2007年度　卒　業　論　文

リアルタイム3DCGによる
漫画的スクリーントーン表現の研究

指導教員：渡辺 大地講師

メディア学部 ゲームサイエンスプロジェクト
学籍番号　M0104477
渡邉　貴純

2007 年度　　卒　業　論　文　概　要

論文題目

リアルタイム 3DCG による
漫画的スクリーントーン表現の研究

メディア学部 **学籍番号：M0104477**	**氏名**	渡邉　貴純	**指導教員**	渡辺 大地講師

キーワード	CG、NPR、シェーダ、スクリーントーン、アニメーション、漫画

近年、多くの映画やゲームで 3DCG のデフォルメ表現であるトゥーンレンダリングが使われるようになってきている。これは陰影の制御や輪郭線の強調を行う日本のアニメ的な表現を目的とするノンフォトリアリスティックレンダリング (NPR) の 1 つである。一方で日本の漫画は色を点や模様で擬似的に表す、スクリーントーンを使った日本の漫画的な表現を実現する手法はあまり提案されていない。そこで点や模様での表現に使うスクリーントーンに着目し、スクリーントーンを使った漫画的な表現を行う手法を提案する。アニメーションはカラー媒体を前提に作られることが殆どであり、色に関する制限はない。一方漫画は白黒媒体が前提である場合が多く、色を使うことが出来ないので代わりに点や模様で色を擬似的に表現している。この点や模様を描画するのに使うのがスクリーントーンである。先行研究として NPR で手書き風の表現を行うものは幾つかある。点で物を表現する点表現は幾つか研究が行われているが点の配置の偏りを解消する為に計算に時間が掛かってしまう。処理が非常に重い為、カメラやモデルの移動に対して十分な速度を保つ事が出来ない。ゲームで使用するには処理したものを予めテクスチャや動画として用意する必要がある。そこで本研究ではスクリーントーンをテクスチャで扱う事で描画結果に貼り付ける形を取り漫画表現を行い、リアルタイム性と有用性の検証を行った。まずモデルをマルチパスレンダリングで 3 つのパスに描画する。第 1 のパスを使い輪郭線を抽出する。第 2 のパスを使い使うスクリーントーンの選別を行う。第 3 のパスを使いスクリーントーンの濃度を決める。3 つの結果を合成し、漫画表現を行う。DirectX と HLSL で実装し、レンダリング結果を検証した。

目 次

第 1 章

はじめに

1.1 研究の背景と目的

3DCG ではノンフォトリアリスティックレンダリング (NPR)[1] という物がある。NPR とは写実的では無い表現を行う手法であり、中でもゲームやアニメーションではトゥーンレンダリング [2][3] が使われる事が多い。トゥーンレンダリングは色の階調表現や輪郭線の強調を行って日本のアニメ的な表現を行うものである。一方で日本の漫画は色を点や模様で擬似的に表す、スクリーントーンを使った日本の漫画的な表現を実現する手法はあまり提案されていない。

アニメーションと漫画の表現について違いのうち、大きな物のひとつに色の扱いがある。アニメーションはカラー媒体を前提に作られることが殆どであり、色に関する制限は無い。一方漫画は白黒媒体を前提にしている場合が多く、色に関する制限が有るので代わりに点や模様で色を擬似的に表現している。本研究ではこの部分に着目し、スクリーントーンを用いた漫画表現を提案する。スクリーントーンとはエセルテ社の商標である。白と黒の点の集まりで構成されるシール状の物であり、点の濃さや密度、パターンによって様々な種類がある。カラーで出力が行えない場合にこれを用いる事で擬似的に色を表現し、漫画では多用される。

図 1.1 はスクリーントーンを用いた表現である。左は濃度の違う点を等間隔に打つ事で整った模様を表現している。右は点の打ち方を場所によって変える事で線のように見せたり、拡散させる事で明暗を表現している。このように点の打ち方

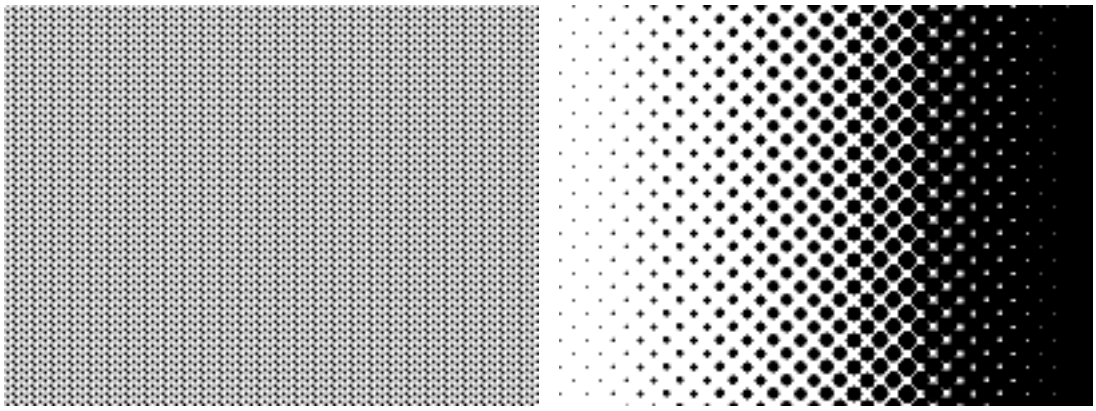

図 1.1: スクリーントーン

で様々な表現を行っている。原稿に貼り付けて使う物であり、ただ貼るだけではなく消しゴムを掛けて薄くしたりカッターで削って模様を欠けさせる手法も取られる。日本の漫画では頻繁にスクリーントーンを使用し、色を表現している。同じスクリーントーンでも消しゴムやカッターで薄くしたりして強弱を付け微妙な色の差を生み出している。海外の漫画ではスクリーントーンの使用頻度は低い。色の表現には用いるが影は線の強弱で表現する事が多く、場合によっては色も線で表現する。

NPR において、漫画表現の様に絵画的な表現を行う手法は幾つか有る。今回の目的の様に点を使って模様を描画するのは点表現 [4][5][6] が向いている。その中でも確率分布に基づく点配置法 [7][8][9] があり、これは確率分布に従って点を配置する事で点の密度を制御し階調の変化を表現する点表現の手法である。しかし計算量が多く処理が重い為、カメラやモデルの移動に対して十分な速度を保つ事が出来ずゲームに使用するには処理したものを予めテクスチャや動画として用意する必要がある。テクスチャをスクリーントーンの様に使う事で絵画風表現を行う手法 [10] はテクスチャのマッピングを手動で設定する必要があり、カメラの位置と注視点が変化するゲームでは使用する事が出来ない。点の濃度も含めて階調の変化を行う場合は唯の濃淡表現に近づいてしまい NPR らしさを失ってしまう。

本研究ではスクリーントーンを使い、色を擬似的に表現する事で漫画的表現を

行い有用性の検証を行う。アニメーションだけでなくゲームでも使える事を視野に入れ、リアルタイム性を保持する事を目標とする。スクリーントーンはテクスチャとして扱い、元の 3DCG の色に応じて貼り付けるスクリーントーンを動的に変更させる。これにより連続した絵の中で急にスクリーントーンが切り替わる事を防ぎ統一感を持たせる。また点の濃度を変化させる事によってスクリーントーンの加工を再現する。

