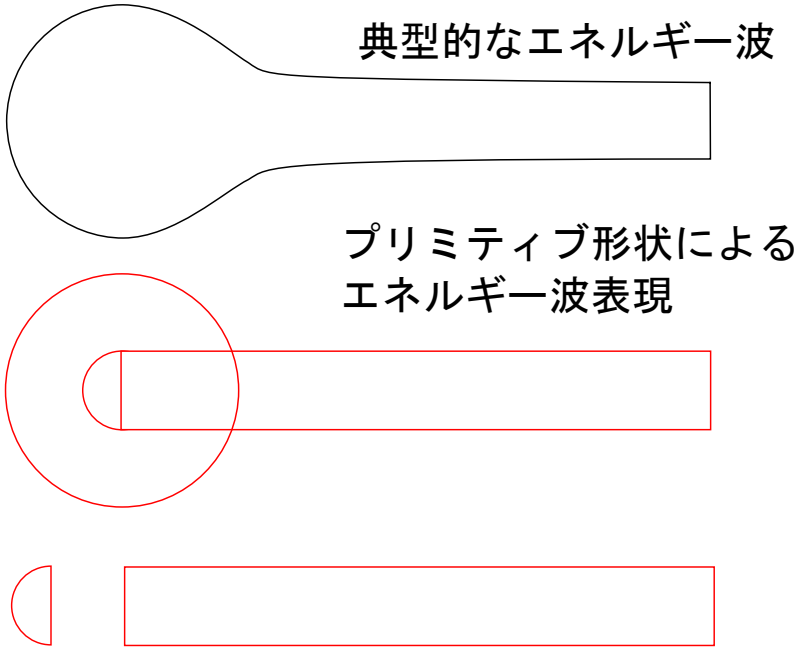

図 2.2: エネルギー波形状の生成

球状のプリミティブ形状と半カプセル状のプリミティブ形状を組み合わせる事で、エネルギー波の形状を再現する。さらに半カプセル状のプリミティブ形状は、エネルギー波の方向に垂直な面によって球状のプリミティブ形状と円柱状のプリミティブ形状を切断し組み合わせる事で生成する

これら球状・円柱状のプリミティブ形状も阿部らの手法と同様、数式によって表わす。球状のプリミティブでは中心点、円柱状のプリミティブでは中心線を用いる。そして任意の地点のエネルギー密度を、このエネルギー中心からの距離 x によって表すエネルギー密度分布関数 $f(x)$ を作成する。ここでは $f(x)$ を式 (2.1) で表す。

$$f(x)\begin{cases} 99 & (x \leq 0.01) \\ \left(\dfrac{1}{x} - 1\right) & (0.01 < x < 1) \\ 0 & (1 \leq x) \end{cases} \tag{2.1}$$

$f(x)$ 内の計算には特に制限は無いが、x の定義域は 0 以上である必要がある。これはエネルギー総量を事前に計算してルックアップテーブルへ保存するためである。また x が 1 以上の場合の $f(x)$ は 0 となる必要がある。

ルックアップテーブルは大きさが有限であり、無限遠までエネルギー密度が変化するエネルギー体は描画できない。エネルギーが存在する空間はエネルギー中心より一定以内と制限する必要があるため、この限界距離を 1 と置いてエネルギー密度関数を作成する必要がある。

このエネルギー密度分布関数を用いて、球形の場合はある点を中心に、円柱の場合はある直線を中心にエネルギーを分布する事でプリミティブ形状を生成する。

2.2 ルックアップテーブルの生成

ルックアップテーブルには、エネルギー中心付近を通る視線上のエネルギー総量を記録する。エネルギー密度はエネルギー中心からの距離によってのみ変化するため、エネルギー体と視線がどのような位置関係であっても、視線上のエネルギー総量は最近距離との対応を取る。この最近距離からエネルギー総量を取得できるルックアップテーブルを生成しておく事で、描画時の計算コストを低減する。ルックアップテーブルは球状のエネルギー体を想定して生成する。円柱状のエネルギー体の描画では参照の仕方を工夫する事によって、球状のエネルギー体に使う物と同じルックアップテーブルを使用できる。

