平成16年度 卒業論文

DirectXを利用した
トゥーンレンダリング

提出日 ： 2005年2月2日

指導 ： 大石 進一 教授

早稲田大学理工学部情報学科

学籍番号：1G01P022-3

大成 高顕

目 次

第1章　序論

1.1　背景

3DCG 技術の発展により、映画などのちょっとしたシーンでも３Ｄグラフィックスが使われ、実際に撮影した映像と殆ど見分けのつかない映像をつくることが可能になってきている。このようなフォトリアリスティクな手法が進歩する中、アニメや漫画のように写真的でない映像を作る手法が生まれてきた。その中でも代表的な手法に「トゥーンレンダリング」がある。日本製のアニメ映画が、世界規模で配給決定するなど、ジャパニメーションに対する世界の評価は高い。しかし、アニメや漫画を作るためには、作画という点だけからみても、アニメーションするために大量のセル画が必要であったり、その大量のセル画を作るために、ひとつの作品を同一人物が手がけることができず、場面によってキャラクターの容姿が変わったりする。アニメーションや漫画の制作の負担をできるだけ軽減することは、アニメーション産業や漫画産業において大きな課題なのである。

1.2　本論文の目的

本研究の目的は、写実的でない映像を作る手段として最も一般的なトゥーンレンダリングを DirectX を用いて、実際に行なってみることで DirectX の使用に慣れるとともに、1 つのモデルを利用し、多方向からの描画をアニメ的漫画的に違和感なく表現する方法についての模索を行なうことである。背景で述べたような製作の負担の軽減という点について、3D でモデルをつくり、それをトゥーンレンダ

リングで違和感なく表現できるのなら、モデルを使いまわすことができ、描画という点に関して、かなりの負担軽減になるのではないだろうかと考えている。

1.3　本論文の構成

本論文の構成は以下の通りである。

第 2 章では，準備としてトゥーンレンダリングの基礎概念・基礎知識等、DirectXについての基礎知識、使用プログラミング言語にした理由について述べる。
第 3 章では，今回作成したトゥーンレンダリングのプログラムの作成工程について述べる。
第 4 章では，本章では，前章で行った実際のトゥーンレンダリングプログラムの製作の結果について述べる。
第 5 章では，前章の結果を元として本論文の結論を述べる。

第2章　準備

2.1　はじめに

本章では，トゥーンレンダリングの基礎知識，基礎概念および、DirectX についての基礎知識、使用プログラミング言語にした理由について述べ。

2.2　トゥーンレンダリングとは

トゥーン（TOON）レンダリングとは、英語の「catoon」(漫画) から来た言葉で、漫画やアニメ的な表現をするためのレンダリングのことである。トゥーンレンダリングというのは、アニメ風の段階的な陰影を付けるトゥーンシェードと、輪郭線を描画するための輪郭線抽出の２つの技術から成り立っている。トゥーンシェードは極端に強調された陰影付けであり、通常は滑らかに変化する陰影を 2 色や 3 色で段階的に表現するものである。ただし、トゥーンシェードをすると同じ色の物体が重なったときに全て同じ色で描かれてしまい、まるで一つの塊のように描画されてしまう。それを解決するのが輪郭線で、トゥーンシェードをするなら輪郭線も絶対に必要になる。

2.2.1　トゥーンシェード

一般的なシェーディングでは Lambert の拡散反射モデルが使われる。これは面の法線ベクトルとライトの逆ベクトルとの内積から陰影を計算する方法である。内積というのは 2 つのベクトルが全く同じ方向を向いているときに最大となり、正

反対の方向を向いているときに最小の値になる。ようするに、面がライトに対して正対している状態が一番明るくなる。さて、トゥーンシェードをするには、この滑らかな陰影をどこかで「明るい/暗い」の 2 値に分ける必要がある。しかし、この明るい部分と暗い部分の境界はほとんどの場合で頂点と頂点の間に存在するので、普通に計算したのでは頂点単位の処理では無理である。PixelShader を使ってピクセル単位で処理すればできそうであるが、テクスチャを使えばもっと簡単に処理することができる。実際に計算するのは色ではなくテクスチャ座標で、色の変化はテクスチャに任せることになる。

