

# ブラックホール4次元時空における OpenGL/GLSL フレームワークの実装

山下 義行<sup>1,a)</sup>

**概要：**一般相対論の予言するブラックホールは光線の進路を湾曲させる。それを数値計算し、ブラックホール近傍の物体を3次元コンピュータグラフィックス（3D CG）として描画する研究が盛んになっている。過去のほとんど全ての研究では描画手法として光線追跡法を適用しているが、本研究ではラスタライズ法を用いる。透視投影変換を高速化するために、最適に構築された計算格子上で光線の湾曲を事前に求め、GPUの線形補間機構と併用する。また、動的テッセレーションを用いて高品位な描画像を実現する。この方法はOpenGL/GLSLフレームワークを自然に拡張するものであるから、通常の3D CGと同様の方法でポリゴン・データの描画が可能である。しかもGPUのハードウェア描画機構をフル活用できる。

**キーワード：**GPU、相対性理論、ラスタライズ法、深度バッファ法

## Implementation of OpenGL/GLSL Framework in a Four-Dimensional Black Hole Space-Time

YOSHIYUKI YAMASHITA<sup>1,a)</sup>

**Abstract:** A black hole predicted by the general relativity bends the trajectories of light rays. By calculating their bendings numerically by computers, the three dimensional computer graphics extended into a black-hole space-time becomes more popular recently. Most of all the past researches use the ray-tracing method for rendering while we use the rasterization method in this paper. In order to establish faster perspective projection we preprocess the numerical calculation of the bendings at every parameter on an optimally constructed computational mesh and also utilize the linear interpolation hardware of Graphics Processing Unit (GPU). Furthermore we implement the dynamic tessellation technique for rendering every triangular polygon. Because the rasterization is the common rendering framework of OpenGL/GLSL, we can manage the polygon data in the black-hole space-time in the usual way as in the common three dimensional computer graphics and also we can fully utilize the computational power of GPU's hardware facilities.

**Keywords:** GPU, relativity theory, rasterization, depth buffer method

### 1. はじめに

一般相対理論 [13], [15] によれば、ブラックホールなどの強い重力源は光線の進路を湾曲させる。コンピュータの演算能力の向上に伴い、その湾曲をコンピュータで数値計算し、ブラックホール近傍の様子を3次元コンピュータグ

ラフィックス（CG）として描画する研究が盛んになっている [2], [4], [5], [6], [7], [8], [9], [11], [12], [16], [17], [18], [19], [20], [21]。特に光線の湾曲で被写体が歪んで見える現象は重力レンズ効果（gravitational lens effect）と呼ばれている [15]。しかし、光線の湾曲の数値計算には膨大な時間を要するため、高速な描画が難しい。

著者の知る限り、既存の全ての研究では描画方法として光線追跡法を採用している。これに対して本研究ではラスタライズ法を用いる。これによって光線が湾曲する場合の

<sup>1</sup> 佐賀大学工学系研究科知能情報システム学専攻、佐賀市本庄町 1  
Department of Information Science, Saga University, Honjo,  
Saga-city, Saga 840-8502, Japan

<sup>a)</sup> yaman@is.saga-u.ac.jp

CG を通常の OpenGL/GLSL[10] と同じく、いわゆるポリゴンレンダリングの枠組みで扱うことができることになる。Graphics Processing Unit (GPU) はこの方法をハードウェア化したものであるから、GPU による高速描画も期待できる。後に示すように、1 万枚のポリゴンで構成される被写体であっても実時間描画が可能になる。

本研究ではラスタライズ法をブラックホール時空に拡張するために以下の二つの工夫を行なう。

- (1) 湾曲する光線の軌跡を事前に計算し、結果をディスクに格納しておく。しかし、無限に存在する軌跡全てを格納できる訳ではないので、計算格子を設定し、格子点の有限個の結果のみを格納しておき、格子点以外については線形補間を用いる。現在の GPU は 1GB 近くのメモリを持つため、線形補間であっても十分な計算精度を達成できる。
- (2) 光線が湾曲する場合、一般に三角形ポリゴンは三角形に透視投影されない。そこで動的テッセレーション (dynamic tessellation) [3], [14], [22] を用いる。つまり、シェーダに投入された初期の三角形をより小さな三角形に再帰的に分割し、透視投影する。光線の湾曲の度合いに応じて分割回数を動的に制御する。

この方法によって、演算能力の低い CPU、GPU を用いても高速な描画が可能となる。タブレット端末においても実時間描画が可能になると期待している。

## 2. 関連研究、本研究の新規性

相対論は特殊相対論 (special relativity) と一般相対論 (general relativity) に大別される。

前者では、ローレンツ変換と呼ばれる 4 次元線形座標変換が追加適用される点が通常の CG との主な違いであり、計算量は通常の CG と大差ない。そのため、20 年以上前から種々研究が行なわれ、GPU を用いた実時間画像生成も行なわれている [6], [17]。

後者は前者を拡張した理論であり、光線が湾曲する点が前者との大きな違いである。この湾曲は測地線の方程式 (geodesic equation) と呼ばれる 4 元連立 2 階非線形常微分方程式で記述されるため、一般には数値計算によって解かねばならず、描画には膨大な時間を要する。