エネルギー中心を A、計算対象とする直線を L とした時の、L 上のエネルギー総量を算出する方法について述べる。L 上に存在する A との最近点を P、AP 間の距離を d、L 上で P から 1 離れた点をそれぞれ Q、R と置く。さらに Q、R の間に任意の点 E を置き、QE 間の距離を s とする。エネルギーの分布範囲は中心から 1 の距離までであるため、d が取りうる値は 0 から 1 の範囲である。また Q、R 間の距離は 2 であるため、s が取りうる値は 0 から 2 の範囲である。図 2.3 はこの模式図である。

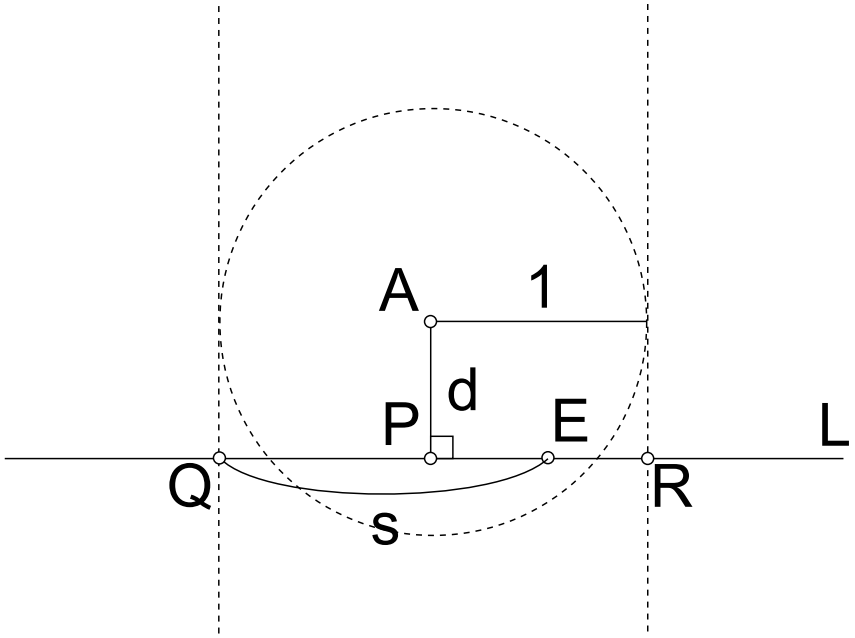

図 2.3: エネルギー総量を計算する区間

ルックアップテーブルには AE 間のエネルギー総量を格納し、d、s を入力とした関数 $g(d, s)$ と見立てる。d、s より $g(d, s)$ は式 (2.2) のように表せる。

$$g(d, s) = \int_0^s f\left(\sqrt{d^2 + (t-1)^2}\right) dt \tag{2.2}$$

エネルギー総量の計算はどのような手法を用いてもよいが、この積分量を解析的に求めることは難しいため、数値積分を用いて近似値を求める。

数値積分に用いる分割数を o として、$g(d, s)$ は式 (2.3) のように表せる。

$$g(d, s) = \sum_{k=0}^{o} f\left(\sqrt{d^2 + \left(\frac{ks}{o} - 1\right)^2}\right) \tag{2.3}$$

数値積分を用いてエネルギー総量を計算する場合は、誤差を減らすために十分な分割数を指定する必要がある。分割数の増加は計算速度の低下につながるが、本研究ではこれは描画時には計算しないため、その計算時間については考慮しない。

ルックアップテーブルはこの d、s を入力とした関数 $g(d,s)$ と見立てる。d、s のそれぞれが取りうる範囲は限定しているので、これをそれぞれ任意の分割数 n、m で区切り、各組み合わせの計算結果をテクスチャーへ記録していく。テクスチャーは $n \times m$ サイズで、浮動小数点フォーマットを使用する。エネルギー総量の値が取りうる幅は大きいため、整数よりも浮動小数点が好ましい。

データを取り出す際に d、s はルックアップテーブルの範囲を範囲をはみ出してしまう場合があるが、シェーダの機能を利用してクリッピングを行う。0 より小さな値は 0 に、最大値より大きな値は最大値にクリップする。記録した配列データの中間値も同様にシェーダの機能を利用して線形補間した値を求める。

2.3 描画

この節では球状、円柱状それぞれのエネルギー体を描画について述べる。

描画の流れは阿部らの手法と同じく、スクリーンピクセル毎に視線上のエネルギー総量を求め、このエネルギー総量とエネルギー波の色を乗算したものをピクセルの輝度とする。これを最後にスクリーンへ加算合成する事でエネルギー波を描画する。球状のエネルギー体と円柱状のエネルギー体の合成は、それぞれ別に描画を行うだけである。

