GPUを用いたブラックホール時空のリアルタイム・コンピュータグラフィックス

山下 義行[1,a)]

概要：一般相対論によればブラックホール周辺では強い重力によって光線が曲がる。そこでレイトレーシング法を光線が湾曲する場合に拡張することでブラックホール時空下での CG を実現できる。しかし、この方法では光線の軌跡の計算に膨大なコストが掛かる。本研究では、画像生成に必要な光線の湾曲のデータを GPU にあらかじめ格納しておく。そして実際の描画では GPU で各頂点毎に光線の湾曲を線形補間で求め、Z バッファ法で隠面消去する。これによってリアルタイムでの CG 画像生成が可能になった。

キーワード：グラフィックプロセッサ、画像生成、シミュレーション・応用計算の実装と評価

GPU-Based Real Time Computer Graphics in a Black Hole Space-Time

Yoshiyuki Yamashita[1,a)]

Abstract: The theory of general relativity says that a light ray bends by strong gravity around a black hole. Simulating the light ray bending and extending the common ray tracing method, we can generate CG images under the curved space-time. However this method requires huge computing resources. In this research we calculate ray bending and store the necessary data in a GPU memory beforehand. Then in actual image generation GPU obtains the ray bending information of each polygon vertex by interpolating the stored data and performs the Z buffer method. This new rendering method attains real-time CG in the curved space-time.

Keywords: Graphics processing unit, Image generation, Simulations and computing applications

1. はじめに

アインシュタインが提唱した相対性理論（以下、簡単に相対論）[4] は我々の常識とかけ離れた現象を予言しており、専門家のみならず、多くの人々の興味を引き付けてやまない。その現象の中でもブラックホール（black hole）は特に強い興味を集めており、これをコンピュータでシミュレートし、コンピュータグラフィックス（以下、CG）映像として表現する試みが 1980 年代後半から種々行われている [1], [3], [5], [7]。

ブラックホール周辺では強い重力によって光線が曲がり、それが様々な奇妙な現象を引き起こす。それらを映像化するにはよく知られたレイトレーシング法 [2] を光線が湾曲する場合に拡張すればよい。しかしこの方法では、画素毎に光線の湾曲を一般相対論の微分方程式に従って計算せねばならず、膨大な計算コストのため、映像をリアルタイムに生成することは困難であった。

これを解決すべく、本研究では、光線の湾曲を事前に計算、保存しておき、それを描画時に Graphics Processing Unit（以下、GPU）を用いて高速に参照する方法を提案する。この方法を用いてブラックホール時空のリアルタイム画像生成が初めて可能となった。

類似の研究として、特殊相対論のローレンツ変換を GPU で行いてリアルタイム描画を実現した研究や、ブラック

[1] 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
[a)] yaman@is.saga-u.ac.jp

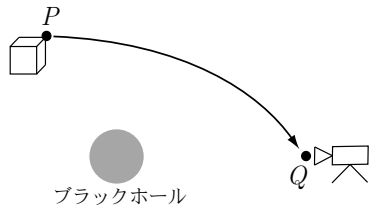

図 1 ブラックホール近傍でのレイトレーシング

ホールの重力レンズ効果に特化して背景画像の歪みを高速に描画した研究がある [1], [3] が、本研究ではそのような制限は加えず、任意の 3 次元被写体の描画を扱う。

以下、2 節では Z バッファ法を用いる本研究のアイディアを示す。3 節ではそのアイディアを具体化し、ブラックホール時空のための計算格子の設定法、格子上の光線の軌跡の計算方法などを述べる。4 節では 3 節の計算格子を改良する方法を述べる。4 節の改良によってシミュレーション誤差の少ない CG 画像が可能となる。5 節では三角形ポリゴンを描画する際の新たな問題と暫定的な解法を述べる。なお、実装言語として OpenGL/GLSL を用いる。開発した描画プログラムは、Mac Pro、MacBook Pro、MacBook Air 上で動作する。

2. 光線の湾曲と２種類の描画法

2.1 レイトレーシング法

いま、ブラックホール近傍にポリゴンがあり、そこから少し離れた位置にそのポリゴンを写すカメラがあると仮定する（**図 1** 参照）。ポリゴン頂点 P から放射された光線はブラックホールの重力によって湾曲し、カメラの視点 Q へ到達し、それが CG 画像の 1 画素の色を決定する。

湾曲する光線の軌跡の具体的な形状は 4 次元座標上の 4 元連立非線形 2 階常微分方程式で規定される。たとえば以下は、最も単純な、球対称で自転していないブラックホール[*1]の場合の方程式である。ただしここでは 4 次元極座標系 (t, r, θ, ϕ) を用い、さらに簡単のため、$\theta = \pi/2$ の平面上に限定したため、3 元連立方程式になっている。

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

$\dot{t}$、$\dot{r}$ 、$\dot{\phi}$ は軌跡のパラメータ λ による 1 階微分 $dt/d\lambda$、$dr/d\lambda$ 、$d\phi/d\lambda$ を表わし、$\ddot{t}$、$\ddot{r}$、$\ddot{\phi}$ は 2 階微分を表わす。定数 a はブラックホールの半径である。これは、光さえもそこから脱出できない球の半径を意味する。

