高速運動する視点と被写体に関する特殊相対論的コンピュータグラフィックス

山下 義行[1,a)]

概要：特殊相対論を 3D CG に適用する試みは 30 年前から行われているが、最近では GPU の性能向上と共にリアルタイム 3D CG アニメーションが可能になってきている。本研究では GPU をフル活用し、3D CG の標準的な描画手法である Phong の反射モデルと補間手法を相対論的に厳密に取り扱う。具体的には視点と被写体が亜高速で運動する場合の 3D CG を行う。相対論的な画像の歪みにはテッセレーションを適用し、正確な形状描画を行う。被写体が亜高速加速度運動を行う場合の取り扱いは既存の研究にはない。iOS 端末上でリアルタイム描画が可能である。

Special Relativistic Computer Graphics for Fast Moving Viewpoint and Objects

1. はじめに

著者は 3 次元コンピュータグラフィックス（3D CG）の一般相対論的拡張、特にブラックホールの 3D CG を研究している [1], [2], [3], [4]。その準備研究のひとつとして本稿では 3D CG の特殊相対論的拡張について報告する。

特殊相対論では座標系が互いに亜光速で相対運動しているときの座標変換、いわゆるローレンツ変換を定式化している。ローレンツ変換では視点のそばを亜光速で通り過ぎる物体の像は歪（ゆが）み、被写体の色は赤方/青方へ偏移して見える。これを 3D CG 画像 として描画する試みは 30 年以上前から種々行われている [5], [6], [7]。しかし形状の歪みや色の変化が注目される中、通常の 3D CG で適用される拡散反射や鏡面反射を精密に相対論的に拡張し、計算する研究はこれまでほとんど行われていない。また、被写体が加速度運動する場合の CG 描画を対象とする研究もほとんど知られていない。

本研究は、これらの特殊相対論的に拡張される 3D CG の未解決問題に挑戦する。これは以下の 3 点で既存研究と異なる。

(1) ほとんど全ての関連研究では、描画手法として光線追跡法を用いるが、GPU の最大性能を引き出すため、本研究ではラスタライズ法を用いる。ラスタライズ法も最大の問題は被写体の像の歪みを正確に描画できないこととされてきた [5]。本研究ではテッセレーションを用いてこれを解決する。

(2) 加速度運動する被写体を取り扱う。文献 [6] ではこれを未解決問題としている。本研究ではゆっくり加速する場合の近似式を構築し、部分的に解決する。

(3) 通常の 3D CG では標準的な技法である Phong の反射モデル、Phong の補間法、Gouraud の補間 [10] を特殊相対論的に精密に拡張し、実装する。これは既存研究ではほとんど行われていない。

本研究のプログラムは、Apple/iPad Pro の上にプログラミング言語 Swift[8]、シェーダ言語 Metal[9] で実装する。

2. 準備

ここではこの論文に必要な特殊相対論の内容をまとめる。

2.1 ローレンツ変換

平坦な時空における 4 次元座標系を $\boldsymbol{x} = (t, x, y, z)$ と表す。ここに t が時間座標、x、y、z が 3 次元直交空間座標である。この $\boldsymbol{x}$ 上の 4 次元ベクトル $\vec{\boldsymbol{a}} = (a_t, \vec{a}) = (a_t, a_x, a_y, a_z)$ の長さが

$$|\vec{\boldsymbol{a}}|^2 = a_t^2 - |\vec{a}|^2 = a_t^2 - (a_x^2 + a_y^2 + a_z^2) \qquad (1)$$

[1] 佐賀大学工学系研究科
Faculty of Science and Engineering, Saga University
[a)] yaman@is.saga-u.ac.jp

© 1959 Information Processing Society of Japan

で定義できる性質を持つとき、$\vec{a}$ を特に **4 元ベクトル**（four vector）と呼ぶ。相対論で扱う光線は 4 元ベクトルであり、時空内の任意の 2 点間の相対位置も 4 元ベクトルである。この論文では、4 元ベクトル (a_t, a_x, a_y, a_z) を矢印付き太文字 $\vec{\boldsymbol{a}}$ で表し、その空間成分 (a_x, a_y, a_z) を矢印付き平文字 $\vec{a}$ で表す。

さて、同じ時空を張る別の 4 次元座標系 $\boldsymbol{X} = (T, X, Y, Z)$ が座標系 $\boldsymbol{x}$ の上を速度 $\vec{\beta} = (\beta_x, \beta_y, \beta_z)$ で移動していると仮定する。ただし、この論文では全ての速度は光速を 1 とする大きさに正規化されていると約束する。よって $|\vec{\beta}| < 1$ が成り立つ。特殊相対論では、座標系 $\boldsymbol{x}$ の上の任意の 4 元ベクトル $\vec{\boldsymbol{a}}$ は以下の行列変換によって $\boldsymbol{X}$ の上の 4 元ベクトル $\vec{\boldsymbol{A}}$ へ変換できる。これをローレンツ変換と呼ぶ。

$$\vec{\boldsymbol{A}} = \boldsymbol{L}(\vec{\beta})\vec{\boldsymbol{a}}$$

$\boldsymbol{L}(\vec{\beta})$ は以下の 4×4 の行列である。

$$\boldsymbol{L}(\vec{\beta}) = \begin{pmatrix} \gamma & \beta_x\gamma & \beta_y\gamma & \beta_z\gamma \\ \beta_x\gamma & 1 + {n_x}^2\eta & n_x n_y \eta & n_x n_z \eta \\ \beta_y\gamma & n_x n_y \eta & 1 + {n_y}^2\eta & n_y n_z \eta \\ \beta_z\gamma & n_x n_z \eta & n_y n_z \eta & 1 + {n_z}^2\eta \end{pmatrix}$$