1.2　本論文の構成

本論分は全 5 章で構成する。2 章ではスクリーントーン表現の実装について具体的な手順を示す。3 章ではモデルを用いて有効性の検証を行う。4 章では問題点を示す。最後に 5 章でまとめを述べる。

第 2 章

スクリーントーン表現の実装

スクリーントーンを使った漫画表現の実装において以下の処理を行った。

1. 輪郭線の抽出
2. スクリーントーンの選別
3. スクリーントーンの濃さの調整
4. 結果の合成

漫画らしさを出すために輪郭線を抽出 [11] する。次に使うスクリーントーンを選別し、どの程度の濃さで使用するのかを決める。最後に各々の結果を合成して完成となる。これらの処理を効率良く行うためにマルチパスレンダリング [12][13][14] を行う。マルチパスレンダリングとは一度に複数の出力を行う事である。出力された結果を元に各々加工を行い、最終的に合成を行う。図 2.1 はマルチパスレンダリングを示す。これを使用する事により描画回数を削減する事が出来、処理速度の大幅な短縮が可能となる。今回は 3 つのパスを利用する。

図 2.2 は使用するスクリーントーンである。使用するスクリーントーンはあらかじめ画像として用意する。RGB に対応した 3 種類と白に対応した 1 種類の合計 4 種類を使用する。画面に等倍で表示する為、画像の解像度は画面の解像度と同一又はそれ以上が必要となる。

本章で用いる記号は次の意味である。

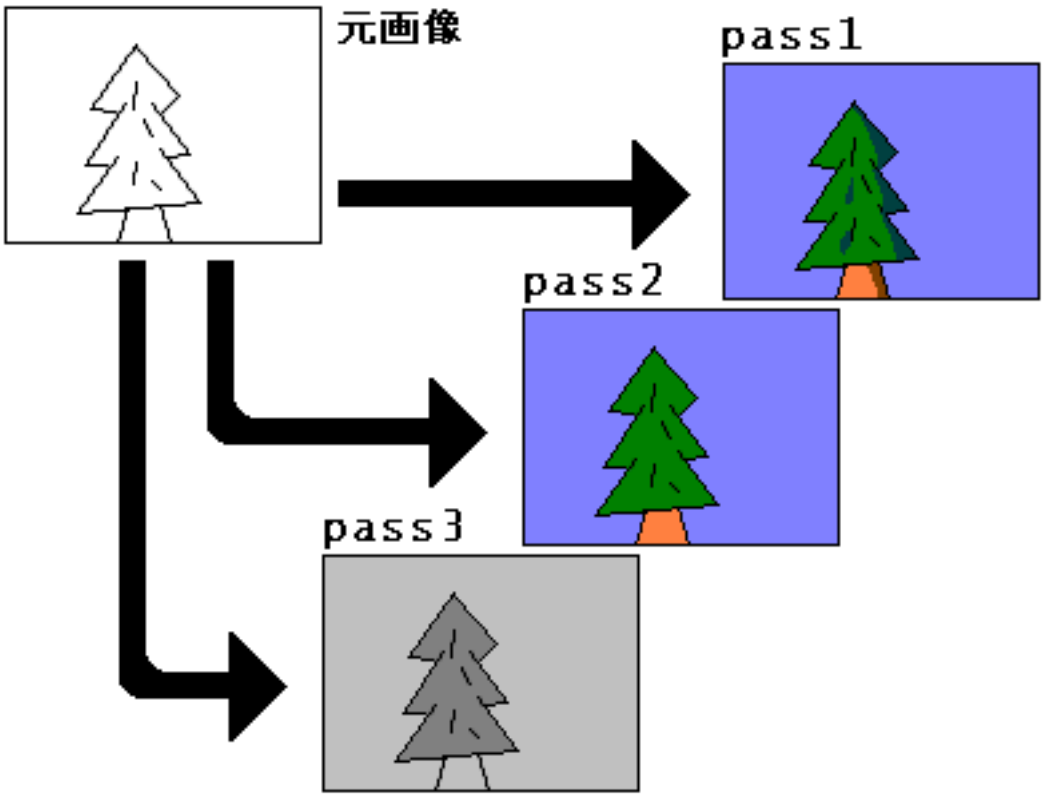

図 2.1: マルチパスレンダリング

- $\hat{\mathbf{n}}$ は n の正規化ベクトルを示す。
- $\otimes$ は成分乗算を示す。
- $\mathbf{a}\cdot\mathbf{b}$ はベクトル $\mathbf{a}$ と $\mathbf{b}$ の内積を示す。
- 色成分は RGB α の 4 つとし α は透明度を示す。値は 0 から 1 の実数とする。

以下、個々の処理について説明する。

2.1 輪郭線の抽出

最初に Lambert の余弦定理 [15][16] を使ったモデルの描画を行う。Lambert の余弦定理とはのように光の向きの逆ベクトルと面の法線を使い光の反射率を計算する方法である。図 2.3 は Lambert の余弦定理を示す。

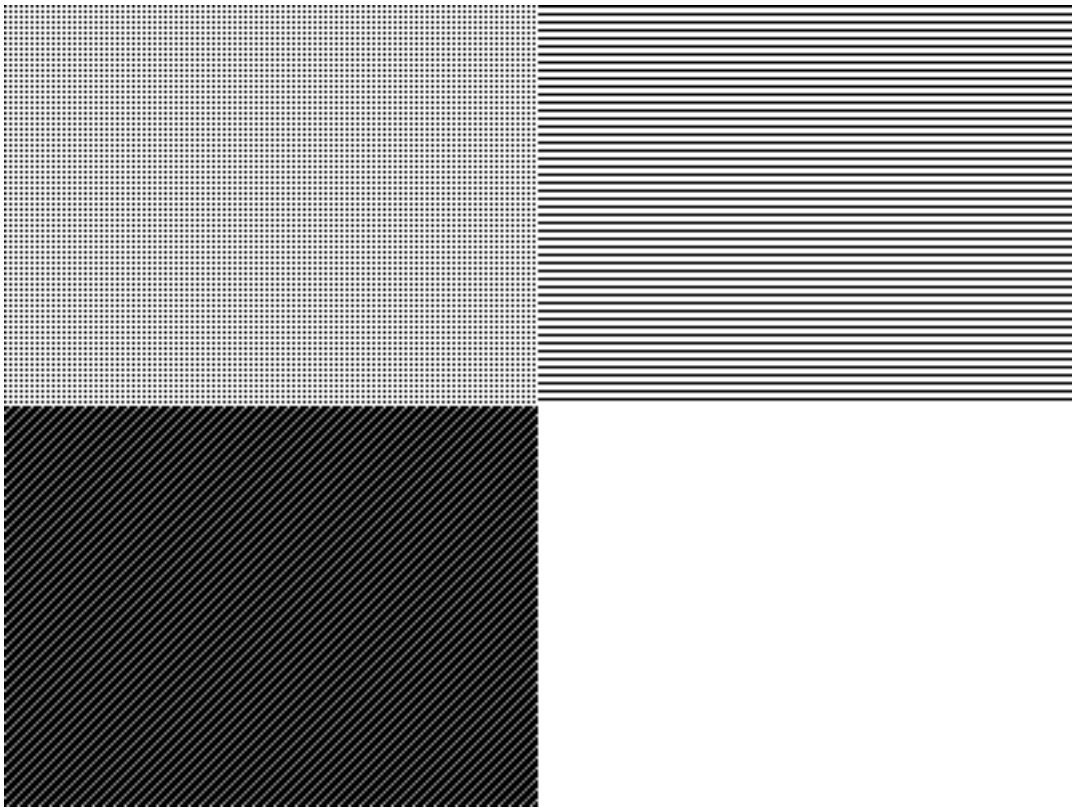

図 2.2: 使用するスクリーントーン

出力する色を E、表面の法線ベクトルを $\mathbf{n}$ 、光の逆ベクトルを $\mathbf{l}$ 、物体の色成分を m、光の色成分を s とした時の式は次のようになる。

$$E = (\hat{\mathbf{n}} \cdot \hat{\mathbf{l}})(m \otimes s) \tag{2.1}$$

光の逆ベクトルと面の法線の内積が 1 に近ければ明るく、0 以下に近くなれば暗くなる。図 2.4 は Lambert の余弦定理を使ってモデルを描画した物である。