視点に入射する光線の軌跡を知るには、視点の位置、視点での光線のベクトルを初期条件として、この微分方程式の初期値問題を解く。そして方程式を解きながら湾曲する光線と三角形ポリゴンとの交差判定を繰り返す。光線がポリゴンに交差することが判明したならば、その後の処理は通常のレイトレーシングと同様である。ただし重力および相対速度による光線の波長の変化を別途加味する。

[*1] Schwarzschild ブラックホールと呼ばれている。

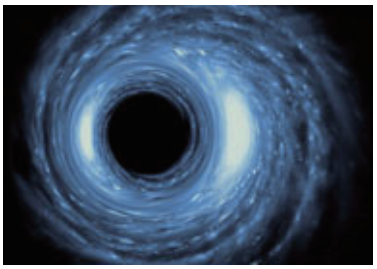

図 2 渦巻き銀河を背景とする重力レンズ効果の CG 画像

上に述べた方法では画像の画素毎に微分方程式を解くため、計算量が膨大である。たとえば**図 2** の CG 画像[*2][6] 1 枚を作るために、Mac Pro 4 台（Intel Xeon コア 29 基、クロック周波数 2.4〜2.8 GHz）を使用した並列計算において約 20 秒を要し、リアルタイムの画像生成は困難である。

2.2 Z バッファ法

レイトレーシング法による計算時間を解決するために、本研究では GPU を利用し、Z バッファ法を光線が湾曲する場合に拡張する。Z バッファ法はほとんどの GPU がサポートする描画手法であり、高速な画像生成が期待できる。

Z バッファ法では、まず、描画するポリゴンの頂点が透視投影される投影平面上の位置を求めねばならない。光線が湾曲する場合にはこの透視投影の計算は微分方程式の境界値問題（3.4 節参照）を解くことに相当し、計算量はレイトレーシング法の初期値問題を解くよりもさらに増大する。そこで本研究では、様々な軌跡について透視投影のデータを事前に準備しておく。そして描画時には、準備したデータを GPU へ転送・格納しておき、必要な投影データは格納されたデータから補間によって求める。この方法では描画時の計算は補間だけであるから高速な描画が期待できる。もし線形補間を用いるならば GPU のハードウェアによって高速な補間計算もできる。

投影変換後の処理は通常の Z バッファ法と同様であるが、追加処理が必要な場合がある（5 節参照）。

3. 透視投影データと計算格子

3.1 透視投影データ

事前に準備すべき透視投影データについて検討する。

原理的には、ポリゴンの頂点 P を出発し、視点 Q へ到達する光線（図 1 参照）が視点へ入射する方向が分かれば投影位置が計算できる。Z バッファ法では Z 値（深度値）も必要であるが、これには点 P、Q 間の軌跡のパラメータ λ（2.1 節参照）の増分値を用いればよい。相対論に固有の

[*2] 渦巻き銀河の原画像は沼澤茂美氏によるイラストである。

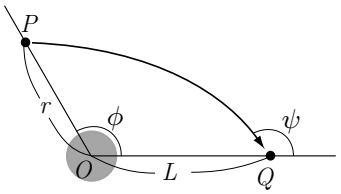

図 3 球対称ブラックホールにおける透視投影データ

データとして点 P、Q 間の重力ポテンシャルの差に起因する波長の変化率も必要であるが、これは P と Q における 4 次元光線ベクトルの時間成分の比率に等しい[*3]。つまり、Z 値と波長の変化率は微分方程式を数値計算した副産物として求めることができる。

今、点 P、Q の位置を一意に表わすために必要な浮動小数点数値（以下これを `float` と表す）の数を m、入射方向を一意に表わすために必要な `float` の数を n と仮定する。Z 値、波長の変化率のために 2 個の `float` も必要である。よって投影変換のためには m 個の `float` から $n+2$ 個の `float` を求める仕組みを作ればよい。もちろん無限個の投影データを準備することはできないから、実装上は有限の大きさの**計算格子**（computational mesh）を想定し、各格子点の投影データを $n+2$ 個の `float` 型 m 次元配列に格納する。そして描画時には補間を行い、目的とする投影データを得ればよい。

さて、球対称で自転していないブラックホールでは $m=3$、$n=1$ でよい（**図 3** 参照）。なぜならば、時空の対称性から、視点 Q とブラックホールの距離 L、頂点 P とブラックホールの距離 r、および線分 OP と OQ のなす角 ϕ が与えられれば、光線の軌跡は一意に定まり、結果、光線の視点への入射角 ψ を求めることができるからである。しかも $m=3$ であるから GPU の 3 次元線形補間（trilinear interpolation）のハードウェアを利用できる。この事実に気づいたことが著者の直接の研究動機である。

以下では、球対称で回転していないブラックホールと線形補間を暗黙に想定し、内容を具体化していく[*4]。

3.2 2 種類の計算格子

図 3 の ψ を関数 $\psi(r,\phi,L)$ と表そう。このとき $\psi(L,0,L)$ の値は不定である（頂点 P、Q が同じ位置のため ψ の値を定義できない）。また、視点の近傍において

$$\psi(L-0,0,L)=\pi,\quad \psi(L+0,0,L)=0$$

となり、この微小な領域で最悪 π $(=180°)$ の補間誤差が生じうる。この誤差は深刻である。これを解決するために、本研究では r の範囲を $r\le L$ と $L\le r$ の二つの領域に分割し、ψ の計算を 2 種類の計算格子に分けて行う。