ここに $\gamma = 1/\sqrt{1 - |\vec{\beta}|^2}$、$\eta = \gamma - 1$、$n_i = \beta_i/|\vec{\beta}|$ $(i = x, y, z)$。逆変換は以下の通りである。

$$\vec{\boldsymbol{a}} = \boldsymbol{L}^{-1}(\vec{\beta})\vec{\boldsymbol{A}} = \boldsymbol{L}(-\vec{\beta})\vec{\boldsymbol{A}}$$

2.2 光線ベクトル

特殊相対論では光線ベクトル $\vec{\boldsymbol{a}} = (a_t, \vec{a})$ の大きさは 0 である。よって式 (1) から以下が成り立つ。

$$a_t = \pm|\vec{a}| \tag{2}$$

$\vec{\boldsymbol{a}}$ が未来へ向かうベクトルならば $a_t > 0$ であり、過去へ向かうならば $a_t < 0$ である。

通常の 3D CG の意味で 3 次元光線ベクトル（運動量ベクトル）$\vec{b}$ が座標系 $\boldsymbol{x}$ の上に定義されていると仮定する。式 (2) からその光線ベクトルの時間成分 b_t を計算し、光線ベクトル $\vec{\boldsymbol{b}} = (b_t, \vec{b})$ を構成する。次にこのベクトルをローレンツ変換で座標系 $\boldsymbol{X}$ の上の光線ベクトル

$$\vec{\boldsymbol{B}} = \boldsymbol{L}(\vec{\beta})\vec{\boldsymbol{b}} = (B_T, \vec{B})$$

へ変換しよう。その空間成分 $\vec{B}$ は $\boldsymbol{X}$ の上の光線の運動量ベクトルである。一般に $\vec{b}$ と $\vec{B}$ は異なる方向のベクトルであり、これが被写体が二つの座標系 $\boldsymbol{x}$ と $\boldsymbol{X}$ で異なって見えることの原因となる。時間成分はその光線のエネルギーに相当するから座標変換前後の時間成分の比 $\eta = B_T/b_t = |\vec{B}|/|\vec{b}|$ は光線の見かけのエネルギーの変化を表すこととなる。この変化が光のドップラー効果として観測される。

2.3 3 種類の座標系

本論文では以下の 3 種類の座標系を用いる。

視点座標系 $\boldsymbol{X}_C$ ... 視点（カメラ）がその原点に静止する座標系

被写体座標系 $\boldsymbol{X}_P$ ... 被写体（ポリゴン）を構成する頂点群がその原点周辺に静止する座標系

光源座標系 $\boldsymbol{x}$ ... 光源（平行光線）が静止する座標系。

3 節では、$\boldsymbol{X}_P$ が $\boldsymbol{x}$ に固定され、X_C が自由に運動する場合を扱う。4 節では、$\boldsymbol{X}_C$ が $\boldsymbol{x}$ に固定され、X_P が自由に運動する場合を扱う。最終的には三つの座標系が独立に運動する場合も取り扱うことができるが、紙面の都合上、この論文では述べない。光源座標系は通常の 3D CG のワールド座標系に相当するものと考えてよい。

3. 高速運動する視点

この節では、視点座標系 $\boldsymbol{X}_C$ の原点（＝視点の位置）が光源座標系 $\boldsymbol{x}$ の上を $\vec{f}(t) = (f_x(t), f_y(t), f_z(t))$ の軌跡で運動すると仮定する。ここに t は $\boldsymbol{x}$ の時間座標である。また、視点は $\boldsymbol{X}_C$ の上で $\vec{F}(t)$ の方向を見ており、視点の上方向は $\vec{U}(t)$ と仮定する。全ての被写体は $\boldsymbol{x}$ の上に静止していると仮定する。

3.1 相対論的透視投影変換

光源座標系の時刻 $t = \hat{t}$ において CG 画像を作成しよう。

その瞬間、視点座標系 $\boldsymbol{X}_C$ の原点は光源座標系 $\boldsymbol{x}$ の上を速度 $d\vec{f}(\hat{t})/dt$ で運動している。そこでローレンツ変換行列 $\boldsymbol{L}(d\vec{f}(\hat{t})/dt)$ を用意する。また視点の視線ベクトル $\vec{F}(\hat{t})$、視点の上方向ベクトル $\vec{U}(\hat{t})$ から $\boldsymbol{X}_C$ 上の視野変換行列 $\boldsymbol{M}_{\mathrm{View}}(\vec{F}(\hat{t}), \vec{U}(\hat{t}))$ を通常の 3D CG の方法で用意する。透視投影変換行列 $\boldsymbol{M}_{\mathrm{Proj}}$ も視野角などを指定して通常の 3D CG の通りに用意する。

N 個のポリゴン頂点で構成された被写体を考える。その被写体座標系 $\boldsymbol{X}_P$ の原点が $\boldsymbol{x}$ 上の $\vec{p}$ の位置にあり、被写体の i 番目の頂点が $\boldsymbol{X}_P$ 上の $\vec{P}_i$ の位置にあると仮定すれば、その頂点は $\boldsymbol{x}$ 上の $\vec{p} + \vec{P}_i$ の位置にある。

時刻 $t = \hat{t}$ の視点は $\vec{f}(\hat{t})$ の位置にいる。よって視点から見た頂点の相対的な位置は

$$\vec{q}_i = \vec{p} + \vec{P}_i - \vec{f}(\hat{t}) \tag{3}$$

である。視点は光線によってその頂点を観測するから、式 (2) によって、3 次元ベクトル $\vec{q}_i$ から以下の 4 元ベクトルが構成できる。

$$\vec{\boldsymbol{q}}_i = (-|\vec{q}_i|, \vec{q}_i). \tag{4}$$

なお、3D CG の観点では視点から被写体頂点へ入る光線は過去へ向かうベクトルであるから $\vec{\boldsymbol{q}}_i < 0$ と設定する。

光線ベクトル $\vec{\boldsymbol{q}}_i$ は以下のローレンツ変換で視点座標系へ変換できる。

© 1959 Information Processing Society of Japan

$$\vec{\boldsymbol{Q}}_i = \boldsymbol{L}(d\vec{f}(\hat{t})/dt)\vec{\boldsymbol{q}}_i \tag{5}$$