以下、2.3.1 項では球状のエネルギー体の描画について、以下、2.3.2 項では球状のエネルギー体の描画について述べる。

2.3.1 球状のエネルギー体の描画

球状のエネルギー体の描画では、まずピクセルの視線とエネルギー中心との最近距離 d を求める。視線方向の単位ベクトルを $\mathbf{v}$、エネルギー中心を A、カメラ位置を B と置いて、d を式 (2.4) で求める事ができる。

$$d = |\mathbf{v} \times (\mathbf{A} - \mathbf{B})| \tag{2.4}$$

この d を用いてルックアップテーブルから視線上のエネルギー総量を取りだす。$g(d,2)$ の値がエネルギー総量となる。任意の半径 r のエネルギー体を描画する場合は、式 (2.5) で求められる d' を用いる。

$$d' = \frac{d}{r} \quad \text{ただし } (0 < r) \tag{2.5}$$

r が 0 の場合には d' を計算ができないが、大きさ 0 のエネルギー体はそもそも描画する必要が無いので除外する。

d' だけでは、エネルギーの分布範囲内にカメラや遮蔽物が入った場合に、その間だけのエネルギー総量を求める事ができない。これを求めるためには、カメラへの距離 y、遮蔽物への距離 u を求める必要がある。y、u はそれぞれカメラと遮蔽物への距離であるが、これは視線上のエネルギー中心への最近点 P からカメラ側へ r 引いた地点 Q を中心とする。

y、u は式 (2.6) と式 (2.7) によって計算できる。

$$y = \frac{1}{r}\left(r - \frac{(\mathbf{A}-\mathbf{B})\cdot\mathbf{v}}{|(\mathbf{A}-\mathbf{B})\cdot\mathbf{v}|}\sqrt{|\mathbf{A}-\mathbf{B}|^2 - d^2}\right) \tag{2.6}$$

$$u = \frac{1}{r}\left(r - \frac{(\mathbf{A}-\mathbf{B})\cdot\mathbf{v}}{|(\mathbf{A}-\mathbf{B})\cdot\mathbf{v}|}\sqrt{|\mathbf{A}-\mathbf{B}|^2 - d^2} + z\right) \tag{2.7}$$

z はカメラから遮蔽物までの距離である。これは遮蔽物を描画する際にあらかじめ保存しておく。

図 2.5 は球状のエネルギー体の描画する模式図である。

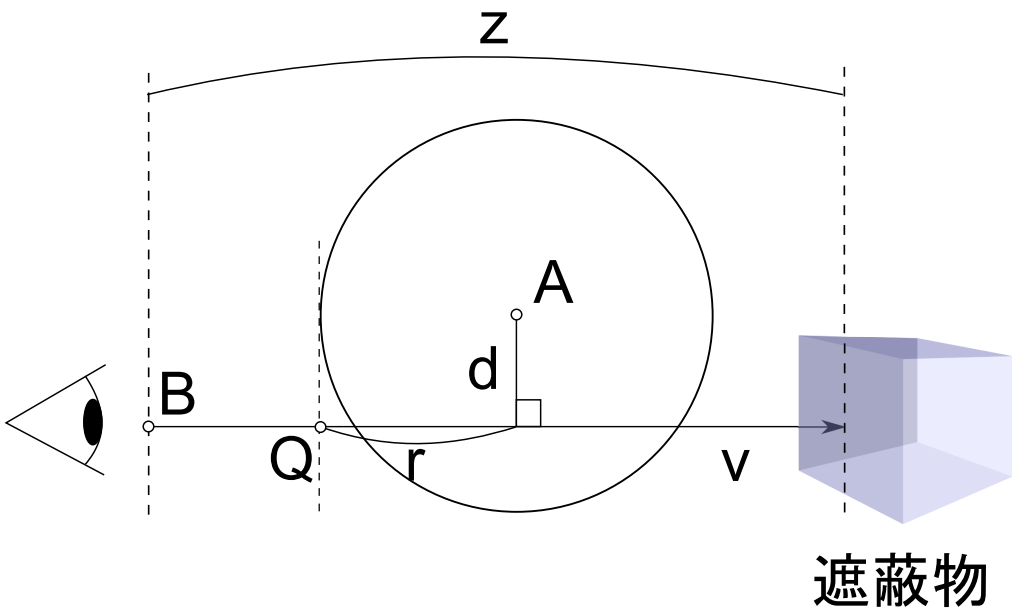

図 2.4: 円柱に斜めに入射した視線

カメラから遮蔽物の区間のエネルギー総量は $g(d', u)$ から $g(d', y)$ を引く事で求められる。

2.3.2 円柱状のエネルギー体の描画

円柱状のエネルギー体を描画する際では、d は円柱の中心線と視線との最近距離を表す。円柱の中心線上の任意の点を A、円柱方向の単位ベクトル $\mathbf{w}$ として、d は式 (2.8) で求める事ができる。