[*3] 相対論では 4 次元光線ベクトルの時間成分はその光線のエネルギーに相当する。
[*4] 2 次補間（cubic interpolation）についても適用を検討したが、近似が行き過ぎるため、むしろ誤差が増す結果となった。

表 1 2 種類の計算格子の各次元の範囲

	内側	外側
r の範囲	$[1+\epsilon, L]$	$[L, r_{max}]$
ϕ の範囲	$[0,\pi]$	$[0,\pi]$
L の範囲	$[1+2\epsilon, r_{max}/2]$	$[1+2\epsilon, r_{max}/2]$

ここに $\epsilon=0.005$、$r_{max}=10^4$ と設定

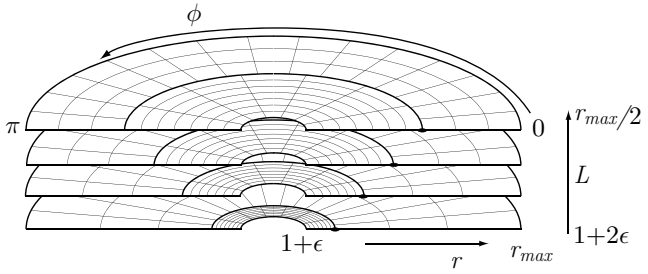

図 4 2 種類の 3 次元計算格子

以下では $r\le L$ の計算格子を**内側**（inside）の計算格子、$L\le r$ のそれを**外側**（outside）の計算格子と呼ぶ。なお、補間計算時に区間の端点の値として $\psi(L,0,L)$ の値が必要になる場合があるが、その値はそれぞれの格子における極限値 $\psi(L\pm 0,\phi,L)$ として定義する。

次に、この 2 種類の計算格子の定義域について検討する。まず簡単化のため、次の約束をする。

- ブラックホールの半径 a（2.1 節参照）を 1 とするような長さの単位系を用いる。

この半径内にポリゴンを置くことは無意味であるから、本研究では $r>1$ の範囲を扱う。具体的には $r\ge 1+\epsilon$ とし、本研究では $\epsilon=0.005$ と設定した。次に、原点から r_{max} 以遠にポリゴンを置かない、すなわち $r\le r_{max}$ と約束し、本研究では $r_{max}=10^4$ と設定した。角度 ϕ の範囲は時空の球対称性から $[0,\pi]$ でよい。視点は移動可能とし、ここでは L の範囲を r の範囲よりも狭く、$1+2\epsilon\le L\le r_{max}/2$ と設定する。

表 1 に内側と外側の計算格子の範囲をまとめた。r の範囲が L に依存することに注意してほしい。$r=L$ の時には内側、外側どちらの格子を用いてもよいとする。また以下では $1+\epsilon$ を r_{min} と表わし、$1+2\epsilon$、$r_{max}/2$ をそれぞれ L_{min}、L_{max} と表す。**図 4** に計算格子を図解で示す。この図では、内側の計算格子では r、ϕ の区間をそれぞれ均等に 7 分割しており、外側の格子では r、ϕ の区間をそれぞれ均等に 3 分割、15 分割している。内側の格子と外側の格子では格子の作り方が異なってもよいことを注意する。L の区間は 3 分割している。表 1 でも示したように、L の値と共に 2 種類の計算格子の扱う r の範囲が変化していく。

3.3 1 次光線と 2 次光線

ブラックホール周辺で光線が湾曲するとき、ひとつのポリゴン頂点から放射された光線が視点へ入射する経路は複

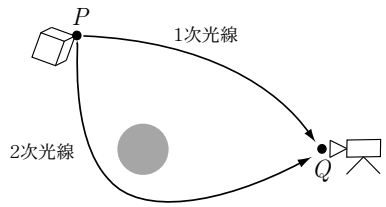

図 5 光線の種類とブラックホールの周回経路

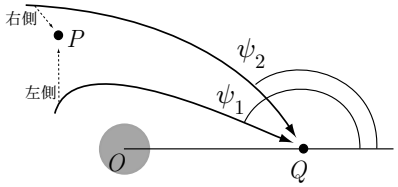

図 6 P から Q へ向かう光線の入射角度を 2 分法を求めるときの判定方法

数個ありうる（**図 5** 参照）。まず、頂点 P から放射された光線の中でブラックホールの周囲を 180° 以下の角度だけ周回して視点 Q に到達する光線が存在する。このような光線をここでは **1 次光線**（primary light ray）と呼ぼう。ここまでの議論は暗黙にこの 1 次光線を想定していた。

また、P から放射された光線の中でブラックホールの周囲を 180° より大きく 360° 以下だけ周回して Q に到達する光線が存在する[*5]。このような光線を **2 次光線**（secondary light ray）と呼ぼう。ブラックホール時空での CG では 1 次光線だけではなく、2 次光線も扱う必要がある。たとえば図 2 では、渦巻き銀河がブラックホールの右側にひとつ、左側にもうひとつ見えている。右側が 1 次光線による映像であり、左側が 2 次光線による映像である。

図 3 の入射角度 ψ について言えば、1 次光線では $0 \leq \psi \leq \pi$ となるが、2 次光線では $\pi < \psi \leq 2\pi$ となる。実は両者の違いはそれだけである。計算格子を Z バッファ法に用いる手法は両者で全く同じであり、同じ描画手順をそれぞれの光線について 2 回実行すればよい。また、1 次光線について内側/外側の 2 種類の計算格子が必要であったように、2 次光線についても 2 種類の計算格子が必要である。よって 4 種類の計算格子を用意せねばならない。