さて、$\vec{\boldsymbol{Q}}_i$ の空間成分 $\vec{Q}_i$ が視点座標系 $\boldsymbol{X}_C$ での被写体頂点の位置である。$\vec{\boldsymbol{Q}}_i$ と $\vec{\boldsymbol{q}}_i$ の時間成分の比 $\eta_i = |\vec{Q}_i|/|\vec{q}_i|$ が座標系間での光線のエネルギー変化である。

次に、空間成分 $\vec{Q}_i$ から 4 次元同次座標系（homogeneous coordinates）の位置ベクトル $\vec{\boldsymbol{Q}}'_i = (\vec{Q}_{i,j}, 1)$ を作り、これに視野変換、透視投影変換を順に適用する。

$$\vec{\boldsymbol{R}}_i = \boldsymbol{M}_{\mathrm{Proj}} \cdot \boldsymbol{M}_{\mathrm{View}}(\vec{F}(\hat{t}), \vec{U}(\hat{t})) \cdot \vec{\boldsymbol{Q}}'_i \tag{6}$$

ここで、$\vec{\boldsymbol{R}}_i = (R_X, R_Y, R_Z, R_W)$ と置くとき、

$$\vec{S}_i = \left(\frac{R_X}{R_W}, \frac{R_Y}{R_W}\right) \tag{7}$$

が CG 画像上の投影位置であり、

$$D_{i,j} = \frac{R_Z}{R_W} \tag{8}$$

が深さ値である。

3.2 描画像の歪みとテッセレーション

通常の 3D CG のラスタライズ法で三角形を描画するにはその 3 頂点を透視投影変換し、投影された 3 頂点を線分で結んだ三角形をラスタライズして描画すればよい。しかし上に述べたローレンツ変換 $\boldsymbol{L}(d\vec{f}(\hat{t})/dt)$ を含む透視投影変換には線分の描画像は湾曲するため、上の通常のラスタライズ法では不十分である。

線分の湾曲を陽な数式で求めることは原理的には可能であるが、任意方向のローレンツ変換での任意の線分の湾曲の式はきわめて複雑で長大であるため、計算プログラムに実装することは現実的ではない。この論文では湾曲を動的テッセレーションで解決する。

テッセレーションは、1 枚の三角形を複数の小さな三角形に分割し、その小さな三角形をそれぞれラスラタイズする方法である。描画像の歪みの程度に応じて分割数を制御すれば高速で正確な描画が可能である。最近の GPU はそれをハードウェアで実行できるため、高速描画も期待できる。GPU によるテッセレーションでは、分割は三角形の各辺の分割数（以下、エッジテッセレーショ係数）および内部の分割数（以下、内部テッセレーション係数）を指定する。この四つの係数を決めれば、GPU のハードウェアテッセレーターが三角形を高速に自動分割する。本研究ではエッジテッセレーション係数を以下の方法で決定する。

今、被写体座標系 $\boldsymbol{X}_P$ の上の位置 $\vec{P}$ を中点とし、$\vec{P}^+ = \vec{P} + \Delta\vec{P}$、$\vec{P}^- = \vec{P} - \Delta\vec{P}$ を両端点とする線分について検討する（図 1 参照）。この 3 点について 3.1 節で述べたローレンツ変換、視野変換、投影変換を行い、投影面上の 2 次元画素位置 $\vec{S}$、$\vec{S}^+$、$\vec{S}^-$ をそれぞれ求めたと仮定する。このとき

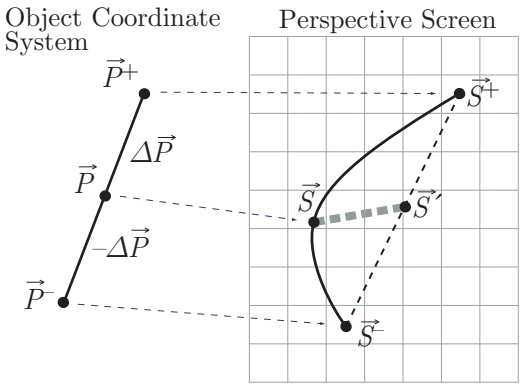

図 1 一辺の湾曲の計算

$$E(\vec{P}, \Delta\vec{P}) = \left|\vec{S} - \vec{S}'\right| \tag{9}$$

を線分を直線近似した場合の $\vec{P}$ の描画位置の誤差と定義する。ここに $\vec{S}' = (\vec{S}^+ + \vec{S}^-)/2$ とする。この誤差を $|\Delta\vec{P}|$ で多項式近似するとき、誤差は $|\Delta\vec{P}|$ の 1 次項を持たず、2 次以上の項からなることが容易に証明できる（証明は省略する）。よって、$|\Delta\vec{P}|$ が十分小さいときには、3 次項以上を無視し、誤差は近似的に $|\Delta\vec{P}|$ の二乗に比例するとみなす。よって、ε ($<< 1$) について近似的に以下が成り立つ。

$$E(\vec{P}, \varepsilon\Delta\vec{P}) = \varepsilon^2 E(\vec{P}, \Delta\vec{P})$$