著者は 1989 年に当時のスーパーコンピュータ (ピーク性能 2GFLOPS) を用いて一般相対論の CG 画像を 1/30 fps (frames per second) の速度で描画できることを示した [18]。その数値から 600 GFLOPS の演算性能があれば 10 fps 程度の実時間画像生成が達成できると予想されていた。実際、最近のハイエンド GPU はその演算性能を越えており、重力レンズ効果の実時間 CG 画像生成を達成した研究が報告されている [2], [8]。しかし、これらの研究では四角板やトーラス体などの比較的単純な形状の被写体を描画しているに過ぎない。本研究では多数のポリゴンで構成

された被写体を対象とする。

一般相対論に基づく CG の研究は、天文学からのアプローチ [2], [6], [8], [11], [12] と物理教育、エンターテインメントからのアプローチ [4], [5], [7], [9], [16], [17], [18], [19], [20], [21] がある。本研究は後者を目指すものである。

前者では、被写体は高密度星を表す球体、降着円盤を表すトーラスでおおむね十分である。

それに対して後者では、被写体としてロケットなどの子どもにも親しみある被写体が望まれる。ロケットを写実的に描画するには複雑なモデリング・データの描画が必須であるが、これまでそのような描画の試みはほとんどなされていない。仮にそれを既存の技術で行うならば、膨大な時間を必要とする。たとえば著者の既存のプログラムでは 1 枚の画像作成に数時間を要した。これでは実時間描画はおろか、数百枚の CG 画像からなる動画作成も難しい。本研究はそれを可能にする。

## 3. プログラムの概要

次節以降に詳細を述べていくが、その前に本研究で開発した CG プログラムの概要を述べておく (図 1 参照)。

### 3.1 事前計算

次節で述べるように、本研究では光線の湾曲をあらかじめ計算し、ハードディスクに格納しておく (図 1 の左半分を参照)。無限に存在する全ての光線について計算できる訳ではないから、あらかじめ計算格子を設定し、格子点についてのみ事前計算を行なう。格子点から外れた点の計算値は格子点の計算結果から線形補間で求める。

補間誤差を最小にするためには、計算格子の最適な設定が重要である。そこで、計算格子を決定するパラメータの調整と計算をフィードバックしながら繰り返し、より良い格子点を設定し、より補間誤差の少ない計算結果をハードディスクへ格納する。

### 3.2 描画処理

実際の描画は、通常の OpenGL/GLSL の描画方法と同様に、GPU のシェーダに投入された三角形ポリゴンについて行なう (図 1 の右半分を参照)。

まず、事前に格子点上で計算されている光線の湾曲の結果を透視投影変換に利用するため、シェーダの 3 次元テクスチャにロードしておく。

バーテックス・シェーダに投入された三角形ポリゴンは透視投影変換後、直ちにジオメトリ・シェーダへ渡される。ジオメトリ・シェーダではバーテックス・シェーダから受け渡された三角形ポリゴンを細分化 (テッセレート) するか否かを判定する。必要に応じて細分化を再帰的に繰り返し、確定した三角形を出力する。出力された三角形はラスタライザーで複数の画素データへ変換され、フラグメント・

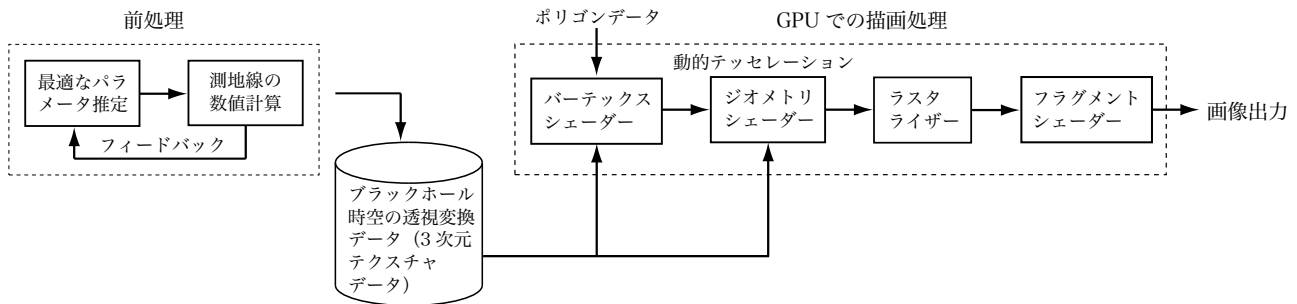


図 1 前処理と描画処理

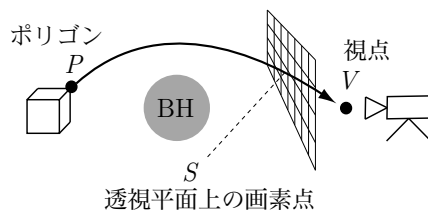


図 2 ポリゴン、透視平面、視点の関係

シェーダで画素点に着色される。ラスタライザ以降の処理内容は通常の OpenGL/GLSL の処理内容とほとんど同じである。

## 4. ブラックホール近傍でのコンピュータグラフィックス

### 4.1 幾何モデル

本研究の幾何モデルは図 2 の通りである。被写体上の点  $P$  から発射された光線が透視投影平面上の点  $S$  を通過して視点  $V$  に入ると仮定する。このとき、ブラックホールによる光線の湾曲は以下の 2 階非線形微分方程式（測地線の方程式と呼ばれている）で記述される [15]。