$$d = |(\mathbf{v} \times \mathbf{w}) \cdot (\mathbf{A} - \mathbf{B})| \tag{2.8}$$

遮蔽物の計算は球状のエネルギー体を描画する際と同様に式 (2.6) と式 (2.7) を利用する。

また円柱状のエネルギー体では、視線が斜めに入射する分だけ視線が円柱内を通る区間が長くなる。図 2.5 は円柱と視線の関係図である。

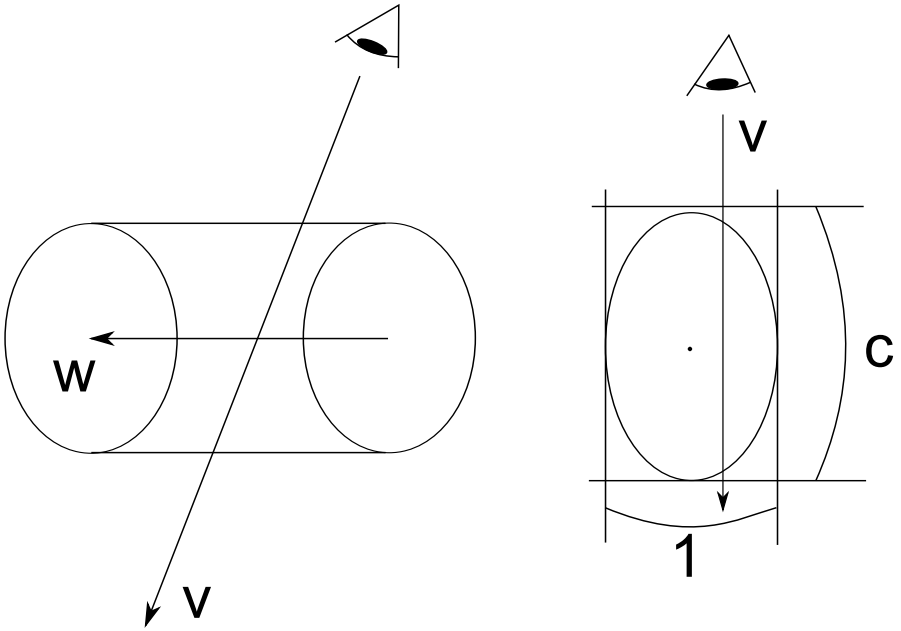

図 2.5: 円柱に斜めに入射した視線

この変化割合 c は式 (2.9) で表すことができる

$$c = \frac{1}{\mathbf{v} \cdot \mathbf{w}} \tag{2.9}$$

この c を、式 (2.6) と式 (2.7) で求めた y、u と、最終的なエネルギー総量に乗算する。

その後の描画手順は球状のエネルギー体の場合と同様である。

第 3 章

評価

本章では本研究で提案したエネルギー波のレンダリング手法に沿って実装したプログラムの実行結果を用い、その評価を行う。本研究で試作したプログラムではMicrosoft 社の Windows OS 向けグラフィックス API である DirectX[21] と、DirectX 上で使われるプログラマブルシェーダ言語 HLSL を用いている。

3.1 実行結果

先行研究と同等のエネルギー分布関数を指定した場合の実行結果が図 3.1 である。ルックアップテーブルの解像度は 32×32 ピクセルで、先行研究と近似した結果となっている。図 3.2 は不透明なオブジェクトを配置した例で、適切に陰面処理が機能しているのが見てとれる。