3.4 境界値問題の解法

式 (1) の微分方程式について、与えられた (r, ϕ, L) から ψ を効率的に求める計算アルゴリズムは知られていない。そこで本研究では 2 分法を使って ψ を探索する。

たとえば**図 6** では、視点 Q へ角度 ψ_1 で入射する光線が頂点 P を左に見ながら通過している。このことから、P を通過し、Q へ入射する光線の角度 ψ_P は ψ_1 よりも小さい。また、Q へ角度 ψ_2 で入射する光線は P を右に見ながら通過している。よって、ψ_P は ψ_2 よりも大きい。本研究では、2 分法に従って同様の判定を 30 回以上繰り返し、10^{-10} ラジアンよりも高い精度で ψ を計算している[*6]。

この計算を全ての格子点について繰り返すには膨大な計算時間を必要とする。計算時間は諸条件に依存するが、たとえば格子の数が 128^3 の場合に 4 枚全ての計算格子について投影データを集めるために 2.1 節の Mac Pro 4 台を用いて約 50 時間を要した。しかし事前準備にどれほど掛かろうとも、描画速度とは無関係である。

3.5 描画プログラムの実装

上述の方法に基づく実装の手順を整理しておく。

3.5.1 事前準備

(1) 図 3 の三つのパラメータ (r, ϕ, L) の数値の範囲とそれに対応する 3 次元計算格子の各次元の格子の数（範囲の分割数）を決める。なお、ここでは格子は均等に配置する（範囲を均等に分割する）と約束する。

(2) 内側/外側の全ての格子点について 1 次/2 次光線の $\psi(r, \phi, L)$、Z 値、波長の変化率の三つの値を計算し、GLSL シェーダの `vec4` 型データ（`float` の 4 個の並び）に格納する。なお、計算は倍精度で行うが、結果は単精度で保存する。

3.5.2 描画プログラムの実行

(1) 上記 `vec4` 型データを GLSL の 3 次元テクスチャとして GPU へ転送しておく。

(2) 通常の OpenGL/GLSL プログラムと同様にポリゴンを描画する。ただし、

(a) バーテックスシェーダは、各ポリゴン頂点の投影位置を上記 3 次元テクスチャを参照して計算し、後続のシェーダへ渡す（概要は 4.4 節に述べる）。

(b) フラグメントシェーダは、通常の処理（各描画点の色の計算など）を行う。

3.6 描画実験：均等な格子の場合

3.5 節の手順を従ってポイントスプライトを描画し（概要は 4.5.1 節で述べる）、透視投影が正しく実行されるか否か確認した。その結果、静止画では、ポイントスプライトは描画されるものの誤った位置に描画されることが目視で判断できた。動画では、本来は画面上を滑らかに移動すべきポイントスプライトが画面上で激しく揺れながら移動した。なお、紙面の都合上、関連画像の掲載は省略する。

表 2 は、約 30 万点の様々な (r, ϕ, L) について、

- 計算格子のデータから補間して求めた ψ と
- 3.4 節の方法で直接計算して求めた ψ

[*5] ブラックホールの周囲を 360° 以上周回する光線も理論上はわずかに存在すると考えられるが、その影響が皆無であるため、ここでは扱わない。

[*6] 微分方程式自体は、著者が 10 年以上前に開発した一般相対論用汎用レイトレーシング・プログラム（4 次の Runge-Kutta 法を使用し、光線の湾曲に応じて刻み値を動的に変える方法）[7] を改造して解いている。

表 2 均等な計算格子の線形補間誤差

格子の数	誤差の最大値		誤差の標準偏差	
	1 次光線		1 次光線	
$M_r \times M_\phi \times M_L$	内側	外側	内側	外側
$16 \times 16 \times 16$	96.576	89.312	36.610	32.641
$32 \times 32 \times 32$	91.263	88.623	32.721	30.121
$64 \times 64 \times 64$	84.526	87.246	28.804	27.279
$128 \times 128 \times 128$	81.782	85.617	14.230	24.178

角度の単位は度数。2 次光線については掲載を省略する。

の差の絶対値の統計データである。誤差の最大値、標準偏差は格子数の増加と共に減少するが、減少率は鈍く、表中の最大値は全て 80° 以上、標準偏差は 14° 以上である。これは極めて大きな誤差であり、画像の間違いが目視で判断できたことが十分に納得できる。

4. 線形補間誤差を最小にする計算格子

3.6 節の描画実験の ψ の誤差が大きい理由は、計算格子の格子点を均等に配置したためである（3.5.1 節参照）。補間誤差に伴う角度 ψ の誤差はポリゴン頂点が視点に近づくほど大きくなるから、本来はそれを考慮した配置にすべきである。そこでこの節では、線形補間誤差を最小にするような計算格子の配置法を検討する。