$$\begin{aligned}\ddot{t} &= -\frac{a}{r^2} \left(1 - \frac{a}{r}\right)^{-1} \dot{t}\dot{r} \\ \ddot{r} &= -\frac{1}{2} \left(1 - \frac{a}{r}\right) \left(\frac{a\dot{t}^2}{r^2} - \frac{a\dot{r}^2}{(r-a)^2} - 2r\dot{\phi}^2\right) \\ \ddot{\phi} &= -\frac{2\dot{r}\dot{\phi}}{r}\end{aligned}\quad (1)$$

この方程式の数値計算には 4 次のルンゲクッタ法を用いるのが一般的である。また以下では  $a = 1$  とする。

### 4.2 光線追跡法

光線追跡法では図 2 の過程を逆に数値計算する。すなわち、 $V$  から過去に向かって発射され、 $S$  を通過するような光線が被写体と最初に交差する点  $P$  を求め、投影平面上の  $S$  に  $P$  の色、輝度を着色する\*1。この場合、式 (1)

\*1 紙面の都合で詳細は述べないが、このとき光の波長の赤方偏移/

表 1 描画手法と微分方程式の解法の関係

| レンダリング手法 | 光線追跡法  | ラスタライズ法 |
|----------|--------|---------|
| 所与の条件    | $V, S$ | $V, P$  |
| 求める値     | $P$    | $S$     |
| 解くべき問題   | 初期値問題  | 境界値問題   |

を「 $V$  から発射され、 $S$  を通過する」という条件で解くこと、すなわち微分方程式の初期値問題を解くことになる（表 1 参照）。

### 4.3 ラスタライズ法

ラスタライズ法では、ポリゴンの頂点  $P$  から発射され、 $V$  へ入る光線が投影平面と交差する点  $S$  を求める。そして投影平面上の  $S$  に  $P$  の色、輝度を着色する。この場合、式 (1) を「 $P$  から発射され、 $V$  へ入る」という条件で解くこと、すなわち微分方程式の境界値問題を解くことになる（表 1 参照）。一般的に、境界値問題の計算量は初期値問題の計算量を遥かに上回るから、描画時に同時に境界値問題を解くことは現実的ではない。本研究では、あらかじめ必要な計算を行ない、結果をディスクに保存しておく（図 1 参照）。

なお、ポリゴンの内部点については、そのポリゴンを構成する頂点群の計算結果から投影平面上の位置、色、輝度を補間によって求める（これには GPU のラスタライザを用いることができる）。

光線追跡法、ラスタライズ法と微分方程式の解法の間を表 1 にまとめておく。

## 5. テクスチャを用いた光線の湾曲の近似計算

ラスタライズ法の問題点のひとつは、境界値問題の数値計算には初期値問題のそれに比べ膨大な時間が掛かることである。これを解決するために、様々なパラメータ値の組み合わせについて境界値問題をあらかじめ計算しておき、描

青方偏移も同時に計算し、色、輝度に反映する。

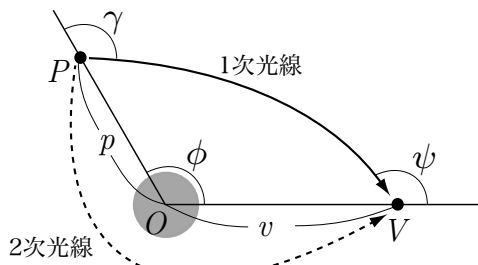


図 3 球対称ブラックホールの幾何モデル

画時にその結果を参照する方法を用いる (図 1 参照)。もちろん、無限に存在する全てのパラメータ値の組み合わせについて事前計算ができる訳ではないから、その組み合わせについては慎重に選択し、存在しない組み合わせについては近傍の値から補間して求める。

かつて著者がこの方法を実験した [19] ときにはコンピュータ (スーパーコンピュータであっても) のメモリ容量が十分ではなく、補間精度が悪く、研究を断念せざるを得なかった。しかし、現在の GPU は 1GB 程度のメモリを持つものはめずらしくない。後に示すように 1GB を使用できるならば、目視で認識できないくらいの精度は達成できる。

### 5.1 使用する 3 次元テクスチャの要件

光線が湾曲する場合のラスタライズ法では、次の 3 種類のデータ (図 3 参照) :