次に色差エッジ抽出を使い輪郭線の抽出 [17] を行う。色差エッジ抽出とは、色と周りの色を比べて色の差が一定以上だった場合に輪郭線と定める手法である。ピクセルの位置 P_1 とその周囲のピクセル P_2、P_3、P_4 に対して、色を $\mathrm{E}(P_x)$ と輪郭線の色 $F(P_1)$ と輪郭線の強度 G とすると式は次のようになる。

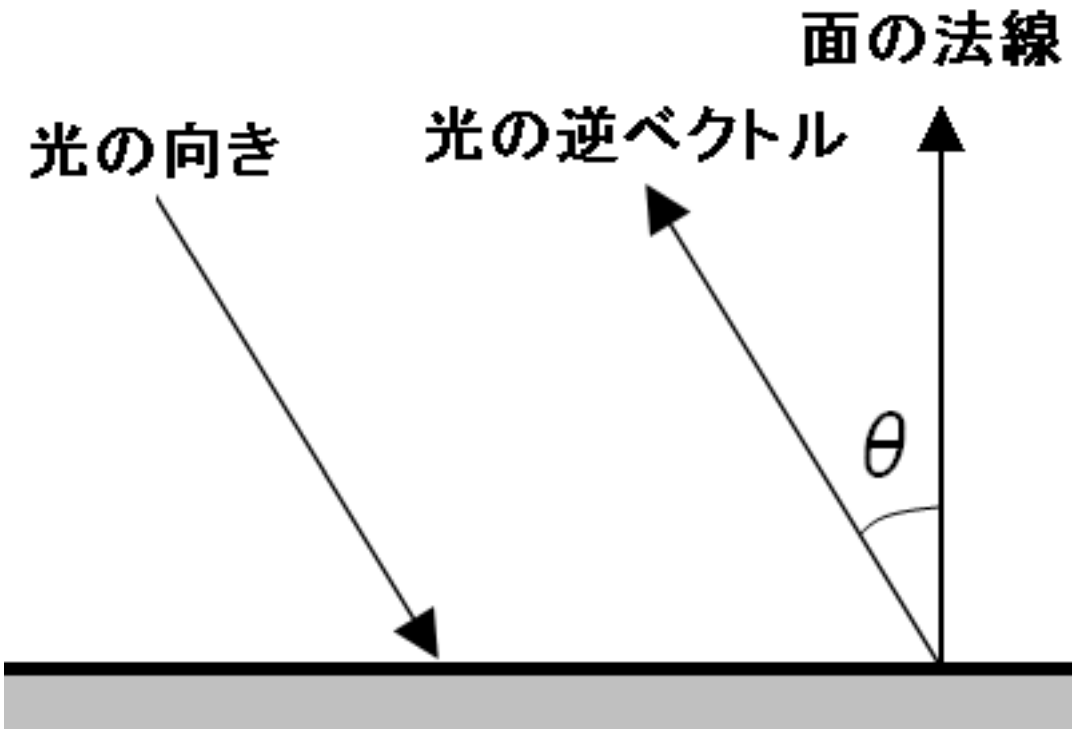

図 2.3: Lambert の余弦定理

$$F(P_1) = G\left(\left|E(P_1) - E(P_4)\right| + \left|E(P_2) - E(P_3)\right|\right) \tag{2.2}$$

P_1 を基点として、周囲のピクセルと斜めに比較する交差比較を行なう。P_2 は P_1 に対して x 軸方向に+1、P_3 は P_1 に対して y 軸方向に+1、P_4 は P_1 に対して x 軸 y 軸にそれぞれ+1 した位置である。色差エッジ抽出には他にも基点を中心に上下左右の色と比べる手法があるが、それは計算に使う点がそれに比べて交差比較は計算量を減らすことが出来る。G を変更すると輪郭線の抽出の強さを変更出来る。G を高くすると輪郭線は強く抽出され、低くすると弱く抽出される。図 2.5 は輪郭線を抽出した物である。

輪郭線の抽出は出来たが、色の差を取っただけなので輪郭線に色が付いている。それを白黒の濃淡に変換する処理を行なう。修正後の輪郭線の色を H とした場合の式は次のようになる。

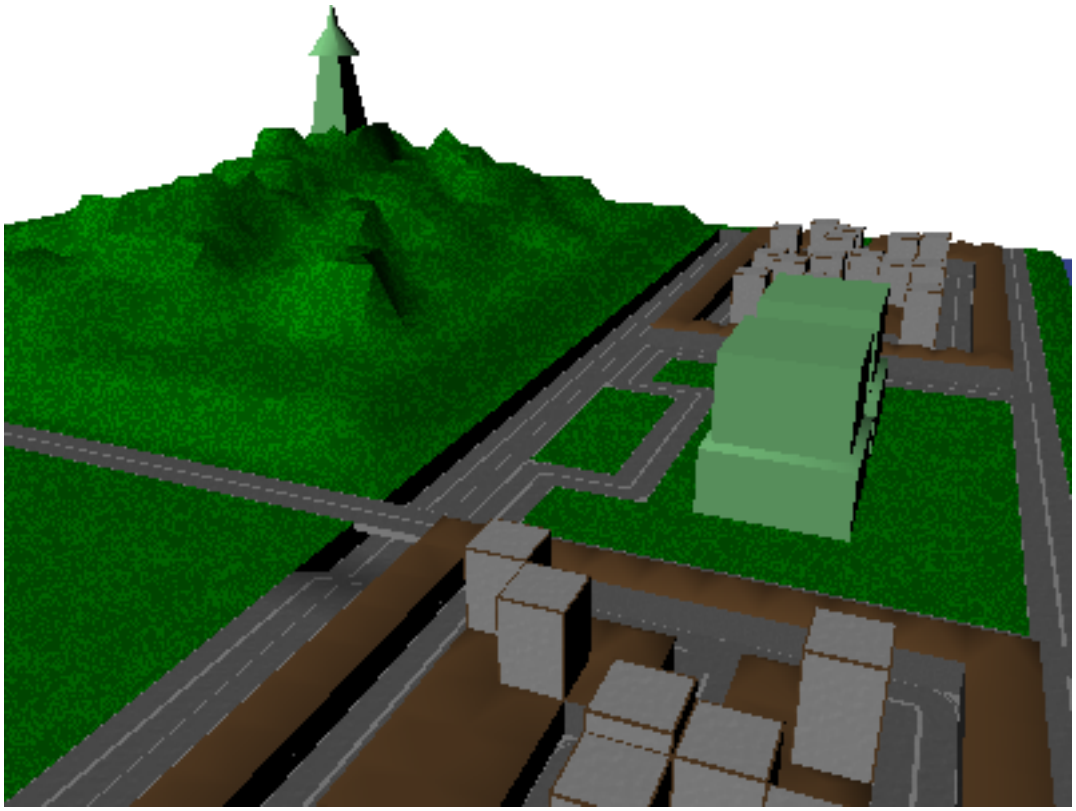

図 2.4: Lambert の余弦定理を使用して描画

$$H\left(P_1\right) \quad = \quad \frac{\left(F\left(P_{1r}\right) + F\left(P_{1g}\right) + F\left(P_{1b}\right)\right)}{3} \tag{2.3}$$