4.1 線形補間の一般式

まず、線形補間の方法を再確認する。

定義域 $[a,b]$ の上に関数 $f(x)$ が与えられたとする。この定義域を $M-1$ 分割し、M 個の代表点 $a = x_0 < x_1 < ... < x_{M-1} = b$ について、あらかじめ関数値 $y_i = f(x_i)$ を計算しておく。ただし、各代表点は、別に与える単調増加関数 $x = p(z)$（$z \in [0,1]$、$x \in [a,b]$、$p(0) = a$、$p(1) = b$）を用いて以下の式で定める。

$$x_i = p\left(\frac{i}{M-1}\right) \qquad i = 0, 1, ..., M-1$$

この p を**格子生成関数**（mesh generating function）と呼ぶ。

任意の点 $\hat{x} \in [a,b]$ が与えられたときに、関数値 $f(\hat{x})$ の近似値 $\hat{y}$ を以下の手順で求める。

(1) $z = p^{-1}(\hat{x})$ を計算する。p^{-1} は p の逆関数である。
(2) $(M-1)z = i + w$ を満たす整数値 i と実数値 w を求める。ただし $0 \le i \le M-2$、$0 \le w \le 1$ とする。
(3) $\hat{y} = (1-w) \cdot y_i + w \cdot y_{i+1}$ を計算する。

なお、GPU は上記 (2)、(3) をハードウェアで実行する。

本研究の技術的な課題は、ブラックホール時空内での透視投影計算において、真値 $f(\hat{x})$ と近似値 $\hat{y}$ の誤差 $\Delta(\hat{x}) = |f(\hat{x}) - \hat{y}|$ が十分に小さくなるような格子生成関数 p を見つけることである。

4.2 光線が湾曲しない場合の考察

最適な計算格子生成関数を探索するために、毎回、3.4 節の境界値問題を解くのは膨大すぎる計算量であり、事実上、不可能である。幸いなことに、視点の近くから放射され、すぐに視点に入射する光線は重力の影響を受けにくく、ほとんど湾曲することなく視点に到達する。そこで、格子生成関数を探す試みでは光線が湾曲しない場合を考察し、その結果を光線が湾曲する場合へ延用する。後に結果を示すようにこの方法は良好な結果を与える。

光線が湾曲しない場合、$\psi(r, \phi, L)$ は以下の式で表わすことができる（図 3 参照）。

$$\psi(r, \phi, L) = \arctan\left(\frac{r \sin\phi}{r\cos\phi - L}\right) \tag{2}$$

この式を用いて格子生成関数を評価していく。問題を単純化するため、L の格子生成関数についてはまだ考察を行わず、任意に与えられた L について、r と ϕ の最適な格子生成関数を求める。様々な予備調査によって格子生成関数として以下の関数形が良い結果を与えることが分かっている。ここに、A が関数形の偏りを定めるパラメータ、$[a,b]$ が定義域である。

$$F[A,a,b](z) = (b-a)\left(\frac{e^{Az}-1}{e^A-1}\right) + a$$

以下では専らこの関数を用いることとし、解くべき問題をパラメータ A の最適値の探索へと読み替える。

さて、内側の計算格子の r、ϕ に関する格子生成関数 $p^{\mathrm{i}}_r()$、$p^{\mathrm{i}}_\phi()$ は以下のように定義しよう。

$$p^{\mathrm{i}}_r(z) = F[A^{\mathrm{i}}_r, r_{min}, L](z)$$
$$p^{\mathrm{i}}_\phi(z) = F[A^{\mathrm{i}}_\phi, 0, \pi](z)$$

そしてパラメータ A^{i}_r、A^{i}_ϕ の最適値を以下の手順でしらみつぶし探索する。

(1) r と ϕ の 2 次元格子の数 $M_r \times M_\phi$ を定める。
(2) A^{i}_r、A^{i}_ϕ の値の組をひとつ選定し、2 次元の計算格子を決定し、式 (2) を用いて各格子点 (i,j) の $\psi_{i,j} = \psi(r_i, \phi_j, L)$ を計算し、配列に格納しておく。
(3) 対象領域内の (r,ϕ) の値の組をひとつ選定し、
  (a) $\psi_{i,j}$ の配列から双線形補間（bilinear interpolation）によって補間値 $\hat{\psi}$ を計算する。
  (b) 式 (2) から真値 ψ を直接計算する。
  (c) 誤差 $\Delta = |\hat{\psi} - \psi|$ を計算する。
(4) 上記 (3) を様々な (r,ϕ) について繰り返し実行し[*7]、その最大値を $\Delta_{max}(A^{\mathrm{i}}_r, A^{\mathrm{i}}_\phi)$ とする。
(5) 様々な A^{i}_r、A^{i}_ϕ について上記 (2)〜(4) を繰り返し実行し[*8]、$\Delta_{max}(A^{\mathrm{i}}_r, A^{\mathrm{i}}_\phi)$ の最小値を与える A^{i}_r、A^{i}_ϕ を求

[*7] 本研究の実験では約 30 万の点を用いた。ただし、全ての評価点と視点との距離は $0.01\sqrt{L}$ 以上とした。たとえば $L = 10000$ のとき、最小距離は 1 である。視点との距離を大きくすれば誤差は減少し、本節で求めた最適値も変わってくる。
[*8] 本研究の実験では A^{i}_r について、-30 から 30 まで 0.1 刻みで調べ、A^{i}_ϕ については 0 から 20 まで 0.1 刻みで調べた。