2.2.2 輪郭線

同じ色で描画された別の物体の区別をつけるために、輪郭線が必要となる。まず、頂点シェーダを使用して、正規化ビューベクトルと、頂点法線との内積を計算する方法がある。値が 0 の場合は輪郭線にあり、値が 0 に近い場合は輪郭線に近いことがわかる。大まかに言うと、内積の値は、頂点が輪郭線のエッジにどれだけ近いかを示しているといえる。また、通常通りにモデルを描画したあとに、モデルを少し拡大して裏面だけを輪郭線の色で描画する方法もある。これは、深度テストの条件を「先に描かれているピクセルと Z 値が同一か大きかった場合のみ描画される」、つまり、通常のモデルと同じか、より奥にある場合だけ描画するように条件を設定するということである。

2.3 DirectX とは

DirectX とは、Microsoft 社が同社の Windows シリーズのマルチメディア機能を強化するために提供している拡張 API 群である。DirectX を使うと、アプリケーションソフトが統一的な手法を用いて直接ハードウェアを制御することが可能になるため、ゲーム機などの専用ハードウェアに負けない高度なマルチメディア処理をパソ

コン上で実現することができる。DirectX は 3 次元グラフィックスや音声を多用したゲームの開発に利用されることが多い。主に Windows 98/Me/2000/XP で動作させることを主眼に開発されているため、Windows NT では DirectX 3 まで、Windows 95 では DirectX 8.0 までしか動作しない。DirectX は用途に応じて様々な API 群に分かれており、DirectDraw(2 次元グラフィックス)、Direct3D(3 次元グラフィックス)、DirectSound(音声)、Direct3DSound(3 次元サウンド)、DirectInput(ジョイスティックなどの入力機器)、DirectMusic(ソフトウェア MIDI)、DirectPlay(ネットワーク対戦ゲーム)、DirectShow(大容量マルチメディアデータのストリーミング再生) などが用意されている。今回は 3D を扱うということ、将来的に Windows で動かすソフトウェアの作成を目標としているので、DirectX を用いてトゥーンレンダリングを試みることにした。

2.4 むすび

本章では準備としてトゥーンレンダリングの説明および、DirectX についての説明を行った，次章ではトゥーンレンダリングを行なうプログラムについて説明する。

第3章　製作

3.1　はじめに

本章では、今回作成したトゥーンレンダリングのプログラムの作成工程について述べる。

3.2　製作工程

3.2.1　一次元テクスチャを使用する

段階的な塗りわけをするにあたって、高さが1ピクセルである「一次元テクスチャ」を使用することにする。このテクスチャを、引き伸ばして貼り付けることで、テクスチャと同じ割合で、モデルを塗り分けることができる。

実装方法

SetTextureStageStateメソッドの第三パラメータをD3DTTFF_COUNT1にセットすることで、一次元テクスチャを使うことが宣言する。

3.2.2　ライティングを行なわない

一次元テクスチャを利用して塗りわけをするので、一般的に使われるライティングによる影の描画は必要なくなる。

実装方法

SetRenderState メソッドを SetRenderState(D3DRS_LIGHTING, FALSE);
のように書くことで、ライティングを行なわないようにする。

3.2.3 法線ベクトルをテクスチャ座標にする

一次元テクスチャを貼るための、テクスチャ座標を求めるにあたって、頂点における法線ベクトルの X 座標を用いることにする。Y 座標については、一次元テクスチャを利用しているために、考えなくて良いことになる。

実装方法

```
SetTextureStageState メソッドを
SetTextureStageState(0,D3DTSS_TEXCOORDINDEX,
D3DTSS_TCI_CAMERASPACENORMAL);
```