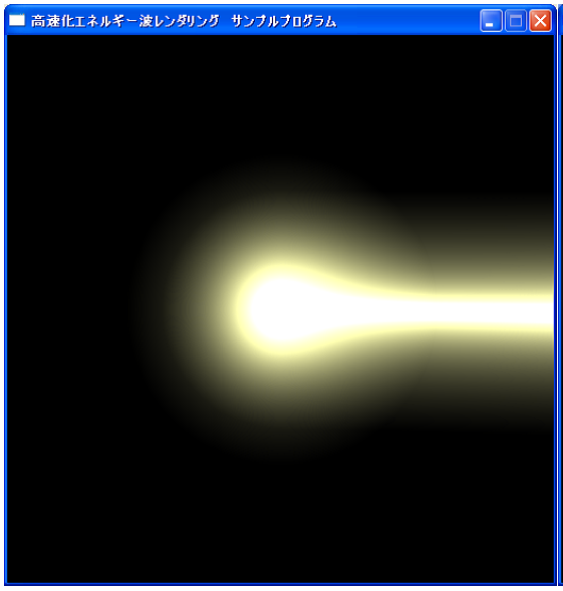

図 3.1: エネルギー波

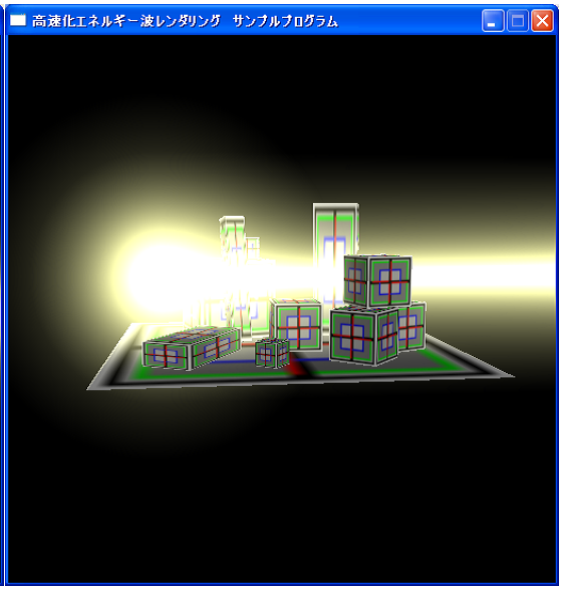

図 3.2: 陰面処理

阿部らの手法では、初等関数の組み合わせのみで線積分計算ができるようなエネルギー密度分布関数しか用いる事ができない制約があった。しかし本手法ではエネルギー総量の計算に数値積分を用いたため、このような制限は存在しない。図 3.3 は阿部らの手法では提案になかった、薄膜状のエネルギー体である。

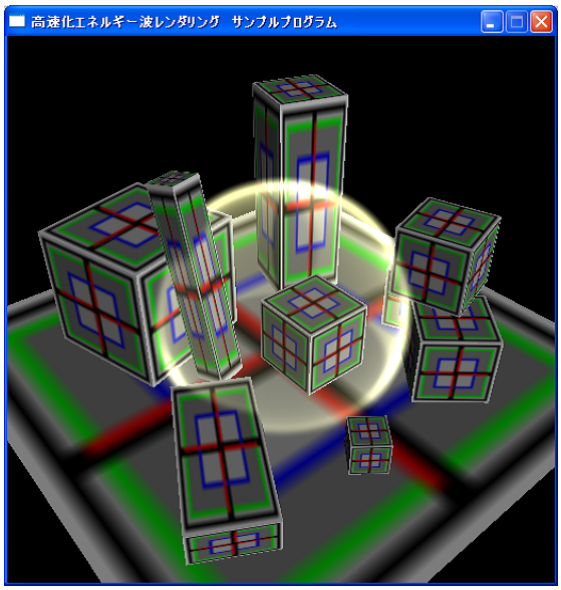

図 3.3: 薄膜状の球状エネルギー

式 (3.1) はこのエネルギー体を定義するエネルギー密度分布関数である。中心部分にはエネルギーが存在せず、エネルギー中心から 0.8 程度離れた空間にのみエネルギーが存在する。

$$f(x)\begin{cases} 5-|100(x-0.8)| & (0.75<x<0.85) \\ 0 & (x\leq 0.75、0.85\leq x) \end{cases} \tag{3.1}$$