- (1) ブラックホール中心から視点  $V$  までの距離  $v$
- (2) ブラックホール中心からポリゴン頂点  $P$  までの距離  $p$
- (3) 視点、ブラックホール中心、ポリゴン頂点のなす角  $\phi$  を引数として、次のデータ 4 種類のデータ :
- (1) 光線の視点への入射角  $\psi$
- (2) 光線のポリゴン頂点からの発射角  $\gamma$
- (3) ポリゴン頂点から視点までの距離 (深度値)  $d$
- (4) ポリゴン頂点と視点における光線のエネルギー比  $e$  を知る必要がある。つまり以下の写像である。

$$(v, p, \theta) \rightarrow (\psi, \gamma, d, e)$$

本研究では、これを GPU 上の 3 次元テクスチャデータとして実装する。今、 $(v, p, \theta)$  の各次元のテクスチャサイズ<sup>\*2</sup>を  $2^i$ 、 $2^j$ 、 $2^k$  とするとき、テクスチャに保存できる光線の湾曲のデータは  $2^{i+j+k}$  個の  $(v, p, \theta)$  についてである。これら組み合わせについては境界値問題を数値的に解き、テクスチャデータとして利用する。それ以外の点については保存されたデータから線形補間によって計算する。

ところで、ブラックホール時空では、ポリゴン頂点  $P$  から視点  $V$  へ入る光線は複数存在しうる。今、 $P$  から  $V$  へ至る光線の中で、ブラックホールを半周 ( $180^\circ$ ) 未満しか

<sup>\*2</sup> GLSL では通常、テクスチャサイズは 2 のべき乗である。

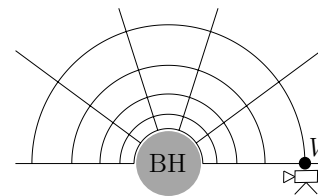


図 4 ブラックホールに境界を持つメッシュ

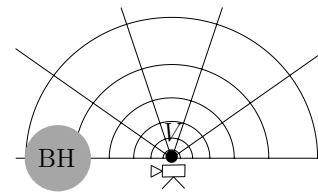


図 5 視点に境界を持つメッシュ

周回しない光線を 1 次光線<sup>\*3</sup> (primary ray) と呼ぼう。これは図 3 の実線に相当する。また、ブラックを半周以上 1 周未満しか周回しない光線を 2 次光線 (secondary ray) と呼ぼう。これは図 3 の点線に相当する。同様に 3 次以上の光線も考えることができるが、CG 画像にはほとんど寄与しないからここでは扱わない。我々の CG では 1 次および 2 次の光線を扱う必要がある。

### 5.2 最適な計算格子の設定

今、湾曲のデータを計算するパラメータの組み合わせ  $(v, p, \theta)$  を 3 次元計算格子 (計算メッシュ) の格子点としよう。このとき、補間誤差をできるだけ小さくするような計算格子の設定が望まれる。特に、格子の境界 (3 次元格子の表面) において高い計算精度を維持するためには、1 次光線については計算格子がブラックホール近傍で図 4 のような形状であり、視点近傍では図 5 のような形状であることが要請される。そこで両方の特徴を併せ持つ図 6 の形状の計算格子を採用する (詳細は付録 A.1 参照)。ただし補間誤差を最小にするには、実際の計算格子の疎密は図 7 のような偏ったものすべきことが分かっている。また、2 次光線についてはブラックホール近傍で図 4 のような形状であることが要請される。この場合も計算格子の最適な疎密は自明でないが、詳細は省略する。

なお、テクスチャの必要メモリ量は  $16 \times$  格子点数  $= 16 \times 2^{i+j+k}$  バイトである。たとえば  $i+j+k = 24$  とするとき必要メモリ量は 256MB となる。最近の GPU のメモリが 1GB 内外であることから、 $i+j+k$  の値は高々 25 程度であろう。この制約の中で十分な計算精度を達成せねばならない。

### 5.3 誤差最小のメッシュ

3 次元テクスチャデータから光線の湾曲のデータを読み出した際の線形補間誤差を最小にするために、計算格子の疎

<sup>\*3</sup> 光線が湾曲しない場合には全ての光線は 1 次光線である。

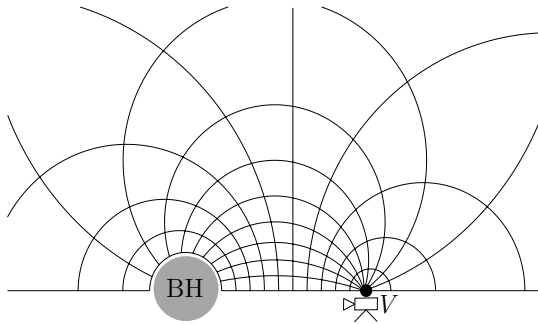


図 6 ブラックホールと視点に境界を持つメッシュ

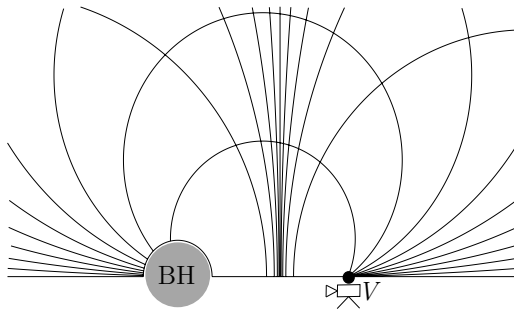


図 7 ブラックホールと視点に境界を持つメッシュ (最適な形状)