$F(P_1)$ に対して色要素を足し、要素の数で割る。これにより輪郭線は白黒の濃淡になる。本来輪郭線は黒で使うものだが、ここでは後の計算の為に濃淡のままとしてある。

2.2 スクリーントーンの選別

最初に陰影を付けないモデルの描画を行なう。Lambert の余弦定理を使用しないが、テクスチャやマテリアルをそのまま描画するのではなく光の色は考慮する。出力する色を I、物体の色成分を m、光の色成分を s とした時の式は次のようになる。

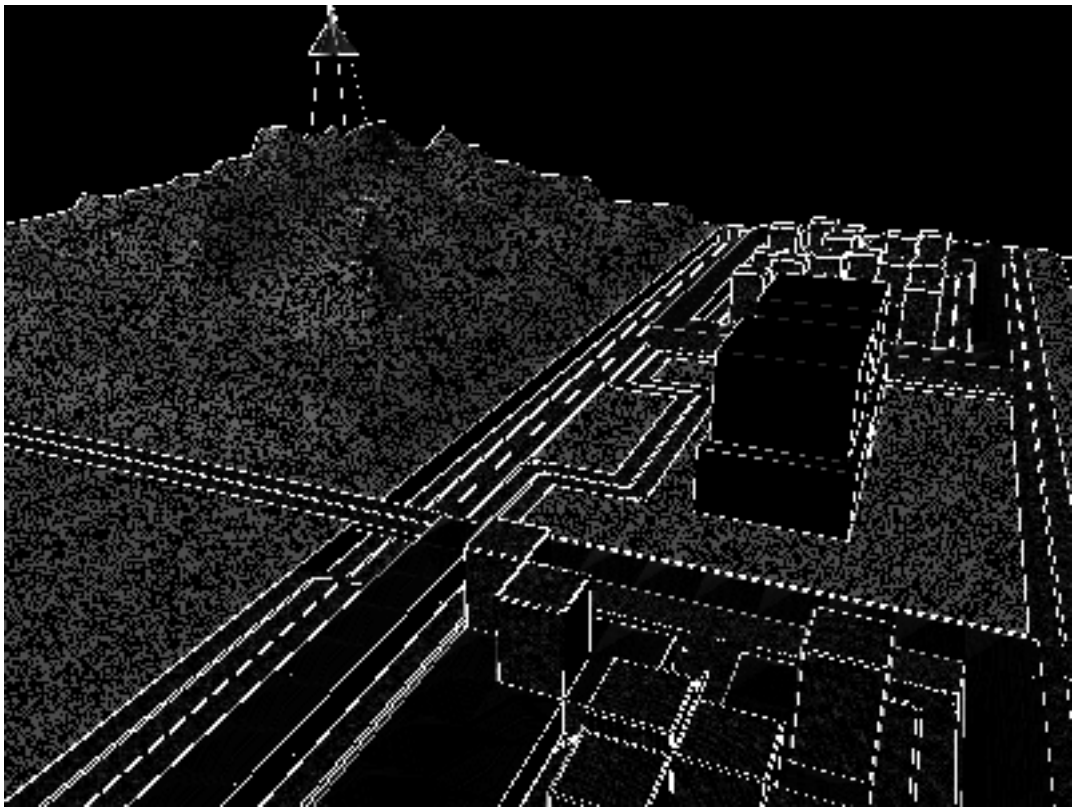

図 2.5: 輪郭線の抽出

$$I \quad = \quad (m \otimes s) \tag{2.4}$$

影の無いモデルが描画される。図 2.6 は影の無いモデルの描画である。

次に使うスクリーントーンの選別を行う。選別には色成分の内 α を除いた 3 色を使用する方法を使う。図 2.7 のように色成分の中で一番強い色を抽出し、その色成分は 1 を、それ以外の色成分は 0 を与える。この時に色成分の合計が 2 を超えていた場合は全ての色成分に 1 を与えて白と扱う。ここで決定された色に従い使用するスクリーントーンを選別する。図 2.8 は使用するスクリーントーンを示した物である。

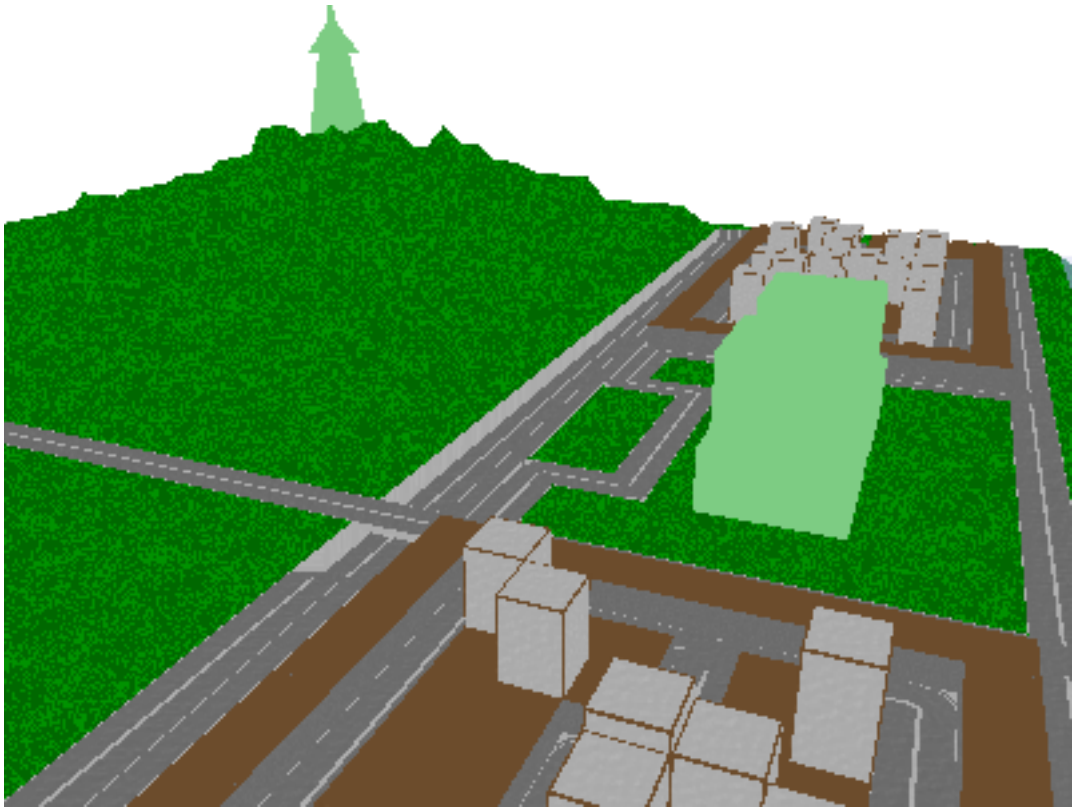

図 2.6: Lambert の余弦定理を使用せず描画

2.3　スクリーントーンの濃さの調整

最初にモデルの影だけを描画する。テクスチャやマテリアルは考慮しない。出力する色を J、表面の法線を $\hat{\mathbf{n}}$ とした時の式は次のようになる。