次に ε を $1/n$ で置換すると以下の通りである。

$$E(\vec{P}, \frac{\Delta\vec{P}}{n}) = \frac{E(\vec{P}, \Delta\vec{P})}{n^2}$$

高品質な画像生成には誤差は 1 画素以下、すなわち

$$E(\vec{P}, \frac{\Delta\vec{P}}{n}) \leq 1$$

が成り立つべきである。よって

$$n \geq \sqrt{E(\vec{P}, \Delta\vec{P})}$$

ここで、n は長さ $|\Delta\vec{P}|$ の分割数であるから、線分長 $|\vec{P}^+ - \vec{P}^-|$ の分割数はその 2 倍に相当する。よって、$\vec{P}^+$ と $\vec{P}^-$ を両端点とする線分を分割するためのエッジテッセレーション係数 T は

$$T = \lceil 2n \rceil = \left\lceil 2\sqrt{E(\vec{P}, \Delta\vec{P})} \right\rceil = \left\lceil 2\sqrt{\left|\vec{S} - \vec{S}'\right|} \right\rceil \tag{10}$$

と設定すればよい。ここに $\lceil\ \rceil$ は天井関数である。

実際のテッセレーションの実装では、図 2 のように、被写体座標系 $\boldsymbol{X}_P$ 上の三角形の頂点および辺の中点の計 6 点について投影平面上の位置を求め、$\left|\vec{S} - \vec{S}'\right|$ を計算し、上記の方法で各辺のエッジテッセレーション係数を決定する。内部テッセレーション係数はこの論文では 3 辺のエッジテッセレーション係数の平均値とする。

正三角形が視点の正面を $\beta = 0.00, 0.33, 0.66, 0.99$ のそ

© 1959 Information Processing Society of Japan

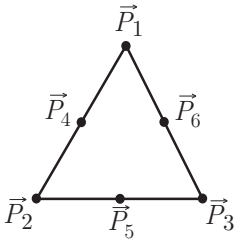

図 2 三角形ポリゴンのパッチ構造

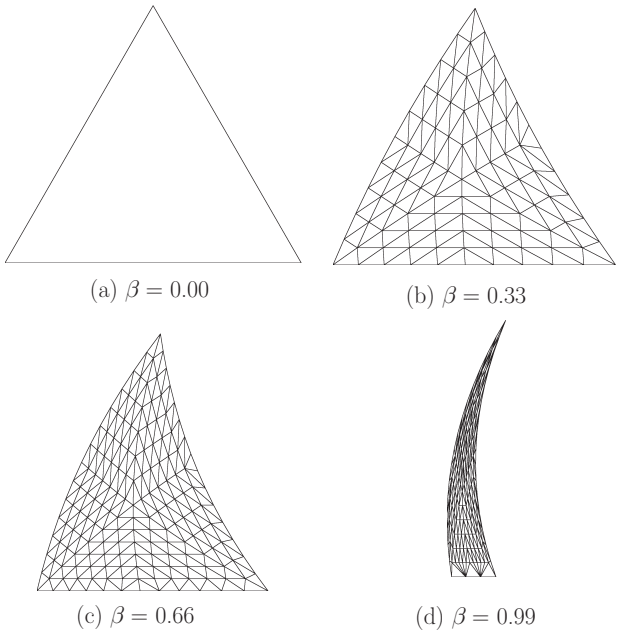

図 3 視点の高速運動と正三角形の歪み

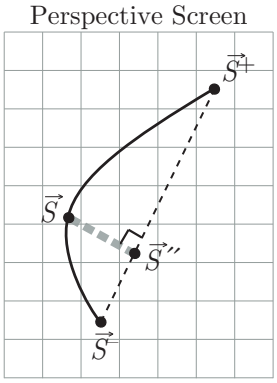

図 4 横滑りを無視した場合

れぞれの速度で横切るときのテッセレーションの様子を実際に求めたものが図 3 である。速度が上がるほどに三角形が湾曲するから、分割数が増えていることが分かる。

ところで、もし三角形の外形の歪みにのみ注目するならば、誤差の計算に図 1 の $\vec{S}$ と $\vec{S}'$ の距離ではなく、図 4 の $\vec{S}$ と $\vec{S}''$ の距離 を用い、テッセレーション係数を以下の式で計算してもよいかもしれない。

$$T = \left\lceil 2\sqrt{\left|\vec{S} - \vec{S}''\right|} \right\rceil \tag{11}$$

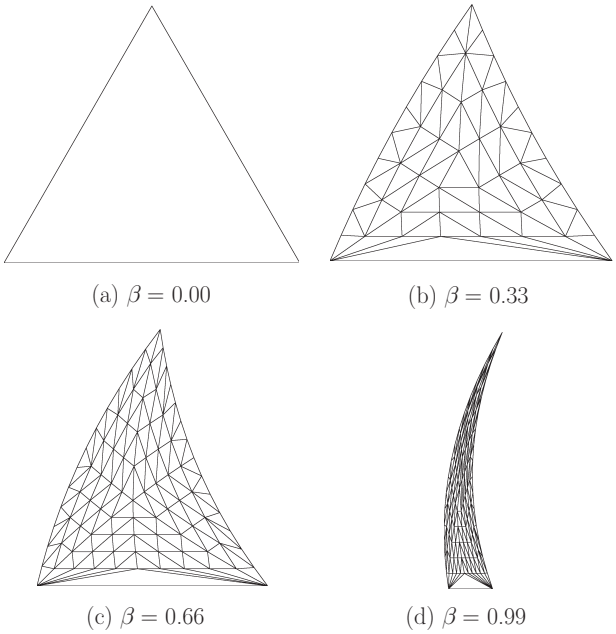

図 5 視点の高速運動と正三角形の歪み（横滑りを無視）

これによってテッセレーション係数を小さな値に抑えることができる。図 5 はこの計算方法を用いて図 3 と同じ条件で描画した例である。図 3 よりも分割数が減っていることが分かる。