のように書くことで、法線ベクトルをテクスチャ座標として扱えるようになる。ただし、法線ベクトルの各成分がとる値は、-1.0 から 1.0 の範囲であるのに対し、テクスチャ座標の各成分がとる値は、0.0 から 1.0 の間と決められているので、変換しなくてはならない。
（テクスチャ座標）＝｛(法線ベクトルの値）× 0.5｝＋ 0.5
としてやることで、法線ベクトルのとる値を 0.0 から 1.0 の間に変換することができる。

```
D3DXMATRIX mTexture, m1, m2;
D3DXMatrixScaling( &m1, 0.5f, 1.0f, 1.0f);
D3DXMatrixTranslation( &m2, 0.5f, 0.0f, 0.0f );
```

```
D3DXMatrixMultiply(&mTexture, &m1, &m2);
```

3.2.4 カメラ空間での法線ベクトルを求める

光源の位置を、シェーディング結果に反映するために、カメラ空間での法線ベクトルを求めることにする。

Mw：ワールド行列、D3DTRANSFORMSTATE_WORLD で設定

Mv：ビュー行列、D3DTRANSFORMSTATE_VIEW で設定

N:カメラ空間の法線ベクトル

Nobject:物体の法線ベクトル

Mwv：Mw*Mv

N = Nobject * (Mwv-1)T

よって、カメラ空間の法線は、オブジェクトの法線に逆転置行列 Mwv を乗算し、その結果を正規化することによって計算される。

実装方法

上の計算を実際に書くと以下のようになる

```
D3DXMatrixInverse(&matTrans1,NULL,&m_view);
D3DXMatrixTranspose(&matTrans2,&matTrans1);
D3DXVec3Transform(&light1, &mVecDir, &matTrans2);
light1.w = 0;W 値を消去
D3DXVec4Normalize(&light2,&light1);
matTrans0 = D3DXMATRIX(
```

```
-light2.x,0,0,0,
-light2.y,1,0,0,
-light2.z,0,1,0,
0,0,0,1
);
// V値を0〜1にスケーリング
D3DXMatrixScaling( &matTrans1, -0.5f,1.0f,1.0f );
D3DXMatrixTranslation( &matTrans2, 0.5f,0,0 );
matTrans = matTrans0 * matTrans1 * matTrans2;
```

3.2.5 輪郭線の描画の実装

輪郭線を表示するために、すこし拡大したモデルの裏面を、輪郭線にしたい色で描画してやる。

実装方法

```
pD3DDevice->SetRenderState(D3DRS_CULLMODE, D3DCULL_CW);//裏面と指定
// メッシュの色を設定（輪郭線の色になる）
int numVertices = m_pMesh->GetNumVertices();
MY_VERTEX* v;
m_pVBMesh->Lock( 0, 0, (LPVOID*)&v, 0 );
for (int i = 0; i < numVertices; i++) {
v[i].color = D3DXCOLOR(0.0f, 0.0f, 0.0f, 1.0f); // 黒色に設定
}
m_pVBMesh->Unlock();
```

```
// 輪郭線幅の指定は少しだけ拡大することで行う
D3DXMatrixScaling(&mWorld, 1.03f, 1.03f, 1.03f);
  pD3DDevice->SetTransform(D3DTS_WORLD, &mWorld);
```

次に、モデルの表面を、白色で描画してやる。
先ほどのプログラムから、拡大に関するところを抜き、色を白に設定してやる。
表面を描画するコマンドは、一行目の最後を D3DCULL_CCW と書き換えてやればよい。
以上の二つを、同時に描画してやることで、輪郭線が表示できる

3.2.6 テクスチャ付トゥーンレンダリング

これまでは、モデル自体にテクスチャが貼られていないものを扱ってきたが、テクスチャが貼られているものをトゥーンレンダリングする方法を考察してみた。まず、テクスチャなしの状態で、段階的な塗りわけを行い、その後、テクスチャステージで掛け合わせることで、テクスチャ付でもトゥーンレンダリングできるのではないかと考えた。

実装方法

```
SetTextureStageState(0,D3DTSS_COLOROP, D3DTOP_MODULATE );
SetTextureStageState(0,D3DTSS_COLORARG1, D3DTA_TEXTURE );
SetTextureStageState(0,D3DTSS_COLORARG2, D3DTA_DIFFUSE );
SetTextureStageState(1,D3DTSS_COLOROP, D3DTOP_MODULATE );
SetTextureStageState(1,D3DTSS_COLORARG1, D3DTA_CURRENT );
SetTextureStageState(1,D3DTSS_COLORARG2, D3DTA_TEXTURE );
```