$$J = \hat{\mathbf{n}} \cdot \hat{\mathbf{l}} \tag{2.5}$$

影の付いている部分が黒く、そうでない部分は白くなる。

次にスクリーントーンの濃さの計算を行なう。漫画でスクリーントーンを使う場合、消しゴムやカッターで削る作業を行なう。この再現を目指す。再現には色の α 成分を使う。α 成分を K、底値を L とした時の式は次のようになる。

$$K = J + L \tag{2.6}$$

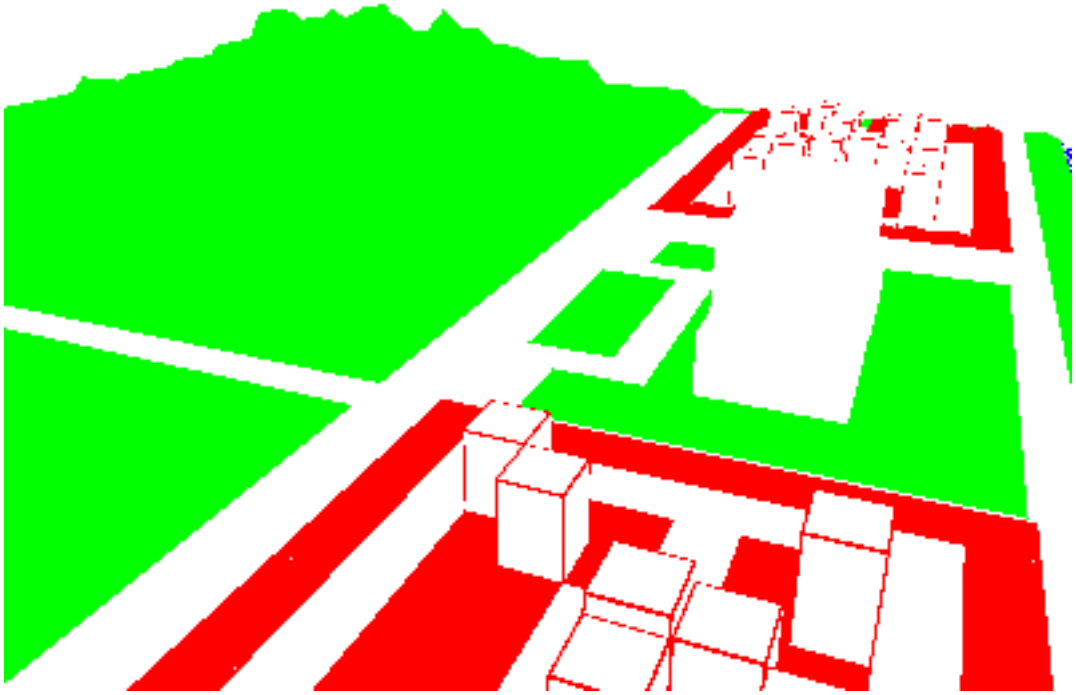

図 2.7: 色の抽出

L は全体の濃さを調整する為に使う。L を高くすると全体が濃くなり、最終的に薄い部分は無くなる。L を低くすると全体が薄くなり、最終的にスクリーントーンを貼らなくなる。