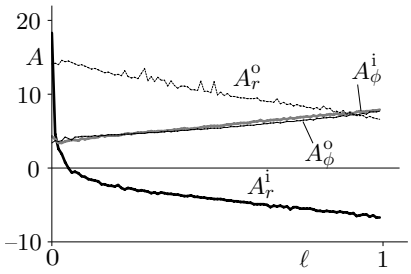

図 7 L と格子生成パラメータの関係

め、それを L に関する最適なパラメータの組とする。外側の格子についても同様である。生成関数を以下のように定義し、与えられた L に関する A_r^{o}、A_ϕ^{o} の最適値を上と同じ手順で求めればよい。

$$p_r^{\mathrm{o}}(z) = F[A_r^{\mathrm{o}}, L, r_{max}](z)$$
$$p_\phi^{\mathrm{o}}(z) = F[A_\phi^{\mathrm{o}}, 0, \pi](z)$$

上の実験を様々な L について行い、L とパラメータのグラフを求めると、たとえば $M_r \times M_\phi = 128^2$ のときに**図 7** のようになった。ここに、ℓ は L を以下のように対数化し、$[0,1]$ に正規化した値である。

$$\ell = \frac{\log L - \log L_{min}}{\log L_{max} - \log L_{min}}$$

最小二乗近似を適用することで、図 7 のグラフの各パラメータは以下の式で近似できた。

$$A_r^{\mathrm{i}} = -4.70 \cdot \ell - 2.12 + \frac{0.123}{l + 0.00605} \quad (3)$$
$$A_\phi^{\mathrm{i}} = 4.38 \cdot \ell + 3.56 + \frac{0.000186}{l + 0.000250} \quad (4)$$
$$A_r^{\mathrm{o}} = -7.95 \cdot \ell + 14.4 \quad (5)$$
$$A_\phi^{\mathrm{o}} = 3.72 \cdot \ell + 3.70 \quad (6)$$

次に、内側と外側の L の格子生成関数を以下のように定義する。

$$p_L^{\mathrm{i}}(z) = F[A_L^{\mathrm{i}}, L_{min}, L_{max}](z)$$
$$p_L^{\mathrm{o}}(z) = F[A_L^{\mathrm{o}}, L_{min}, L_{max}](z)$$

上述の手順と同様にして、A_L^{i}、A_L^{o} の最適な値を求めると、たとえば格子数 $M_L = 128$ のときに以下の通りであった。

$$A_L^{\mathrm{i}} = 15.3, \quad A_L^{\mathrm{o}} = 8.54 \quad (7)$$

4.3 線形補間誤差の評価

表 3 は、4.2 節で求めた格子生成関数を光線が湾曲する場合に適用し、線形補間誤差を評価した結果である。

まず、表 2 と比較して誤差が桁違いに小さいことに気づく。また、表中のどの種類の誤差も格子点数の増加と共にほぼ一定の割合で減少している[*9]。

格子点数が 128^3 の場合では、1 次、2 次光線とも外側格子の最大誤差は約 0.4° である。仮に画像の水平方向の視野角が 40° ならば、誤差 0.4° は 画像の水平方向の長さの約 1% に相当する。これは目視で辛うじて誤差に気づく程度であろう。同じ条件の標準偏差は約 0.02° であり、画像の水平方向の長さの約 0.05% に相当する。すなわち、ほとんどの評価点において誤差を目視で判別できないレベルである。よって、4.2 節で求めた計算格子は CG 画像を作る上では十分な精度を与えると考えられる。

[*9] 格子数が 8 倍増える毎に誤差は平均して約 0.6 倍減少している。

```
vec4 proj;
float r = length(vec3r);
float dotLR = dot(normalize(vec3L),normalize(vec3r));
float p = acos(dotLR);
float L = length(vec3L);
float el = (log(L)-log(Lmin))/(log(Lmax)-log(Lmin));

if(r <= L){
    float Ari = Ari0*el+Ari1+Ari2/(el+Ari3);
    float Api = Api0*el+Api1+Api2/(el+Api3);
    float zri = pinv(r,Ari,1+eps,L);
    float zpi = pinv(p,Api,0,PI);
    float zLi = pinv(L,ALi,1+eps2,Rmax2);

    proj = texture3D(inProjTex,vec3(zri,zpi,zLi));
} else {
    float Aro = Aro0*el+Aro1;
    float Apo = Apo0*el+Apo1;
    float zro = pinv(r,Aro,L,Rmax);
    float zpo = pinv(p,Apo,0,PI);
    float zLo = pinv(L,ALo,1+eps2,Rmax2);

    proj = texture3D(outProjTex,vec3(zro,zpo,zLo));
}
```

図 8 バーテックスシェーダの透視投影データの計算部

4.4 シェーダプログラムの実装

プログラム全体の実装は 3.5 節と同様である。計算格子の作り方は 3.5 節よりも複雑であるが、4.2 節の格子生成関数を用いて格子点を求め、投影データを準備すればよい。紙面の都合上、詳細は省略する。

投影データを線形補間で求めるバーテックスシェーダのプログラム片を**図 8** に示す。`vec3r`、`vec3L` にポリゴン頂点、視点の 3 次元座標が格納されているとしよう。このとき、図 8 のプログラムは以下を実行する。