DirectX において、１つのテクスチャステージでは、テクスチャが１枚しか扱えないので今回のように、２種類のテクスチャを合成したいときは、テクスチャステージを二つ用意してやる必要がある。モデルのメッシュの色（3行目）と、一次元テクスチャ（2行目）を掛け合わせてやり（1行目）、その出力を5行目で呼び出し、本来のテクスチャ（6行目）と掛け合わせてやっている（4行目）。

3.3　むすび

本章では、実際に自分の作ってみたトゥーンレンダリングプログラムの実装方法についての説明を述べた。次章では本研究の結果について述べる。

第4章　結果

4.1　はじめに

本章では，前章で行った実際のトゥーンレンダリングプログラムの製作の結果について述べる．

4.2　トゥーンシェード

前章で述べた、「1 次元テクスチャを使わない」、「ライティングを行なわない」、「法線ベクトルをテクスチャ座標に変換」、「カメラ空間での法線ベクトル」を実装した結果及び、通常のシェーディング方法での描画結果を次のページに載せてある（図 4.1、4.2 参照）。また、モデルに関しては、DirectX のサンプルから teapot.x を使用した。きちんと段階的に塗りわけられていることがわかると思う。

図 4.1: 通常のシェーディング

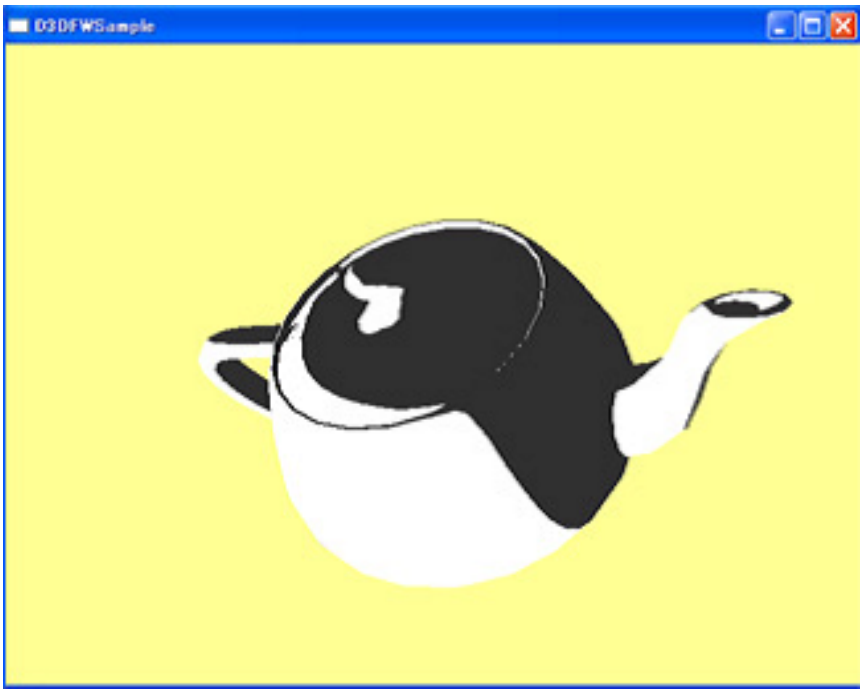

図 4.2: トゥーンシェード

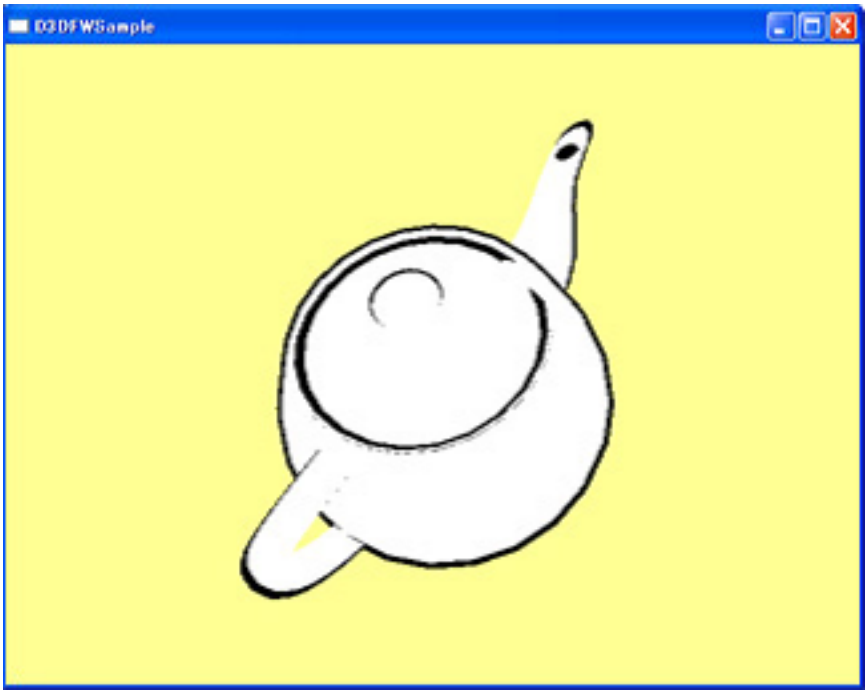

図 4.3: 輪郭線を描画

4.3 輪郭線の描画

前章で述べたように、輪郭線を表示するために、すこし拡大したモデルの裏面を、輪郭線にしたい色で描画し、その後通常通りのサイズでモデルを描画した（図 4.3 参照)。きちんと輪郭線が描画されているのがわかると思う。

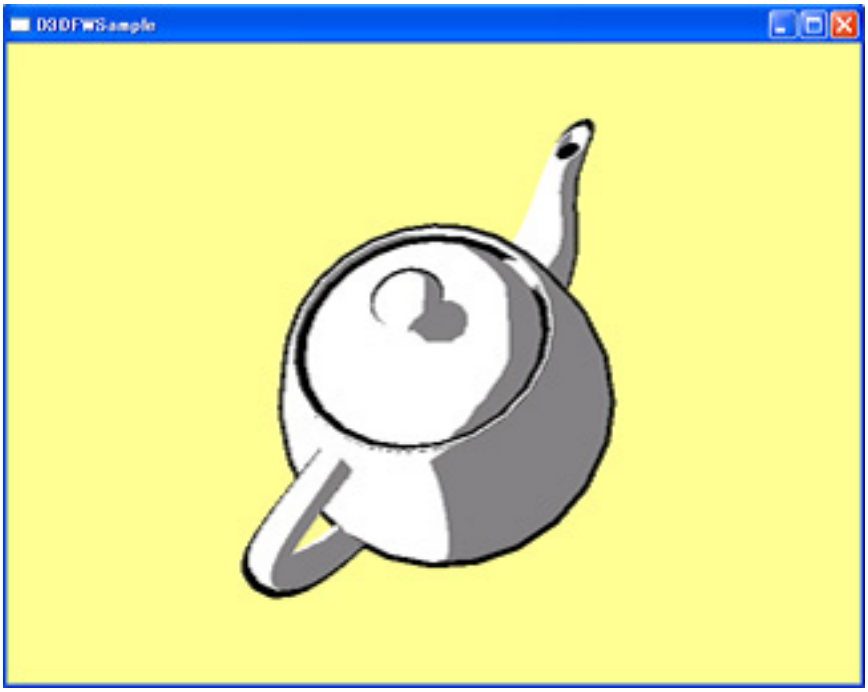

図 4.4: トゥーンレンダリング

4.4 トゥーンレンダリング

トゥーンシェードと輪郭線の描画を同時に取り入れて、トゥーンレンダリングした（図 4.4 参照）。きちんと輪郭線が表示され、段階的に塗りわけられていることがわかると思う。