得られた K を先ほどの I の α 成分に代入する。

$$I_{\alpha} = K \tag{2.7}$$

これでスクリーントーンの濃さの調整は完了となる。

2.4 結果の合成

結果の合成を行う。抽出した輪郭線、選別したスクリーントーン、その濃さを合成する事でスクリーントーン表現を行う。最終的な色を C とした時の式は次の

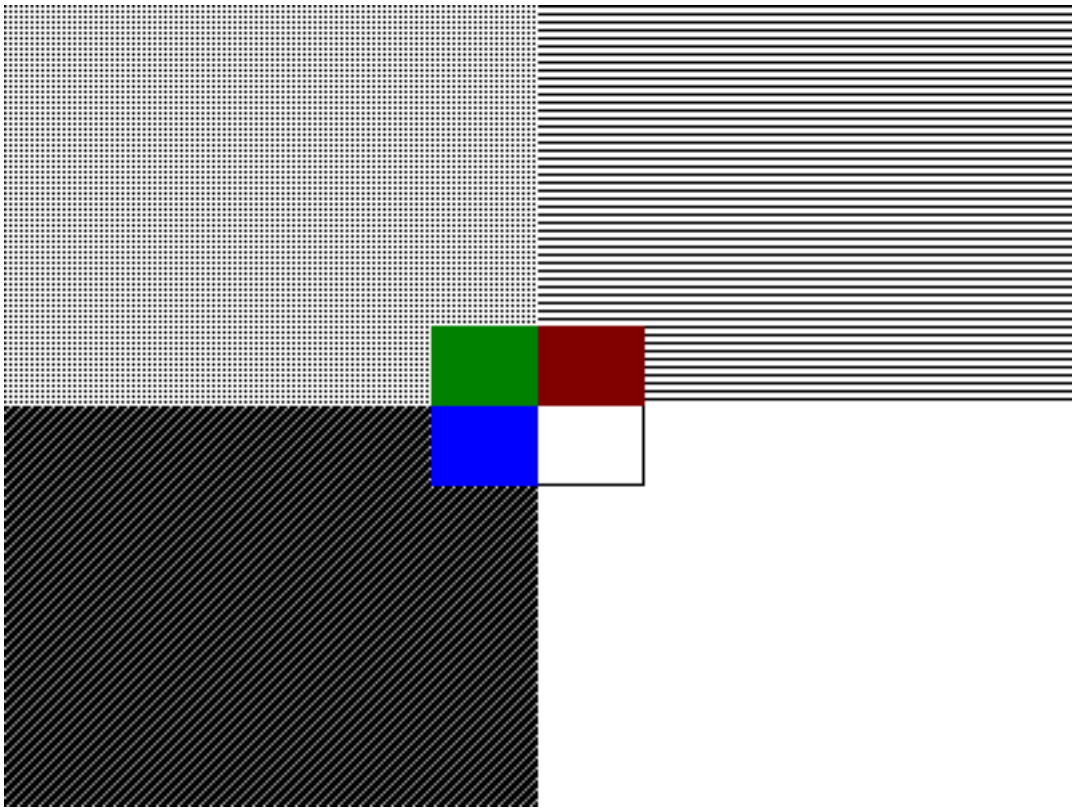

図 2.8: 色によるスクリーントーンの選別

ようになる。

$$C = T - HI \tag{2.8}$$

図 2.9 は最終的なスクリーントーン表現である。

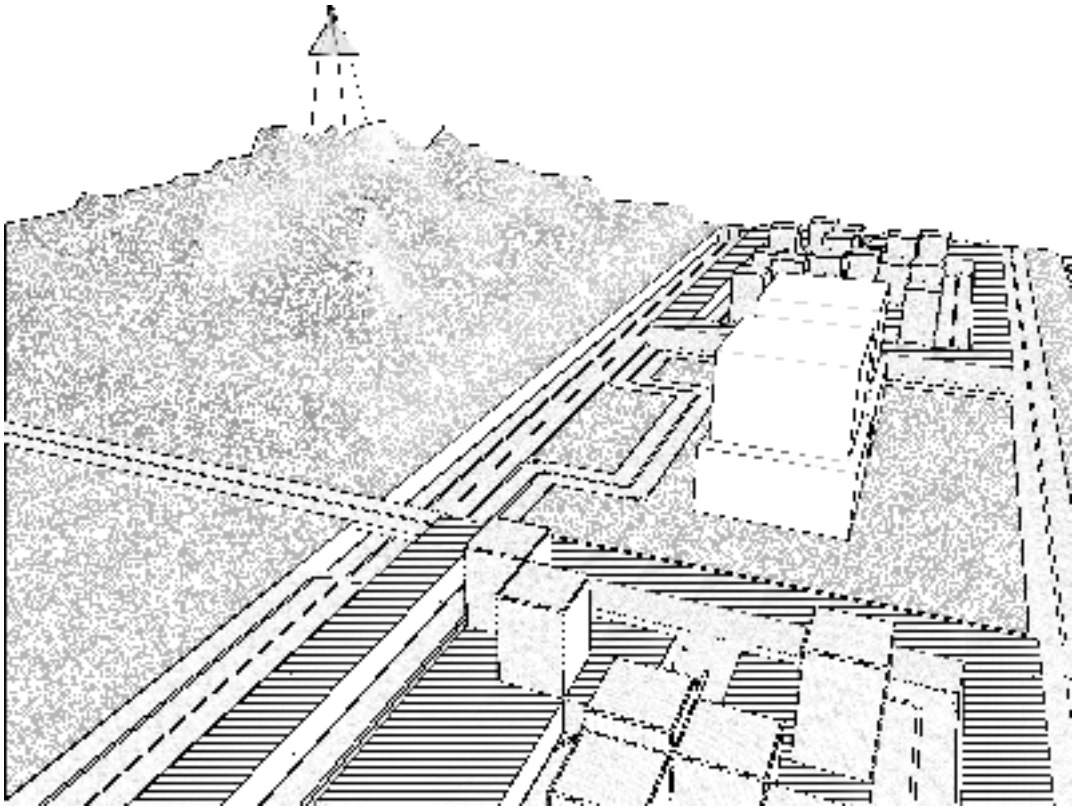

図 2.9: 最終的な漫画表現

第 3 章

有効性の検証

スクリーントーン表現は DirectX[18] と HLSL[19] を使用しシェーダ [20] として実装を行った。スクリーントーンは第 2 章中の図 2.2 を使用した。使用する命令と処理速度からヴァーテックスシェーダ及びピクセルシェーダのバージョンは 2.0 を使用した。

3.1 キャラクターモデル

キャラクターモデルへの検証を行う。これは人や動物と言った有機的な物である。図 3.1 はキャラクターモデルにスクリーントーン表現を行った物である。表 3.1 は使用したパラメータである。輪郭線の強調は余り行わず、不要な線の抽出が起こらないようにしている。スクリーントーンはやや濃くし、メリハリが出る様にしている。色の選別によって髪や服に別々のスクリーントーンを貼っている事が確認出来る。また肌色の部分は白と認識されスクリーントーンを貼っていない。服の模様も崩れずに描画しているが、これはスクリーントーンではなく輪郭線として抽出した物を描画した結果である。

3.2 メカニカルモデル

メカニカルモデルへの検証を行う。これはロボットや自動車と言った無機的な物である。図 3.2 はメカニカルモデルにスクリーントーン表現を行った物である。

表 3.1: キャラクターパラメータ

項目	パラメータ
輪郭線強度	5.0
スクリーントーンの濃さ	0.4

表 3.2: メカニカルパラメータ

項目	パラメータ
輪郭線強度	10.0
スクリーントーンの濃さ	0.0

表 3.2 は使用したパラメータである。輪郭線の強調を強めに行い際立たせる事で構造を判り易くしている。その為変化の小さい部分も抽出し、金属らしい光沢を出している。反対にスクリーントーンは強調せずそのまま使っている。腕や体、爪と各部の色を識別してスクリーントーンが分けられているのが判る。

3.3 背景モデル

背景モデルへの検証を行う。これは地面や木、建物と言った複合的な物である。図 3.3 は背景モデルにスクリーントーン表現を行った物である。表 3.3 は使用したパラメータである。輪郭線の強調はモデルの形が判る程度であり、細かな形までは取らない。同じくスクリーントーンの濃さはやや強め程度に抑えている。草原の描画を崩さず描画しており、丘の反射具合も抑える事で自然らしさを出している。

表 3.3: 背景パラメータ

項目	パラメータ
輪郭線強度	8.0
スクリーントーンの濃さ	0.3

3.4　モデルを用いた動画

モデルを用いた動画への検証を行う。これはキャラクターモデル、メカニカルモデル、背景モデルを合わせ動画にした物である。動画の場合は画面のどれかに焦点を合わせてパラメータを設定する事になる。キャラクターが画面中央に大きく映っているならばキャラクターに合わせたパラメータ、メカニカルが大きく映っているならばメカニカルに合わせたパラメータ、そして遠景の場合は背景に合わせてパラメータを設定する。パラメータの合っていないモデルは最適な表示を行う訳ではないが大きく崩れる事は無いので動画の場合は容認出来る。しかしモデルが動いた時に点と輪郭線の動きが一致しない為、ちらつきを感じる。

3.5　リアルタイム性の保持

モデルを用いた動画を使って Lambert の余弦定理を使用した基本描画と漫画表現描画の実行速度の比較を行った。比較に使用したパソコンの条件は以下の通りである。

- CPU　PentiumD 2.80GHz
- メインメモリ 2.00GB
- GPU GeForce7800GTX
- 解像度 SVGA(800×600)

GPU はシェーダバージョン 3.0 まで対応しており、今回の 2.0 も対応している。プログラムで FPS の制御を行わない状態で動画を再生させ、FPS を記録した。表 3.4 は記録の結果である。

FPS を比べた結果漫画表現描画は基本描画よりも速度は落ちる。しかし標準である 60 を大きく上回る数値を出しておりリアルタイムで使える事が確認出来る。速度が落ちるのはシェーダの処理が増えるからであり当然の結果であるが、シェー

表 3.4: 実行速度の比較

項目	FPS
基本描画	620
漫画表現描画	380

ダのメインがポストエフェクトの為モデルの量が増えてもそれほど実行速度に影響は無い。よって大量にモデルを使う事になるゲームにおいても本手法はリアルタイムで使えると考える。