3.3 陰影計算とドップラー効果

この節の仮定では光源は光源座標系に静止しており、被写体も光源座標系に静止しているから、被写体表面での Phong 反射光（拡散反射光、環境反射光、鏡面反射光）は通常の 3D CG と同様に計算できる。通常の 3D CG との違いは、計算された Phong 反射光が視点に入射するときに光のドップラー効果を受けることである。この論文では、光線は黒体輻射スペクトルに従うものとし、ドップラー効果を色温度の変化に還元して計算する。光のドップラー効果を黒体輻射スペクトルで計算する方法は文献 [2] をそのまま用いる。

3.4 実装と描画実験

本研究では描画プログラムを Apple iOS をプラットフォームとしてホストプログラムには Swift[8] を用い、シェーダプログラムには Metal[9] を用いて開発している。図 6 は、GPU シェーダのブロックダイアグラムである。以下は各シェーダの概要である。なお、ローレンツ変換行列 $\boldsymbol{L}(d\vec{f}(\hat{t})/dt)$、視野変換行列 $\boldsymbol{M}_{\mathrm{View}}(\vec{F}(\hat{t}), \vec{U}(\hat{t}))$、投影変換行列 $\boldsymbol{M}_{\mathrm{Proj}}$ はあらかじめホスト側で計算し、GPU へ転送する。

計算シェーダ (1) ... 個々の頂点について式 (3)〜式 (7) を適用し、投影位置を求める。

計算シェーダ (2) ... 各三角形の上の 6 点（図 2 参照）の投影位置に式 (10) または式 (11) を適用し、テッセレー

© 1959 Information Processing Society of Japan

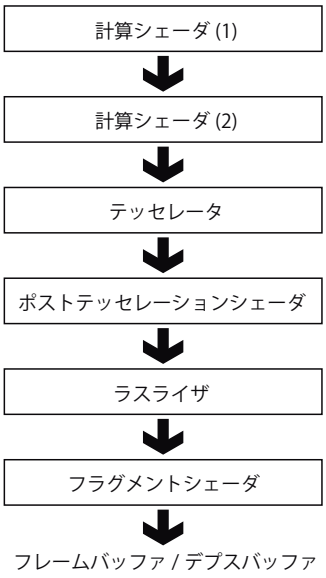

図 6 描画のためのシェーダ構成図

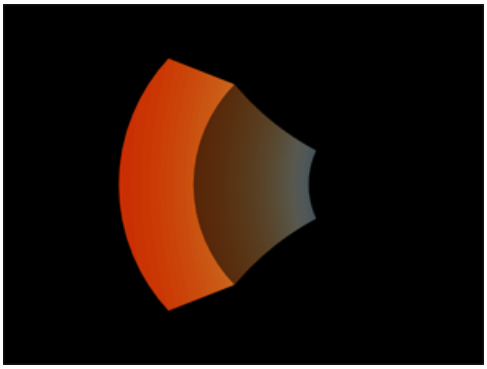

図 7 立方体の描画例

ション係数を求める。

テッセレータ ... テッセレーション係数に従ってより各パッチを小さな三角形に分割する。

ポストテッセレーションシェーダ ... 分割で得られた各三角形の各頂点について投影面上の位置 $\vec{S}$、深さ値 D、光源座標系 $\boldsymbol{x}$ 上の位置 $\vec{p}$、法線ベクトル $\vec{n}$ 、ドップラー効果による光線の波長の変化率 η を求める。

ラスタライザ ... 三角形の各内部画素について上記 D、$\vec{p}$、$\vec{n}$ 、η を線形補間する。

フラグメントシェーダ ... $\boldsymbol{x}$ 上の光線ベクトル $\vec{L}$ 、視点の位置 $\vec{C}$、線形補間された $\vec{p}$、$\vec{n}$ を用いて Phong 反射光の強度を計算する。光源光線の色温度、被写体表面の色温度を η だけ偏移させて、この画素の RGB 値を求める。線形補間された D をこの画素の深さ値とする。

図 7 は、視点が巨大な立方体の側を光速の 0.99 倍の速度で通過する際の描画例である。形状の湾曲と光のドップラー効果が観察できる。図 9 は、点が巨大な球と格子模様の側を光速の 0.99 倍の速度で通過する際の描画例である。球は形状が歪まないことが知られており、図 9 の球は歪んでいない。図 10 は 10 両編成の列車の描画例である。ローレンツ短縮が観察される。

表 1 はテッセレーションによる描画速度の低下を FPS 値

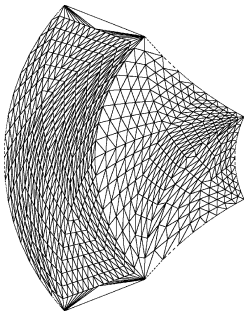

図 8 図 7 の立方体の分割の様子

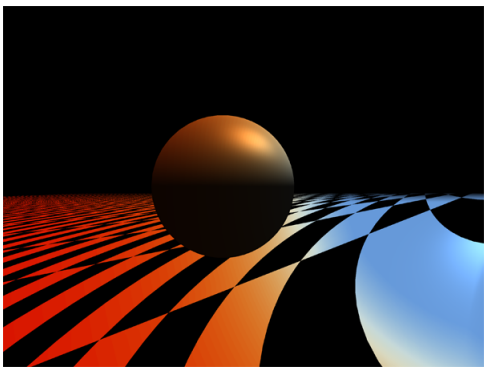

図 9 球と格子の描画例

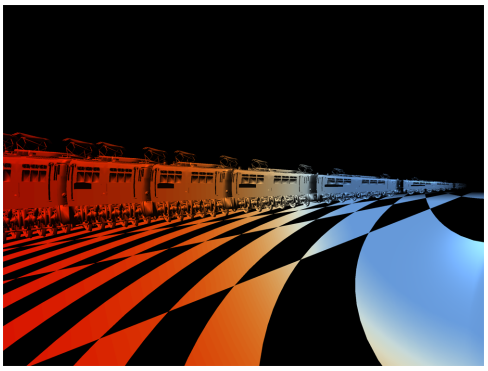

図 10 列車の描画例

表 1 描画速度

被写体		テッセレーション		
			横滑り	
(ポリゴン数)	速度	無し	無し	有り
立方体	0.00	306	280	270
(12)	0.99	434	339	334
球	0.00	378	281	286
(1,520)	0.99	458	301	305
格子模様	0.00	348	136	141
(40,000)	0.99	331	126	136
列車	0.00	248	59	59
(477,640)	0.99	166	56	60