3.2　先行研究との速度比較

阿部らのプログラムと本研究のプログラムの速度検証は表のとおりである。

表 3.1: 検証に使用した環境

CPU	Intel®Core(TM)2 Duo 3.16GHz
RAM	4GB
GPU	nVIDIA GeForce 9800GTX

表 3.2: 速度の比較

セット数	先行研究	本研究
1	48FPS	3200FPS
5	30FPS	880FPS
10	25FPS	460FPS

球および半カプセル形状を組み合わせたエネルギー波一つを 1 セットとする。どちらも解像度 512 × 512 ピクセルの解像度で、ルックアップテーブルの解像度は 32 × 32 ピクセルである。この設定では阿部らの手法に比べ最大 66 倍の高速化が達成できた。

本手法ではエネルギー形状 1 つに対し 1 回づつ描画を行うため、エネルギー波の個数にほぼ比例してフレームレートが低下している。対して阿部らの手法でフレームレートの低下がセット数に比例していないのは、GPU と CPU 間のデータ転送がネックになっているためである。セット数 1 の結果と 10 の結果からエネルギー波の純粋な描画時間を逆算すると、阿部らのプログラムでは 1 セットあたり約 0.002 秒の処理時間がかかっていた。対する本研究では 1 セットあたり約 0.0003 秒の処理時間となっており、単純な計算部分での処理速度を比較しても 6 倍程高速な結果となった。

3.3 問題点

本手法の問題点として、無限遠に変化の続くエネルギー密度分布を指定できない点があげられる。本手法で用いるルックアップテーブルは有限データであるため、エネルギー中心から一定以上の距離のエネルギー密度は常に 0 としなければならない。しかし、無限遠に変化の続くエネルギー密度分布を指定する必要がある場面は多くなく、使用上の問題とはなりえないと考えられる。

その他、ルックアップテーブルの解像度やエネルギー総量の計算精度に由来する表現誤差も問題と考えられる。図 3.4 は視線と円柱方向のベクトルが平行に近い

場合に発生する、意図しない筋である。ルックアップテーブルの解像度が低いほど、またエネルギー密度の変化勾配が強いほど筋が発生しやすい。これはルックアップテーブルの解像度を上げる事である程度解消できる。

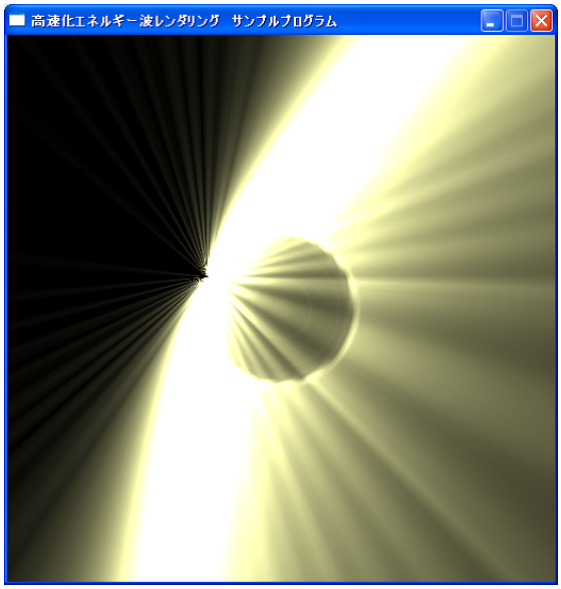

図 3.4: 不自然な筋

第 4 章

まとめ

本研究では先行する阿部らの研究を基にこれの高速化を行った。阿部らの手法では、エネルギー波の輝度を求めるために、スクリーンピクセル毎にその視線上のエネルギー総量を求めて計算していた。この計算には初等関数に展開した線積分が使われていたが、この計算量の多さが一つの問題であった。また GPGPU を用いてこれを計算する際にデータ転送による過大なオーバーヘッドが発生し、処理速度を激しく損なう一因となっていた。

本研究で提案する手法では、ピクセル毎の描画計算から積分計算を排除し計算量を削減した。積分計算による視線上のエネルギー総量の計算は事前に済ませておき、描画時にはその結果を記録したルックアップテーブルからデータを取得するという手法を取った。さらに描画は全てプログラマブルシェーダ内で完結し、阿部らの手法で発生していたオーバーヘッドを解消した。これを合わせて、阿部らの手法に比べて大幅な高速化が達成できた。対して画質の劣化は最小限に留まり、一部の状況で発生する意図しない表現も多少のコストを支払うことで緩和することが可能である。これをもって、本手法がリアルタイムレンダリングにおいて有効であることを示した。