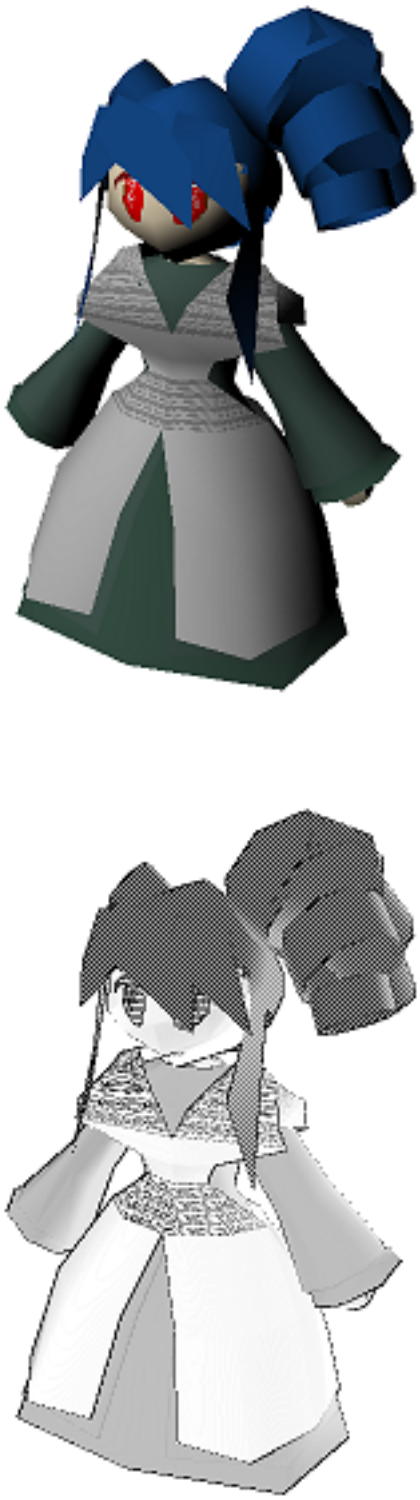

図 3.1: キャラクターモデルに適用

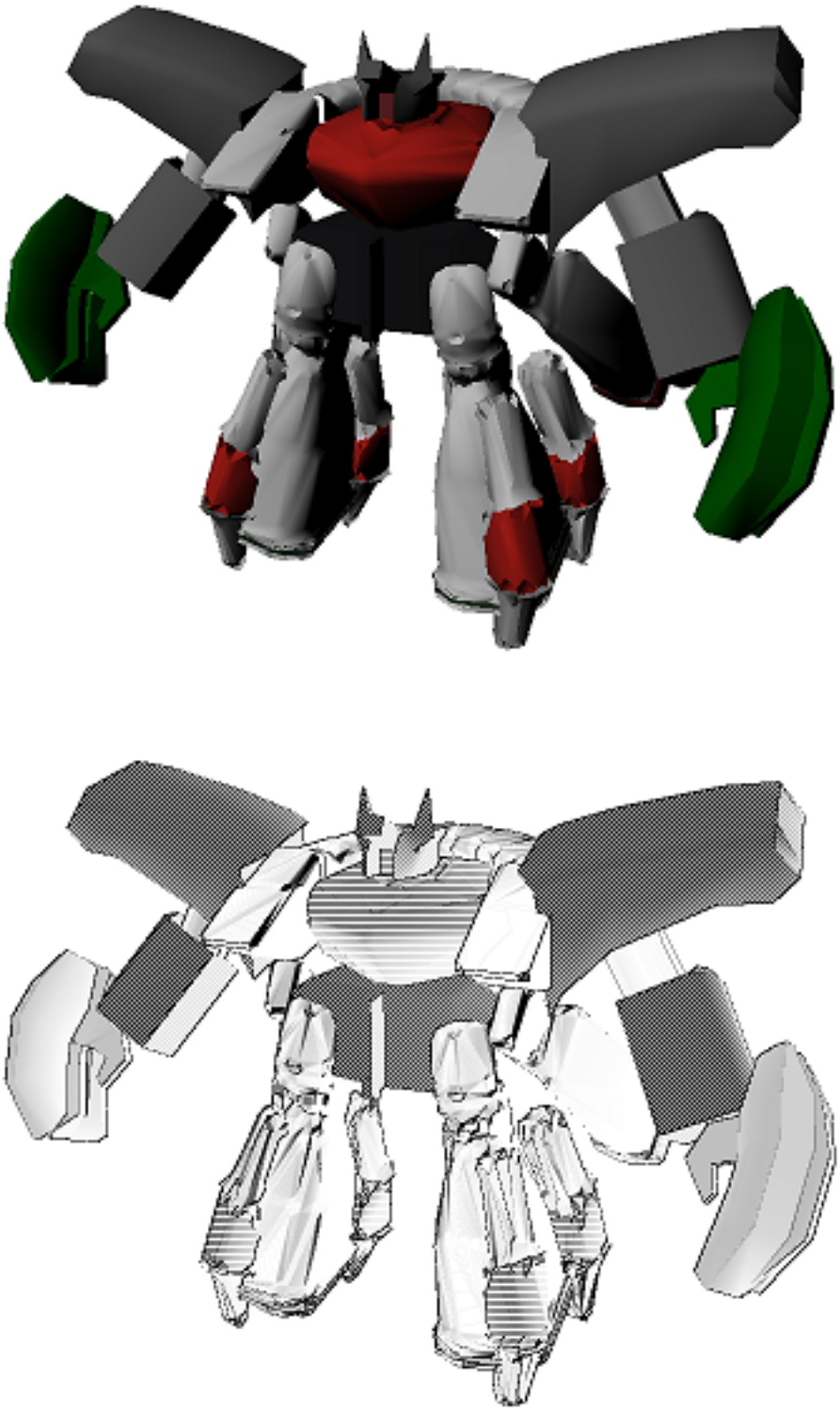

図 3.2: メカニカルモデルに適用

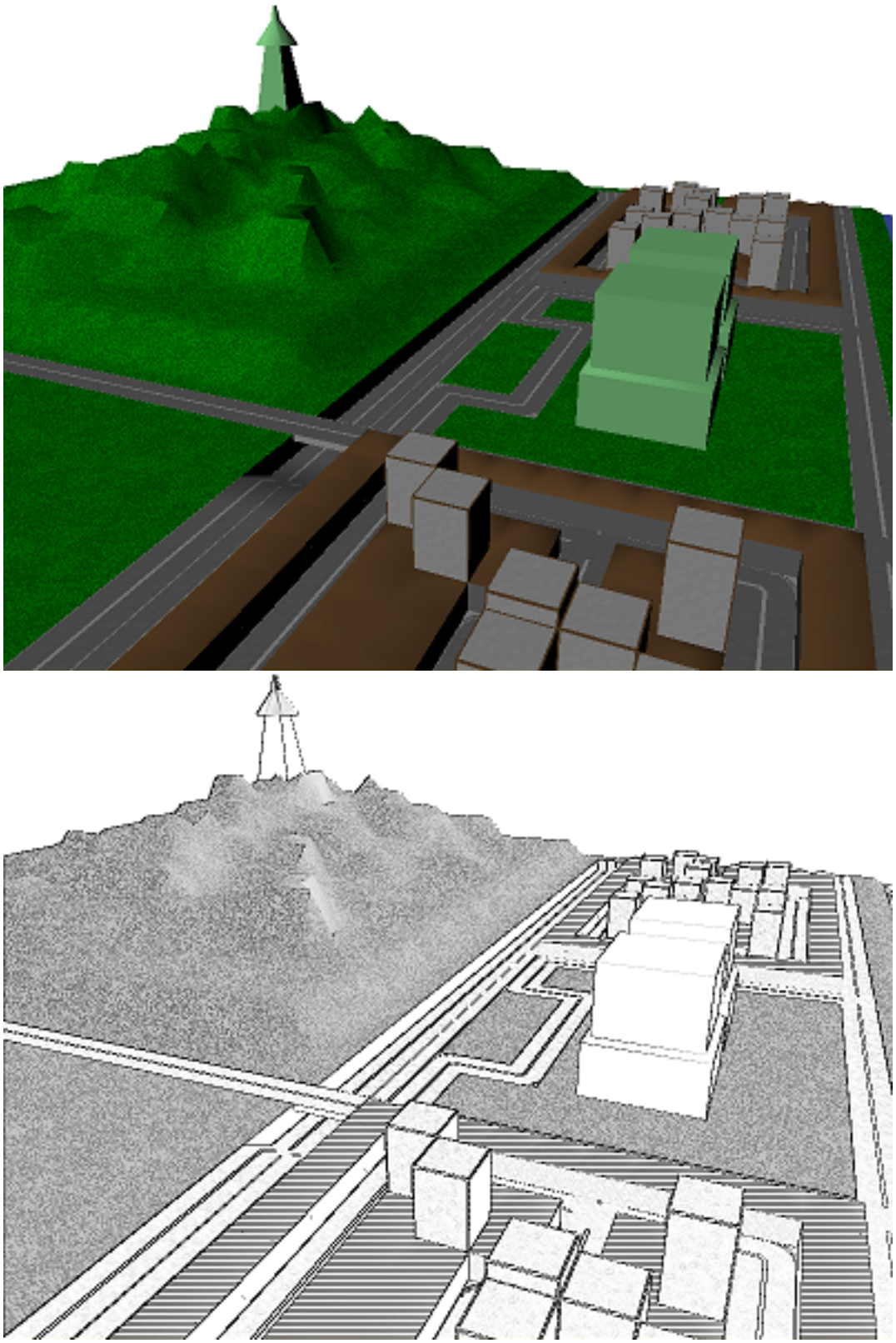

図 3.3: 背景モデルに適用

第 4 章

問題点

検証の結果得た問題点を挙げる。最初にちらつきが挙げられる。これはモデルの動きとスクリーントーンの動きが一致していない事から来ている。必要な部分に上からスクリーントーンを貼り付けているので輪郭線やスクリーントーンの濃さと模様が一緒に動かない。これを改善するにはモデルにテクスチャをマッピングする必要があるが、カメラやモデルが動く度にマッピングをしなければ点の模様がずれてしまう。

次に適用するモデル毎にパラメータの調整が必要である。特にキャラクターモデルとメカニカルモデルではパラメータの調整が反対にあり、どちらかに合わせる事になる。背景モデルも考慮すると遠景ではメカニカルモデル寄りの設定が良いが、人間や動物が大勢居る空間ではキャラクターモデル寄りに設定するのが好ましい。輪郭線の抽出も色による識別だけであり奥行きを考慮しておらず、背景と似たいような色の輪郭線が抽出出来ない場合がある。これは深度比較を適用する事で解消出来ると考えるが今度はテクスチャマッピングで同じ深度で輪郭線を出したい場合に対応出来ない。色差エッジ抽出と深度比較を両方使うだけでは輪郭線の強調が激しくなってしまうので上手な使い方が求められる。

最後にクオリティが解像度に左右される事が挙げられる。本来のスクリーントーンは点の濃さや密度で色や模様を表現しているので、それを意図通りに再現しようとすると大きな解像度が必要となる。大きな解像度を用意出来ない場合は、そ

れに合わせてスクリーントーンの模様が粗くなってしまう。目的の解像度に合ったテクスチャを用意する事で WUXGA や Full HD にも対応する事が出来るが処理速度が大幅に下がってしまう。映りのクオリティを高める為に解像度を高めるか、リアルタイム性を高める為に解像度を抑えるかは目的に応じて変える必要がある。

第 5 章

まとめ

本手法を使う事で 3DCG に対して漫画表現を行う事が出来、トゥーンシェードとは違ったアプローチによるデフォルメ表現を行う NPR を実現する事が出来た。実装には DirectX と HLSL を使い、マルチパスレンダリングを行う事で表現を維持しつつも処理速度を保つ事でリアルタイム性の保持に成功した。スクリーントーンとして使うテクスチャを交換する事で任意のスクリーントーンを使用する事が出来る。しかし動きによるちらつきや解像度によるクオリティの問題は改善出来ず、課題として残っている。今後はこれらの課題を解決しつつ更なる漫画表現への発展を行いたいと考える。

謝辞

本研究を行うにあたり指導してくださった渡辺大地先生を初めとした先生方に多大な感謝の意を示したいと思う。また共に研究を進め、支えあったゲームサイエンスの研究生達に対してありがとうと言葉を述べたい。最後に, 研究の原動力となった Fine, Rein, Lione, Mirlo, Milky, Altezza, Sophie, Ichele, Nina, Saya, Shiyon, Gorchel, Loloa, Nursya, Harney, Quarry, Julia, Joiner, Pearl といった多くの Princess に情熱と感動を持って感謝する。Thank you Prinses Fain.

参考文献

[1] ディジタルイメージクリエイション編集委員会, “ ディジタルイメージックリエイション ”, ディー・ディー・エス, pp119-120, 2002.

[2] Randima Fernando, Mark J.Kilgard, “ The Cg Tutorial 日本語版 ”, ボーンデジタル, pp242-243, 2003.