単位は Frames Per Second

© 1959 Information Processing Society of Japan

で計測した結果である。テッセレーションを行わない場合が一番高速であるのは当然である。テッセレーションを行う場合、ポリゴン数が多い場合に性能低下が大きくなる傾向にある。横滑りを行うことで分割数が小さくなるため、性能がやや向上することが分かる。

4. 高速運動する被写体

この節では前節とは逆に、視点座標系 $\boldsymbol{X}_C$ の原点（＝視点の位置）が光源座標系 $\boldsymbol{x}$ に静止し、被写体座標系 $\boldsymbol{X}_P$（$1 \leq i \leq M$）の原点が $\boldsymbol{x}$ の上を 自由な軌跡で運動する場合の 3D CG 画像の生成を行う。ただし、後に述べるように、被写体が加速度運動を行う場合には十分にゆっくりとした加速度運動であるという制限を課す。

基本的な描画方法は前節と同じであり、被写体の各頂点についてテッセレーション、投影変換、Phong シェーディングを行う。視点（＝カメラ）は静止しているから、視点についてローレンツ変換は不要であるが、この節では被写体が運動しているため、被写体頂点についてローレンツ変換を必要となる。

4.1 被写体のローレンツ短縮

今、被写体座標系 $\boldsymbol{X}_P$ の原点が光源座標系 $\boldsymbol{x}$ の上を等速直線運動 $\vec{\beta}t + \vec{p}$ の軌跡で動いていると仮定する。ここに $\vec{\beta}$、$\vec{p}$ は定数、t は $\boldsymbol{x}$ の時間座標とする。このとき、$\boldsymbol{X}_P$ から $\boldsymbol{x}$ への座標変換はローレンツ変換と並行移動を組み合わせたポアンカレ変換として以下のように表現できる。

$$\vec{\boldsymbol{x}} = \boldsymbol{L}(\vec{\beta})\vec{\boldsymbol{X}}_P + (0, \vec{p})$$

前節同様に、被写体は $\boldsymbol{X}_P$ の原点の周りの N 個の頂点から構成されており、i 番目（$1 \leq i \leq N$）の頂点の位置が $\boldsymbol{X}_P$ 上の空間位置 $\vec{P}_i$ であると仮定する。よってこの頂点の $\boldsymbol{X}_P$ 上の軌跡は $(T, \vec{P}_i)$ である。ここに T は $\boldsymbol{X}_P$ の時間座標とする。この軌跡に上記ポアンカレ変換を適用し、T を消去すると、$\boldsymbol{x}$ 上でのこの頂点の軌跡 $\vec{h}_i(t)$ が以下のように求められる。

$$\vec{h}_i(t) = \vec{\beta}t + \vec{p} + \vec{P}_i - \left(\sqrt{1 - |\vec{\beta}|^2} - 1\right)\left(\vec{n} \cdot \vec{P}_i\right)\vec{n} \quad (12)$$

ここに $\vec{n} = \vec{\beta}/|\vec{\beta}|$ である。上の式中の

$$-\left(\sqrt{1 - |\vec{\beta}|^2} - 1\right)\left(\vec{n} \cdot \vec{P}_i\right)\vec{n}$$

は視点から見た被写体頂点の進行方向へのローレンツ短縮を表している。

上の式は被写体が等速直線運動を行なっている場合であるが、加速が十分に小さい場合にはこの式を近似的に拡張適用できるだろう。つまり、被写体座標系 $\boldsymbol{X}_P$ の原点が光源座標系 $\boldsymbol{x}$ を $\vec{g}(t)$ の軌跡で運動しているとき、$\boldsymbol{X}_P$ 上の $\vec{P}_i$ に静止する被写体頂点の $\boldsymbol{x}$ 上での軌跡を以下の通りと考える。

$$\vec{h}'_i(t) = \vec{g}(t) + \vec{P}_i - \left(\sqrt{1 - |\vec{\beta}(t)|^2} - 1\right)\left(\vec{n}(t) \cdot \vec{P}_i\right)\vec{n}(t)$$

ここに $\vec{\beta}(t) = d\vec{g}(t)/dt$、$\vec{n}(t) = \vec{\beta}(t)/|\vec{\beta}(t)|$ である。

さらに被写体のゆっくりとした旋回のために下の拡張を行う。ここに $\boldsymbol{R}(t)$ は被写体座標系原点を中心とする空間回転行列とする。

$$\begin{aligned}\vec{h}''_i(t) \;=\; & \vec{g}(t) + \boldsymbol{R}(t)\vec{P}_i \\ & - \left(\sqrt{1 - |\vec{\beta}(t)|^2} - 1\right)(\vec{n}(t) \cdot \boldsymbol{R}(t)\vec{P}_i)\vec{n}(t)\end{aligned} \quad (13)$$

上式は近似式であることに注意する。

4.2 被写体の描画位置の決定

式 (13) のように、被写体座標系 $\boldsymbol{X}_P$ の頂点 $\vec{P}_i$ が $\boldsymbol{x}$ 上の軌跡 $\vec{h}''_i(t)$ で与えられていると仮定する。