表 2 メッシュサイズと線形補間誤差

| 必要メモリ<br>(MB) | 1 次光線の誤差 |       | 2 次光線の誤差 |       |
|---------------|----------|-------|----------|-------|
|               | 最大       | 平均    | 最大       | 平均    |
| 4             | 4.658    | 0.234 | 0.044    | 0.005 |
| 8             | 3.660    | 0.034 | 0.041    | 0.004 |
| 16            | 1.630    | 0.026 | 0.033    | 0.004 |
| 32            | 1.630    | 0.018 | 0.029    | 0.002 |
| 64            | 0.782    | 0.008 | —        | —     |
| 128           | 0.663    | 0.007 | —        | —     |
| 256           | 0.375    | 0.005 | —        | —     |
| 512           | 0.226    | 0.002 | —        | —     |

誤差の単位は度数

密を様々に変えて、約 500 万点の  $(v, p, \theta)$  ( $v, p \in [a, 10^3a]$ ,  $\theta \in [0, \pi]$ ) について、補間誤差を測定した。また各次元の格子のサイズ  $2^i + 2^j + 2^k$  も様々に変えて、補間誤差を測定した。そして、3 次元テクスチャデータの必要メモリ容量を固定したときに (すなわち  $i + j + k$  を固定したときに)、500 万点の  $(v, p, \theta)$  の中の最大の誤差を最小するような  $2^i + 2^j + 2^k$ 、および格子の疎密を求めた。そのときの光線の入射角  $\psi$  の最大誤差、平均誤差を表 2 にまとめた。

今、ユーザが水平方向の大きさ 100cm のモニタ・ディスプレイ上で CG 画像をみるとき、目視でディスプレイ上の被写体の位置の違いに違和感を感じる距離を高々 5 mm としよう。次に CG 画像の水平方向の画角を  $40^\circ$  とするならば、5 mm は角度にして  $0.02^\circ$  に相当する。

表 2 の最大誤差は 1 次光線ではメモリ 512MB を使用するとき  $0.02^\circ$  と同程度の誤差になっている。2 次光線については、メモリ 4MB を試用する場合でも最大誤差は  $0.02^\circ$  以下である。以上のことから、本研究の方法を用い

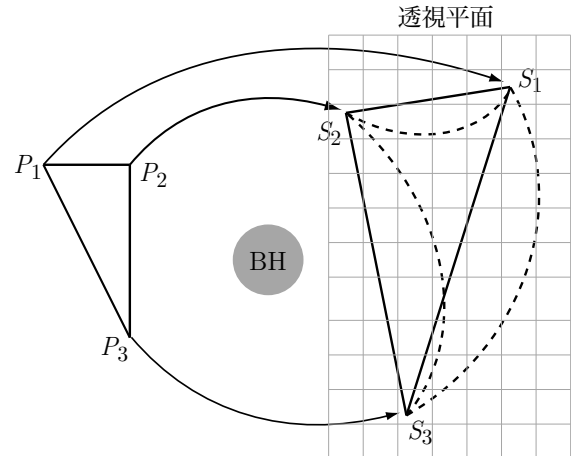


図 8 光線が湾曲する場合の三角形ポリゴンの投影

ても、誤差による違和感はほとんど感じないと言えよう。

## 6. 動的テッセレーションの導入

### 6.1 問題点

光線が湾曲しない場合、3 次元空間内の三角形は、投影された透視平面上でも三角形である。それゆえ、投影された 3 頂点を透視平面内でラスタライズすれば、正しい描画像を得ることができる\*4。

しかし光線が湾曲する場合には、これは成り立たない。

たとえば図 8 では、ポリゴン頂点  $P_1, P_2$  が透視平面上の  $S_1, S_2$  へ投影されている。このとき、ブラックホールによる光線の湾曲のため、線分  $P_1P_2$  は投影平面上では曲線をなすかもしれない (図 8 の  $S_1$  と  $S_2$  を結ぶ点線)。線分  $P_2P_3, P_1P_3$  についても同様である。よって、3 頂点  $P_1, P_2, P_3$  が定める三角形を 3 頂点  $S_1, S_2, S_3$  が定める三角形へ投影し、ラスタライズすることは問題が多い。特に光線の湾曲の影響が極端に現れる場合、たとえば三角形がブラックホールを挟んで反対側に位置する場合には、透視平面上では円弧 (アインシュタイン・リング [15] と呼ばれる) に投影されるべきであり、3 頂点から単純に三角形を作るべきではない。

### 6.2 三角形の分割

上の問題を解決するために、三角形ポリゴンをより小さな三角形ポリゴン群に分割する手法 (テッセレーション) [22] を行なう。

たとえば、図 8 の場合、 $P_1, P_2, P_3$  の中点  $P_4, P_5, P_6$  を求める。そして 4 枚の三角形をそれぞれ描画すれば図 9 のような、より誤差の小さな投影が実現できる。これを再度繰り返せば、図 10 のようになり、描画精度はさらに上がる。