[3] Mark DeLoura, “ Game Programming Gems ”, ボーンデジタル, pp436-443, 2002.

[4] L.Streit, O.Veryovka and J.Buchanan, “ Non-photorealistic Rendering Using an Adaptive Halftoning Technique ”, Skigraph, 1999.

[5] Oleg Verevka and John W.Buchanan, “ Halftoning with Image-Based Dither Screens ”, Graphics Interface, 1999.

[6] L.Streit and J.Buchanan, “ Importance Driven Halftoning of 3D Scenes ”, EUROGRAPHICS, 1998.

[7] A. Secord, 井上光平, 浦浜喜一, “ 3D 物体の鉛筆画風画像の生成 ”, 映像情報メディア学会誌 Vol.60, 2006.

[8] A. Secord, W. Heidrich and L. Streit, “ Fast Primitive Distribution for Illustration ”, Proc. 13th Eurographics Rendering Workshop, 2002.

[9] 井上光平, 浦浜喜一, “ 確率分布に基づく点再配置法のモノクロ多値点描画とカラー点描画への拡張 ”, 映像情報メディア学会誌 Vol.57, 2003.

[10] Victor Ostromoukhov, “ Digital Facial Engraving ”, Proc. SIGGRAPH, pp. 417-424, 1999.

[11] T. Saito and T. Takahashi, “ Comprehensive rendering of 3-D shapes ”, Proc. SIGGRAPH, pp197-206, 1990.

[12] Microsoft Corporation, 複数のレンダリング ターゲット, <http://msdn.microsoft.com/library/ja/directx9_c/directx/graphics/programmingguide/advancedtopics/pixelpipe/multiplerendertarget.asp>.

[13] 鎌田茂雄, “ DirectX 逆引き大全 500 の極意 ”, 秀和システム, pp428, 2006.

[14] 星 正明, “ DirectX9 実践プログラミング ”, 工学社, pp152, 2003.

[15] Ron Fosner, “ Real-Time Shader Programming 日本語版 ”, ボーンデジタル, pp41-44, 2002.

[16] 金谷 一郎, “ 3D-CG プログラマーのためのリアルタイムシェーダー ”, 工学社, pp22-25, 2004.

[17] 今給黎 隆, “ DirectX9 シェーダプログラミングブック ”, 毎日コミュニケーションズ, pp465-467, 2004.

[18] Microsoft Corporation, Microsotf DirectX, <http://www.microsoft.com/japan/Windows/directx/default.mspx>.

[19] Intel Corporation, インテル用語集, <http://www.intel.com/jp/business/glossary/58402774.htm>.

[20] 松浦健一郎, 司ゆき, “ ゲームエフェクトマニアックス ”, ソフトバンククリエイティブ, pp4-5, 2006.