光源座標系の時刻 $t = \hat{t}$ の CG 画像を作ることを考える。視点は $\boldsymbol{x}$ 上の $\vec{f}$ に静止しているものとする。時刻 $t < \hat{t}$ に位置 $\vec{h}''_i(t)$ の被写体頂点から発射された光線がちょうど視点に届くには

$$\hat{t} - t = \left|\vec{f} - \vec{h}''_i(t)\right|$$

が成り立たねばならない。この式を満たす t を求めたい。上式を変形すると

$$t = \hat{t} - \left|\vec{f} - \vec{h}''_i(t)\right|$$

となる。そこで、数列 (t_k)（$k = 0, 1, , ...$）を導入し、上の式を漸化式に変形する。

$$t_{k+1} = \hat{t} - \left|\vec{f} - \vec{h}''_i(t_k)\right| \quad (k \geq 0) \quad (14)$$

この漸化式を以下が成り立つまで繰り返す。

$$|t_{k+1} - t_k| < \delta \quad (15)$$

ここに δ は十分小さな正定数である。

命題 1　式 (14) は条件：

$$\left|\frac{d\vec{h}''_i(t)}{dt}\right| < 1$$

が成り立つならば、すなわち頂点の速度が光速よりも小さいならば、最悪 $\max\left|d\vec{h}''_i(t)/dt\right|$ の速度で一時収束する（紙面の都合上、証明は省略する）。

言うまでもなく収束回数は初期値 t_0 の与え方に強く依存する。最も単純な初期値は視点の時刻 $\hat{t}$ を用いる方法である。この研究では CG アニメーションを想定しているため、直近のコマの計算履歴を元に収束値を予測する方法を

© 1959 Information Processing Society of Japan

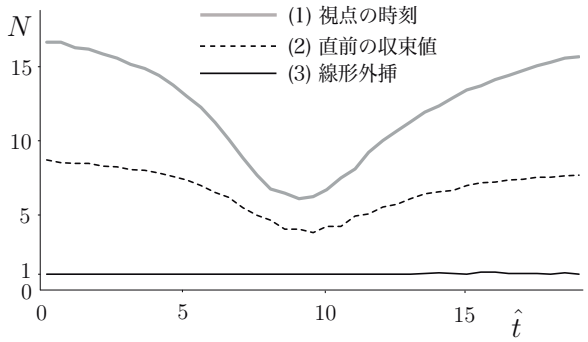

図 11 光速の 0.5 倍で目前を通過する立方体の位置の収束計算回数

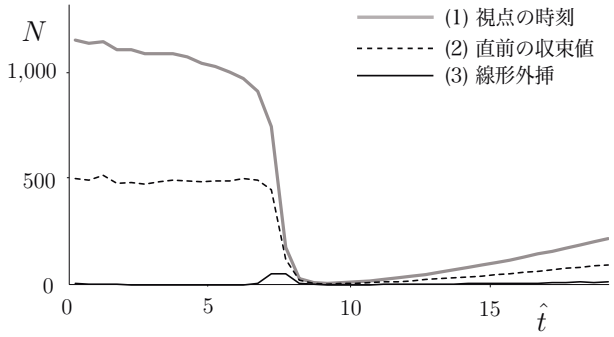

図 12 光速の 0.99 倍で目前を通過する立方体の位置の収束計算回数

加え、以下の 3 種類の方法を用い、式 (14) の収束性を実験する。

(1) 視点位置の時刻 $\hat{t}$ を用いる。
(2) 直近のフレームの収束値を用いる。
(3) 直近の 2 フレームから線形外挿する。

表 11 は、立方体が光速の 0.5 倍で視点の前を通過する場合の前後 20 秒間について上記の 3 方法で収束回数を測定したグラフである。同様に、表 12 は、光速の 0.99 倍のグラフである。いずれの実験でも (3) の直近の 2 フレームから線形外挿する方法が優れており、以下、この方法で描画を行う。

4.3 被写体座標系での陰影計算

Phong 反射光 [10] の計算は被写体座標系 $\boldsymbol{X}_P$ の上で通常の 3D CG の方法で計算する。反射光の計算には

(1) 法線ベクトル
(2) 入射光ベクトル
(3) 視線ベクトル

が必要であるが、(1) は元々 $\boldsymbol{X}_P$ の上で定義されている。(2)、(3) は光源座標系 $\boldsymbol{x}$ （= 視点座標系 $\boldsymbol{X}_C$）で定義されているから、反射光の計算では (2)、(3) をローレンツ変換せねばならない。この変換は画素毎に必要であるため、その計算量は膨大であるが、後に示すように最近の GPU を用いるならばリアルタイムで可能なところになってきている。Phong の補間法ではなく、（特殊相対論的に拡張された）Gouraud の補間法を用いれば計算量の削減が期待できる。それについては後に実験結果のみを示す。

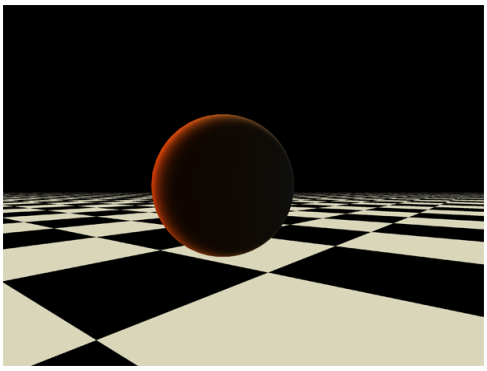

図 13 球の描画例相対速度が光速の 0.99 倍のときの球の描画（床および光源は静止）

被写体座標系で計算された反射光には被写体の運動に伴うドップラー効果を加味する。

上の計算は Phong 反射光を相対論的に厳密に扱うものであるが、既存研究ではほとんど行われていない。

4.4 実装と描画実験

描画プログラムは前節のそれを改造して作ることができる。GPU シェーダのブロックダイアグラムも前節と同じである。以下は各シェーダの概要である。

計算シェーダ (1) ... 以下の計算を行う。
- (1) 個々の頂点について式 (14) を $\delta = 10^{-4}$ と設定し、最大 1000 回繰り返して、収束時刻 t_k を求める。図 11、図 12 のグラフで見たように、ほとんどの場合、数回の計算で収束する。
- (2) 3.4 節の計算シェーダ (1) と同様に、t_k における投影位置を求める。