\*4 ただし陰影付けは補間によって行なうため、完全に正しい描画像とは言えないが、グローシェーディングやフォンシェーディングを用いるならば、大きな問題にはならない。

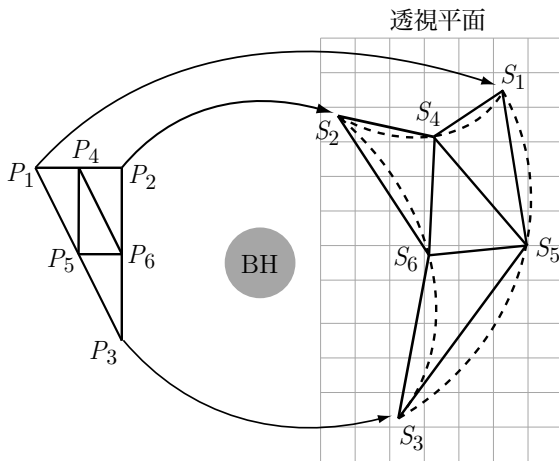


図 9 三角形ポリゴンの分割 (1 段階)

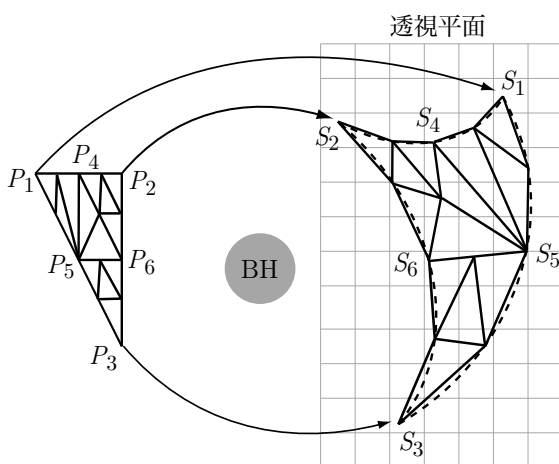


図 10 三角形ポリゴンの分割 (2 段階)

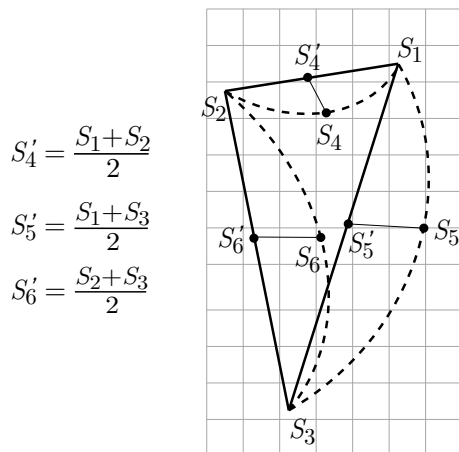


図 11 三角形ポリゴンの分割の要否の判定

### 6.3 分割の要否の判定

文献 [3] と同様に、三角形の分割は再帰的に繰り返せばよい。

分割を止める条件は次の通りである (図 9、図 11 参照)。今、三角形ポリゴンの頂点  $P_1$ 、 $P_2$ 、その中点  $P_4 = (P_1 + P_2)/2$  がそれぞれ  $S_1$ 、 $S_2$ 、 $S_4$  へ投影されたと仮定する。次に、 $S_1$  と  $S_2$  の中点  $S'_4 = (S_1 + S_2)/2$

表 3 ポリゴンデータ毎の CG 画像生成枚数/秒

| データ名            | 頂点数   | ポリゴン数 | 枚/秒  |
|-----------------|-------|-------|------|
| アポロ指令船 (図 12)   | 4594  | 9100  | 17.5 |
| スペースシャトル (図 13) | 28318 | 27096 | 6.9  |

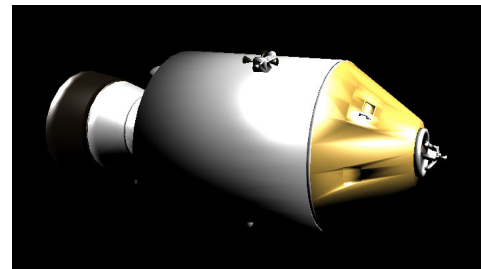


図 12 アポロ指令船のモデリング・データ

を求める。もし  $|S_4 - S'_4|$  が 1 画素未満ならば、辺  $P_1P_2$  の分割を行なわない。さもなくば  $P_4$  を新たな分割点として登録する。同様の判定を辺  $P_1P_3$ 、 $P_2P_3$  についても行なう。そして、もしどの辺にも新たな分割がない場合には再帰を終了する。分割がある場合には文献 [3] と同じ方法で再帰分割を続ける。

ただし本研究では、再帰がある一定の深さ以上になる場合には分割を停止し、そのポリゴンを描画しない<sup>\*5</sup>。アインシュタイン・リングが形成されるような場合がこれに該当し、そのような場合には三角形を描画すべきでない。