4.5 テクスチャ付トゥーンシェード

テクスチャ付の場合のモデルについてトゥーンシェードを試みた。比較できるように、通常のテクスチャ付シェーディングと、トゥーンシェーディングの二つを次のページに載せておく（図 4.5、4.6 参照）。きちんとテクスチャ付で段階的に塗りわけられていることが、多少違和感を感じる結果になっている。モデルは DirectX のサンプルから tiger.x を使用した。

図 4.5: テクスチャ付の通常のシェーディング

図 4.6: テクスチャ付トゥーンシェード

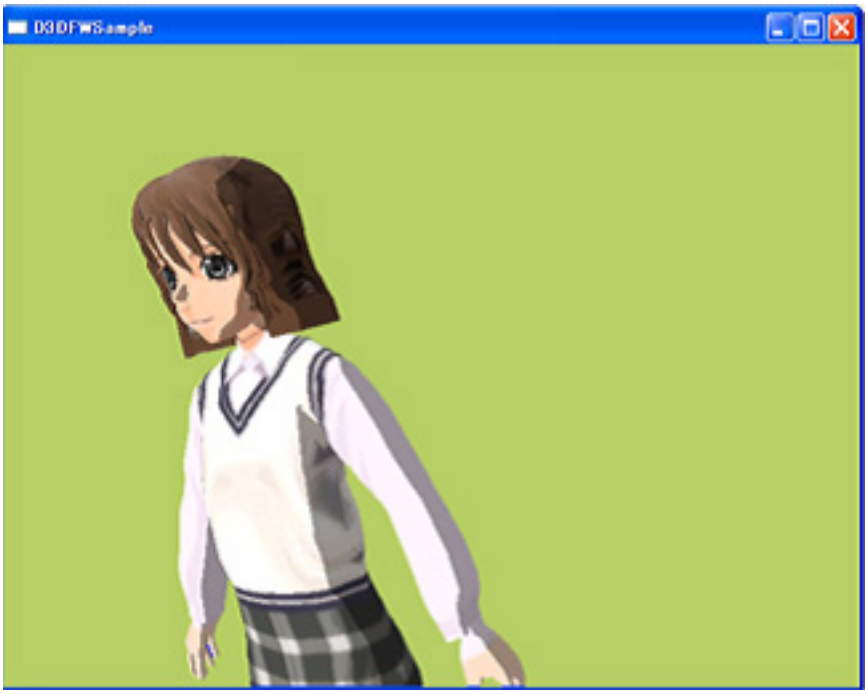

図 4.7: 人体モデルのテクスチャ付トゥーンシェード

4.6　人体モデルのテクスチャ付トゥーンシェード

アニメや漫画的なキャラクターを自分で作り、それに対してトゥーンシェードを行なってみた。モデルの作成には六角大王 SUPER4 の人体作成モード機能を使用した。モデルが複雑になるにつれ、現状の方法で違和感なくレンダリングすることが難しくなってきた。今回は、1 次元テクスチャの割合を変化させることで、多少違和感を軽減させることに成功したが、それでもまだ違和感が残っている。

4.7　むすび

この章では実際に行なったトゥーンレンダリングの結果について述べた．次章ではこの結果を元に考察を行う．

第5章　結論

5.1　はじめに

本章では，本論文の結びとして，本論文の結論を述べる。

5.2　統括

第 2 章では，準備としてトゥーンレンダリングの基礎概念・基礎知識等、DirectXについての基礎知識、使用プログラミング言語にした理由について述べた。
第 3 章では，今回作成したトゥーンレンダリングのプログラムの作成工程について述べた。
第 4 章では，本章では，前章で行った実際のトゥーンレンダリングプログラムの製作の結果について述べた。
第 5 章では，前章の結果を元として本論文の結論を述べた。

本論文の構成は以上のようであった。

5.3　結果のまとめ

現状の方法でも、ティーポッドのような単純な形をしたモデルについては、きちんとトゥーンレンダリングが行なわれることがわかった。しかしその一方で、動物や人体をモデルとした場合、現状の方法では形状の複雑さについていけず、違

和感のあるレンダリングがされてしまうという結果になった。人体をモデルとしたときは、1次元テクスチャの白黒の割合が5割では違和感があまりにも大きかったので、1次元テクスチャの割合を変更させて対処することで対処した。