また高速化や応用にはまだ余地が残っている。本研究ではルックアップテーブルは完全に事前計算としていたが、これも GPU を用いて描画すれば毎フレーム生成できるほど高速になる。もしくはルックアップテーブルとなるテクスチャーを

複数毎用意し、これをフレーム毎に切り替えれば計算コストの増加無しにエネルギー密度分布を変化することも可能である。

謝辞

先行研究の執筆者の阿部先輩、指導してくださった渡辺先生、三上先生、他院生の方々、そして愉快な研究室のメンバーに感謝。

参考文献

[1] D. Nowrouzezahrai, J. Johnson, A. Selle, D. Lacewell, M. Kaschalk, and W. Jarosz. A programmable system for artistic volumetric lighting. *ACM Transactions on Graphics (TOG)*, Vol. 30, No. 4, p. 29, 2011.

[2] 鳥山明. 「ドラゴンボール」. 集英社, 1985.

[3] 「機動戦士ガンダム」. 日本サンライズ, 1979.

[4] Tamás Umenhoffer, László Szirmay-Kalos, and Gábor Szijártó. Spherical billboards and their application to rendering explosions. In *Proceedings of Graphics Interface 2006*, GI '06, pp. 57–63, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.

[5] ボリュームライト. *http://maverickproj.web.fc2.com/pg44.html*.

[6] M. Levoy. Display of surfaces from volume data. *Computer Graphics and Applications, IEEE*, Vol. 8, No. 3, pp. 29–37, 1988.

[7] R.A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. In *ACM Siggraph Computer Graphics*, Vol. 22, pp. 65–74. ACM, 1988.

[8] Takafumi Saito. Real-time previewing for volume visualization. In *Proceedings of the 1994 symposium on Volume visualization*, VVS '94, pp. 99–106, New York, NY, USA, 1994. ACM.

[9] Ye Zhao, Xiaoming Wei, Zhe Fan, Arie Kaufman, and Hong Qin. Voxels on fire. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, VIS '03, pp. 36–, Washington, DC, USA, 2003. IEEE Computer Society.

[10] Tamás Umenhoffer, László Szirmay-Kalos, and Gábor Szijártó. Spherical billboards and their application to rendering explosions. In *Proceedings of Graphics Interface 2006*, GI '06, pp. 57–63, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.

[11] Hanspeter Pfister, Jan Hardenbergh, Jim Knittel, Hugh Lauer, and Larry Seiler. The volumepro real-time ray-casting system. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pp. 251–260, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[12] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, Vol. 21, pp. 163–169, August 1987.

[13] Han Xiao and De-Gui Xiao. An accelerating splatting algorithm based on multi-texture mapping for volume rendering. In *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, GRAPHITE '06, pp. 205–208, New York, NY, USA, 2006. ACM.

[14] T. Günther, C. Poliwoda, C. Reinhart, J. Hesser, R. Männer, H.P. Meinzer, and H.J. Baur. Virim: A massively parallel processor for real-time volume visualization in medicine. *Computers & graphics*, Vol. 19, No. 5, pp. 705–710, 1995.

[15] Jeff A. Stuart, Cheng-Kai Chen, Kwan-Liu Ma, and John D. Owens. Multi-gpu volume rendering using mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pp. 841–848, New York, NY, USA, 2010. ACM.

[16] Ivan Viola, Armin Kanitsar, and Meister Eduard Gröller. Gpu-based frequency domain volume rendering. In *Proceedings of the 20th spring conference on Computer graphics*, SCCG '04, pp. 55–64, New York, NY, USA, 2004. ACM.

[17] J. Kruger and R. Westermann. Acceleration techniques for gpu-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, VIS '03, pp. 38–, Washington, DC, USA, 2003. IEEE Computer Society.

[18] Markus Hadwiger, Patric Ljung, Christof Rezk Salama, and Timo Ropinski. Advanced illumination techniques for gpu volume raycasting. In *ACM SIGGRAPH ASIA 2008 courses*, SIGGRAPH Asia '08, pp. 1:1–1:166, New York, NY, USA, 2008. ACM.

[19] 阿部雅樹, 渡辺大地. エネルギー波表現のリアルタイムレンダリング. 芸術科学会論文誌, Vol. 9, No. 3, pp. 93–101, 2010.

[20] Cuda. *"http://www.nvidia.co.jp/object/cuda_home_jp.html"*.

[21] Microsoft. Microsoft directx. *http://www.microsoft.com/japan/directx/default.mspx*.