### 6.4 シェーダへの実装

現在、動的テッセレーションをジオメトリ・シェーダに実装している (図 1 参照)。

## 7. 画像生成実験

アポロ司令船 (図 12 参照)、スペースシャトル (図 13 参照) のモデリング・データについて描画実験を行なった。

まず、モデリング・データ (今回は 3DS 形式のデータを使用) を描画プログラムで読み込み、ポリゴン・データの RGB 値を色温度へ変換しておく。これによって特殊相対論的効果である光のドップラー効果、一般相対論的効果である重力による光の赤方偏移を色温度の変化として扱うことができるようになる。そして、ポリゴン・データをバーテックス・シェーダへ渡して描画を行なう。

図 14 に、アポロ司令船がブラックホールをはさんで視点の反対側にあるときの描画例を示す。司令船の像が重力レンズ効果によって歪んでいる様子が分かる。像は円弧上になっているが、これはアインシュタイン・リングと呼んでいる [15]。

表 3 に、それぞれのポリゴンデータの頂点数、三角板ポリゴン数、そして画像生成枚数を載せる。画像生成枚数は、視点とブラックホールを固定し、ポリゴンデータをブ

<sup>\*5</sup> シェーダプログラムが使用できる作業メモリ、描画速度等を勘案し、本研究では再帰分割の最深度を 7 としている。

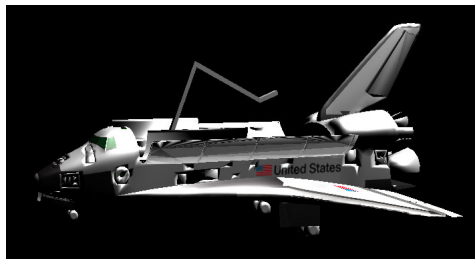


図 13 スペースシャトルのモデリング・データ

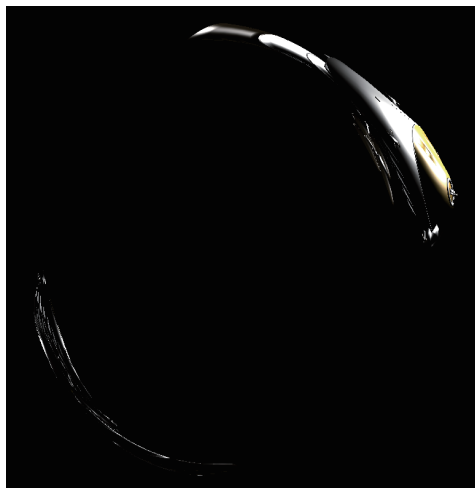


図 14 アポロ指令船によるアインシュタイン・リングの例

ラックホールを周回させたときの 1 秒間の平均画像生成枚数 (fps=frame per second) である。ポリゴン数が約 1 万枚のアポロ指令船では 18fps を達成しており、実時間描画が可能である。ポリゴン数が約 3 万枚のスペースシャトルでは 7fps であり、実時間描画にはやや足りない。生成時間はおおよそポリゴン数に比例していることも分かる。

なお、著者はすでに光線追跡法で同等の映像を生成するプログラムを作成済みである [21]。Apple Mac Pro 3 台 (Intel Xeon 28 コア) を用いて MPI 環境下でスペースシャトルの画像生成を並列実行した場合、1 枚当たり約 5 時間の生成時間を要した。これは、 $5.5 \times 10^{-5}$  fps に相当する。単純な比較はできないものの、本研究の生成速度 (表 3 の 6.9 fps) はそれに比べ、約 10 万倍高速である。

## 8. 今後の課題

今回、テッセレーションを導入することで、光線が湾曲する場合のラスタライズ法の問題を解決できた。今回はその実装をジオメトリ・シェーダで行なっているが、テッセレーション処理は本質的に逐次的な処理であるから、高速描画を阻害されている [1]。それを解決するために最新の GLSL、Direct 11 ではシェーダにテッセレータ・フェーズが導入されている [10]。今後はその利用を検討したい。