(1) r、ϕ、L の値を求める。
(2) 正規化された L の値 ℓ を求める。
(3) もし $r \leq L$ ならば
  (a) パラメータ A_r^{i}、A_ϕ^{i} を求める。
  (b) ${p_r^{\mathrm{i}}}^{-1}(r)$、${p_\phi^{\mathrm{i}}}^{-1}(\phi)$、${p_L^{\mathrm{i}}}^{-1}(L)$ の値を求める。
  (c) 内側の計算格子の 3 次元テクスチャ（図 8 では `inProjTex`）から 3 次元線形補間によって透視投影データを取り出す。
(4) $L < r$ の場合も同様にして、外側の計算格子の 3 次元テクスチャ（図 8 では `outProjTex`）から 3 次元線形補間によって透視投影データを取り出す。

そして取り出された投影データを基に当該頂点の投影位置を計算すればよい。

表 3 最適な計算格子の線形補間誤差

格子の数	誤差の最大値				誤差の標準偏差			
	1 次光線		2 次光線		1 次光線		2 次光線	
$M_r \times M_\phi \times M_L$	内側	外側	内側	外側	内側	外側	内側	外側
$16 \times 16 \times 16$	2.881	7.949	1.688	5.941	0.520	1.485	0.306	1.878
$32 \times 32 \times 32$	0.837	1.226	0.940	1.679	0.145	0.214	0.210	0.227
$64 \times 64 \times 64$	0.392	0.751	0.470	0.874	0.034	0.065	0.023	0.071
$128 \times 128 \times 128$	0.201	0.401	0.229	0.432	0.009	0.019	0.009	0.020

角度の単位は度数。

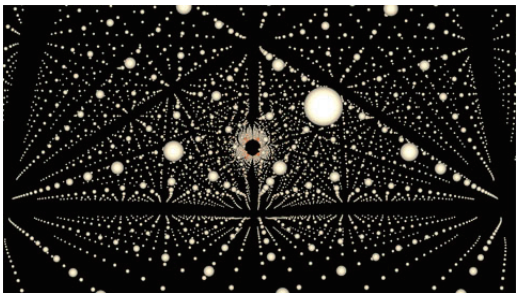

図 9 ブラックホール周辺に整列したポイント・スプライトの描画

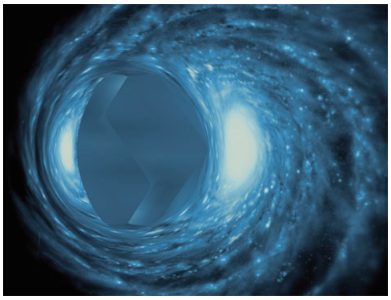

図 10 渦巻き銀河テクスチャを張った三角形ポリゴンの描画

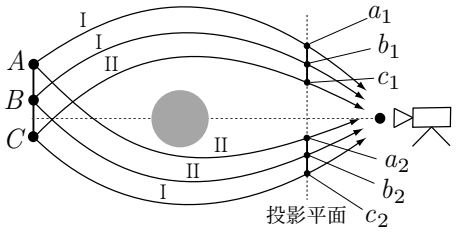

図 11 レイトレーシングによる線分の正しい描画

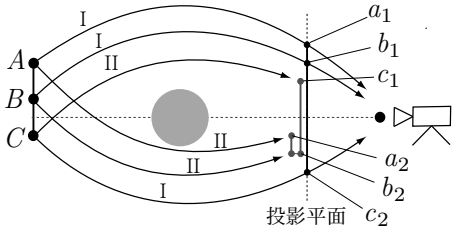

図 12 Z バッファ法による線分の間違った描画

4.5 描画実験：補間誤差最小な格子の場合

ここでは二つの描画実験について述べる。

4.5.1 ポイント・スプライトの描画

線形補間誤差の検証のために、多数の小球状のポイント・スプライトをブラックホールの回りに直方体をなすように整列させた場合の CG 画像を作成した（**図 9** 参照）。この画像中央にブラックホールが見え、その部分の球の配置が歪んでいる。ブラックホール周辺の球の色が赤みを帯びているのは、重力によって波長が伸びたためである。この画像は格子数が 128^3 の場合であるが、格子数が 32^3 以下の場合には球の列が乱れる（補間誤差が無視できなくなる）ことが目視で確認できた。

画像の平均描画時間は、画像解像度を 1280×720、ポイント数 4000、GPU に NVIDIA GeForce 8800 GT を用い、Mac Pro 上で約 0.34 ミリ秒/枚（約 2900 fps）であった。

4.5.2 三角板の描画

次に**図 10** は、三角形ポリゴン 2000 枚を使って構成した正方形に渦巻き銀河のテクスチャを張ったオブジェクトを、図 2 とほぼ同じ条件の下に描画した画像である。ただし、格子数は 128^3 である。銀河の外縁部の描像は、図 10 と図 2 で似ていることが分かる。しかし、図 10 では、本来ブラックホールの黒円が見える画像中央部に銀河の歪んだテクスチャが描画されており、明らかに間違った画像である。これは 3 節の描画手法の欠陥によるためである。この問題点の原因と対策を次節で議論する。

5. 三角板の描画

5.1 問題点

図 10 の画像の間違いの原因は**図 11** と**図 12** を用いて説明できる。ただし簡単のために、三角形ポリゴンではなく線分を用いる。

いま、ブラックホールを挟んで視点とはほぼ真反対の位置に二本の線分 AB と BC が配置されていると仮定する。これをレイトレーシング法で描画するならば、光線は図 11 のような軌跡を描くから、線分 AB は透視平面上に線分 a_1b_1 および a_2b_2 として描画され、BC は同じく b_1c_1、b_2c_2 として描画される。これが正しい描画である。