計算シェーダ (2) ... 3.4 節と同じ
テッセレータ ... 3.4 節と同じ
ポストテッセレーションシェーダ ... 3.4 節と同じ
ラスタライザ ... 3.4 節と同じ
フラグメントシェーダ ... 以下の計算を行う。
- (1) 光源座標系から被写体座標系へのローレンツ変換行列 $\boldsymbol{L}(\vec{\beta}(t_k))$ を求める。被写体が加速度運動している場合には $\boldsymbol{L}(\vec{\beta}(t_k))$ は一般には画素点毎に異なる。
- (2) $\boldsymbol{L}(\vec{\beta}(t_k))$ を用いて、光源座標系上の光線ベクトル、視点ベクトルを被写体座標系上へ変換する。
- (3) 光線ベクトル、視線ベクトル、ポリゴンの法線ベクトルから Phong 反射光を計算する。
- (4) 光源光線の色温度、被写体表面の色温度を η だけ偏移させて、強度と合わせて画素の RGB 値を求める。
- (5) 線形補間された D を深さ値として設定する。

図 13 は、球が光速の 0.99 倍で視点のそばを通過する場

© 1959 Information Processing Society of Japan

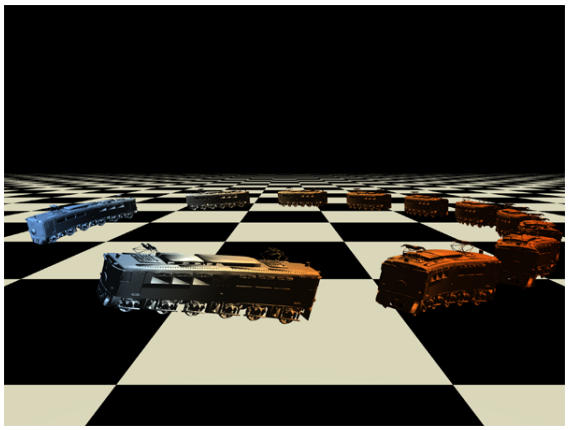

図 14 光速の 0.9 倍で旋回する列車

表 2 描画速度

被写体	運動の	補間手法	
（ポリゴン数）	種類	Phong	Gouraud
立方体	直線	282	357
（12）	回転	363	353
球	直線	288	335
（1,520）	回転	296	335
列車	直線	15	17
（477,640）	回転	34	37

単位は Frames Per Second

合の描画像である。視点が球のそばを通過する場合の図 9 と比較してほしい。球の形状が一致しているのは、球と視点の相対運動が両画像で同じだからである。しかし、光源、球、視点の関係は両者で異なるため、球の陰影付けが異なる。

図 14 は、列車が光速の 0.9 倍で回転運動する場合の描画例である。ドップラー効果によって視点に近く車両は青く輝き、遠のく車両は赤黒く描画されている。式 (13) は、ゆっくりとした加速、旋回に近似的に適用すべきであると述べた。この回転運動がそれに当てはまるかの判断は難しい。ひとつの近似解として理解すべきであろう。

表 2 に描画速度をまとめた。直線運動は、図 13 のように視点のそばを被写体が光速の 0.99 倍で通過する場合である。回転運動は、図 14 のように視点の前方で被写体が光速の 0.9 倍で旋回運動する場合である。Phong 補間を用いた場合よりも Gouraud 補間の場合がやや高速であり、これは十分納得できる。表 2 の直線運動、Phong 補間の描画速度を表 1 と比較した場合、列車の描画では被写体が運動する方が遅いが、立方体、球ではほぼ同じ描画速度である。ポリゴン数が多い列車の場合は、GPU のメモリアクセス速度が低下するなどの問題があるのかもしれない。

今後、さらに実験を重ね、検証を進める。

5. 今後の課題

今後、視点、被写体がそれぞれ独立して自由に運動する場合の 3D CG について研究を進める。これには本稿の 3 節と 4 節の内容を合併すればよい。

次に、本研究の特殊相対論 CG の技法を一般相対論 CG、特にブラックホールの CG[2] へ拡張適用する予定である。

参考文献

[1] 山下義行. 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, 火の国情報シンポジウム 2013 CD-ROM. 情報処理学会九州支部, 2013.
[2] Yoshiyuki Yamashita. Implementing a rasterization framework for a black hole space-time. *Journal of Information Processing*, Vol. 24, No. 4, pp. 1–10, July 2016.
[3] 山下義行. タブレット端末上での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, 電気・情報関係学会九州支部第 69 回連合大会 CD-ROM. 2016.
[4] 山下義行. 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：亜光速運動する視点と被写体の描画, 電気・情報関係学会九州支部第 70 回連合大会 CD-ROM. 2017.
[5] F. W. Hehl, R. A. Puntigam, and Hanns Ruder, editors. *Relativity and Scientific Computing: Computer Algebra, Numerics, Visualization*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.
[6] Thomas Muller, Sebastian Grottel, and Daniel Weiskopf. Special relativistic visualization by local ray tracing. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 16, pp. 1243–1250, May 2010.
[7] Zachary W. Sherin, Ryan Cheua, and Philip Tan. Visualizing relativity: The openrelativity project. *American Journal of Physics*, Vol. 84, pp. 369–374, May 2016.
[8] Apple Inc. *Swift home page*. https://swift.org.
[9] Apple Inc. *Metal Shading Language Guide*. https//developer.apple.com.
[10] Graham Sellers, Richard S. Wright, and Nicholas Haemel. *OpenGL SuperBible: Comprehensive Tutorial and Reference*. Addison-Wesley Professional, 6th edition, 2013.

© 1959 Information Processing Society of Japan