現在、本研究内容をタブレット端末へ移植中である。相対論 CG がタブレット端末上で動くことのインパクトは大

きい (教育効果は高い) と考えている。タブレット端末の GPU は非力である、OpenGL の機能が低いなど問題は多いが、ひとつずつ解決していく予定である。

## 参考文献

- [1] D. Blythe: The Direct3D 10 system, Proc. SIGGRAPH '06, ACM (2006) pp.724-734.
- [2] C. Chan, D. Psaltis, And F. Özel : GRay: a Massively Parallel GPU-Based Code for Ray Tracing in Relativistic Spacetimes, arXiv:1303.5057v1 (2013).
- [3] A. J. Chung and A. J. Field: A Simple Recursive Tesselator for Adaptive Surface Triangulation, Jour. of Graphics Tools, Vol. 5 (2000) pp.1-9.
- [4] A. J. S. Hamilton: Inside Black Holes, <http://jila.colorado.edu/ajsh/insidebh/index.html>.
- [5] A. J. S. Hamilton and G. Polhemus: Stereoscopic Visualization in Curved Spacetime: Seeing Deep inside A Black Hole, New Jour. Phys., Vol 12 (2010) 123027.
- [6] F.W. Hehl, R.A. Puntigam, H. Ruder, eds.: *Relativity and scientific computing : computer algebra, numerics, visualization*, Springer (1996).
- [7] D. Kuchelmeister, T. Müller, M. Ament, G. Wunner, and D. Weiskopf: GPU-Based Four-Dimensional General-Relativistic Ray Tracing, Computer Physics Communications Vol. 183 (2012) pp. 2282-2290
- [8] T Müller and J. Frauendiener: interactive visualization of a thin disc around a Schwarzschild black hole, Eur. J. Phys. 33 (2012) 955.
- [9] R. Nemiroff: Virtual Trips to Black Holes and Neutron Stars, [http://apod.nasa.gov/htmltest/rjn\\_bht.html](http://apod.nasa.gov/htmltest/rjn_bht.html).
- [10] The OpenGL Shading Language, Version: 4.40, <http://www.opengl.org> (2014).
- [11] D. Psaltis and T. Johannsen: A Ray-tracing Algorithm for Spinning Compact Object Spacetimes with Arbitrary Quadrupole Moments. I. Quasi-Kerr Black Holes, The Astrophysical Journal, Vol. 745, No. 1 (2012).
- [12] D. D. Schnittman, J. H. Krolik, and J. F. Hawley: Light Curves from An MHD Simulation Of a Black Hole Accretion Disk, The Astrophysical Journal, Vol. 651 (2006) pp. 1031-1048.
- [13] J. L. Synge : *Relativity: The General Theory*, North-Holland Pub. (1960).
- [14] A. Tatarinov: Instanced Tessellation in DirectX10. Game Developers Conference 2008 (2008).
- [15] E. F. Taylor and J. A. Wheeler : *Exploring Black Holes: Introduction to General Relativity* Addison Wesley Longman (2000).
- [16] F. H. Vincent, T. Paumard, E. Gourgoulhon, and G. Perrin: GYOTO: A New General Relativistic Ray-Tracing Code, Classical and Quantum Gravity, Vol. 28 (2011) 225011.
- [17] D. Weiskopf, et al. : Explanatory and Illustrative Visualization of Special and General Relativity, IEEE Trans. on Visualization and Computer Graphics, VOL. 12, NO. 4 (2006) pp.522-534.
- [18] 山下義行 : ブラック・ホールのコンピュータグラフィックス : 光線追跡法の曲がった 4 次元時空への拡張、情報処理学会論文誌、30-5 (1989) pp.642-651.
- [19] 山下義行 : ブラックホール近傍におけるデブスバッファ・アルゴリズム、情報処理学会第 43 回全国大会論文集、第 2 分冊 (1991) pp.487-488.
- [20] 山下義行 : 相対論の CG 静止画・動画集、<http://www.fu.is.saga-u.ac.jp/~yaman/RCG/index.html>.

- [21] 山下義行：一般相対論汎用CGプログラムの開発、情報処理学会第53回全国大会論文集、第4分冊(1996) pp.109-110.
- [22] D. Zorin, and P. Schröder : Subdivision for Modeling and Animation, ACM SIGGRAPH Courses Notes (2000).

## 付 録

### A.1 テクスチャ座標への写像

ブラックホールの半径を  $r_{bh}$  とし、視点の位置を  $(r_v, 0)$  とする (ただし  $0 < r_{bh} < r_v$ )。このとき、平面上の座標点  $(p, q)$  (ただし、 $p^2 + q^2 \geq r_{bh}^2$ 、 $q \geq 0$ ) からテクスチャ矩形内の点  $(u, v) \in [0, 1] \times [0, 1]$  への写像を以下のように定義する。

#### A.1.1 $(p, q)$ から $u$ への写像

中心が  $x$  軸上の点  $(x_u, 0)$  である円

$$(x - x_u)^2 + y^2 = r_u^2$$

を考える。ここに  $x_u$ 、 $r_u$  は以下の通りとする。

$$\begin{aligned} x_u &= r_u + u \cdot r_{bh} + (1 - u) \cdot r_v \\ r_u &= \left( \frac{u}{1 - 2u} \right) (u \cdot r_{bh} + (1 - u) \cdot r_v) \end{aligned}$$

このとき、点  $(p, q)$  は、唯一の  $u \in [0, 1]$  によってこの円の上の点となり、 $u$  は2次方程式の解として計算できる。

#### A.1.2 $(p, q)$ から $v$ への写像

座標原点  $(0, 0)$ 、視点  $(r_v, 0)$ 、座標点  $(p, q)$  の3点を通る円

$$x^2 - r_v \cdot x + y^2 - \left( \frac{p^2 - r_v \cdot p + q^2}{q} \right) y = 0$$

を考える。この円が  $x$  軸と交わる角度  $\theta \in [0, \pi]$  は以下の式で計算できる。

$$\theta = \arctan \left( \frac{r_v \cdot q}{p^2 - r_v \cdot p + q^2} \right)$$

この値に  $1/\pi$  を乗じ、 $[0, 1]$  に正規化した値を  $v$  とする。

このように、 $(p, q)$  から  $(u, v)$  への計算は四則演算と初等関数のみで可能であるから、シェーダへの実装に困難はない。