同じ被写体を Z バッファ法で描画するならば、線分は図 12 のように投影される。図 11 と図 12 の大きな違いは、図 11 の b_1c_1、b_2c_2 に相当する線分が図 12 には現れない点である。この理由は、図 11 の線分 b_1c_1、b_2c_2 の両端は一方の端点は 1 次光線、他方は 2 次光線であるが、3.3 節

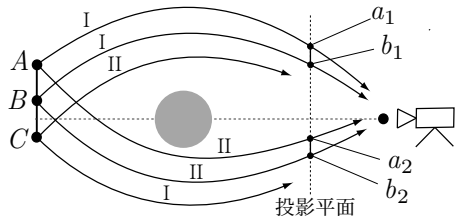

図 13 Z バッファ法による線分の描画（線分のフィルタリングを適用）

で述べた方法では次数の異なる光線を両端とする線分ポリゴンを描画できないからである。よって図 12 の BC は 1 次光線のみを両端とする線分 b_1c_2 と 2 次光線のみを両端とする線分 b_2c_1 へ投影される。しかしこれら線分は実際には描画すべきではない。特に b_1c_2 はブラックホールの前面に立ちふさがり、その特徴的な黒円を隠してしまうという点で問題が大きい。図 10 でもこれと同様の現象が起きているものと考えられる。また、b_1c_2 が最も深度の浅い描画像である[*10]ならば、本来描画されるべき a_2b_2 は描画されないという問題も生じる。

5.2 ジオメトリシェーダを用いた解決法

上記問題の最も単純な解決法は、描画すべきでない線分（図 12 の b_1c_2、b_2c_1）をフィルタリングし、描画しないことである。そうすれば**図 13** のような描画となる。正しい描画（図 11）に比べ、描画すべき線分が一部抜け落ちる不都合は生じるが、描画の間違いは軽微に抑えることができる。

描画すべきでない線分は、その性質上、投影平面上で異常に長い線分に投影されることが分かっている。三角形ポリゴンに言い換えるならば、描画すべきでないポリゴンは、投影平面上で異常に面積の大きいポリゴンに投影される。

そこで、バーテックスシェーダを以下のように改造し、新たにジオメトリシェーダをユーザ定義する。

5.2.1 バーテックスシェーダの改造

各頂点について、ブラックホールが存在 する 場合の投影位置（この計算法は 4.4 節で述べた通り）とブラックホールが存在 しない、通常の透視投影法での投影位置を計算し、両方の計算結果をジオメトリシェーダに渡す。

5.2.2 ジオメトリシェーダの改造

(1) ジオメトリシェーダに入力された 3 頂点から、ブラックホールが存在 する 場合に投影される三角形の面積を計算し、S とする。

(2) 同上の 3 頂点から、ブラックホールが存在 しない 場合に投影される三角形の面積を計算し、S' とする。

(3) 面積比 S/S' がある定数値（本研究では 9.0）よりも大きいならばその 3 頂点を後続のシェーダに渡さない。

[*10] 深度値の大小関係は状況によって様々であるため、これ以外の場合もありうる。

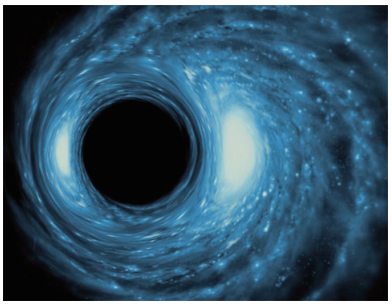

図 14 渦巻き銀河テクスチャを張った三角形ポリゴンの描画（手法改良後）

さもなくば通常通り、3 頂点を後続のシェーダに渡し、描画する。

5.3 描画実験

上記の改造によって**図 14** の CG 画像を生成できた。詳細に見れば図 2 の画像とは異なる点もあるが、差はほとんど目立たない。この画像の格子数は 128^3 であるが、4.5.1 節と同様に格子数の減少と共に画像の劣化が大きくなる。

なお、図 14 の平均描画時間は、三角形ポリゴン 20000 枚を用い、その他の条件は 4.5.1 節と同じとして約 9.7 ミリ秒/枚（約 103 fps）であった。

6. 今後の課題

まず、様々な 3D モデリングデータを表示できるように CG プログラムを改良中である。同時に、描画プログラム（特にシェーダ・プログラム）の高速化を進めている。

また、今回はブラックホール外部の CG を検討したが、原理的には同様の手法をブラックホール内部へ応用できるはずである。カメラがブラックホール内部へ落ち込むときの CG 映像はまだほとんど知られていない。理由のひとつはブラックホール内部のレイトレーシングの膨大な計算時間なのだが、今回の手法でそれを克服できるかもしれない。

参考文献

[1] D. Kobras, et al. : General Relativistic Image-Based Rendering, The Visual Computer, Vol. 18, No. 4 (2002) pp. 250-258.
[2] Turner Whitted : An Improved Illumination Model for Shaded Display, Comm. of the ACM, Vol.23, No.6, (1980) p343–349.
[3] D. Weiskopf, et al. : Explanatory and Illustrative Visualization of Special and General Relativity, IEEE Trans. on Visualication and Computer Graphics, VOL. 12, NO. 4 (2006) pp.522-534.
[4] 内山龍雄：一般相対性理論、裳華房 (1978).
[5] 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.
[6] 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.
[7] 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.