5.4 考察

上で取り上げた研究テーマについて考察する。

5.4.1 陰影について

1次元テクスチャを利用するやり方は、1次元テクスチャの白黒の割合がそのままモデルの陰影に反映される為、正面から人間の顔などをレンダリングした際に、左半分は白、右半分は黒というような形になってしまう。この事について、今回は1次元テクスチャを絵画ソフトで作り直し、白黒の割合を変更することで対処した。しかし、モデルを変えるたびに絵画ソフトを起動して、1次元テクスチャを作り直すのは大変手間のかかる作業である。この問題を解決するために、プログラム内の数字により1次元テクスチャ自体を作成し、それを利用できるようにするのが良いのではないかと考えた。イラストの作成が、感覚的に行なわれていることもあるので、影用のバーなどを用意し、それを左右に動かすことで白黒の割合を決め、同時にその変更がモデルに対しても適用されるというような形が使いやすいと考えられる。

5.4.2 モデルについて

レンダリングはモデルの形自体を変えることは出来ないので、どうしてもレンダリング結果はモデルの質に依存することになる。よりアニメ的より漫画的に違和感のないトゥーンレンダリングを行なうには、モデル自体もそれに適した形状

をしていることが望ましいと考えられる。今回は、六角大王というソフトの人体作成モードというのを利用して、人型のモデルを作成した。この機能を使うと、一枚の正面画から、ある程度の質を持った人体モデルが、3 時間ほどで作成できる。今回の研究の目的には、3D モデルを使用することにより現状のアニメや漫画の作成の手間を減らすというものがあるので、モデルの作成時間も 3 時間程度で行うというのが理想的である。しかし、3 時間程度で作れるモデルでは、トゥーンレンダリング後、違和感が生じてしまった。レンダリング方法の模索とともに、容易に作成でき、なおかつトゥーンレンダリング向きのモデルを作る方法を考えることも研究の結果を良いものにするために必要だと感じられた。

5.4.3　そのほかの方法について

ベジュ曲線を 3D 空間において処理し、線のみで 3D モデルを作成するという方法を考えてみた。漫画・アニメが境界線を基本として成り立っているので、中身の詰まった 3D モデルを作成する必要性は特にないと考えられる。この方法を利用することで、不必要な部分を作りこむ手間を省くことができ、より短時間で、モデルを作ることを出来ると考えられる。。漫画やアニメにおいて、どの程度の曲線をもって立体を認識させているのかなどを研究し、まず最小限の線でのモデルを作り、そこからオリジナルの線をモデルに加えていけるような形が、感覚的にわかりやすいのではないかという考えに至った。

5.5　むすび

本研究の目的は、写実的でない映像を作る手段として最も一般的なトゥーンレンダリングを DirectX を用いて、実際に行なってみることで DirectX の使用に慣れるとともに、1 つのモデルを利用し、多方向からの描画をアニメ的漫画的に違和感なく表現する方法についての模索を行なうことであであった。拙いながらも

トゥーンレンダリングのプログラムを製作し，それを通して、現在のトゥーンレンダリングプログラムをさらに発展させ、モデルを違和感なく表示できるようにすることと、モデル自体を、違和感なく表示できるような形で作り出していくというふたつの研究テーマを考えた．今後は、さらに DirectX や、トゥーンレンダリング、3D モデルの作成ついての勉強を進め，この研究テーマを実現に向けて努力していく必要がある．

謝辞

本研究を進めるに当たり, 終始丁寧な御指導及び御激励を賜り, その他多くの面でも色々と御面倒を見て下さり御助言を与えて下さいました大石進一教授，に深く感謝いたします.

また, 終始丁寧な御指導と御教示をして下さいました大石研究室助手, 中谷 祐介氏, 丸山 晃佐氏, 荻田 武史氏, に大いに感謝いたします.

最後に，日常生活において色々とお世話になりました，大石研究室の皆様に深く感謝いたします.

参考文献

1. 大石進一, 牧野光則:”グラフィックス”, 日本評論社,1994 年,

2. N2Factory:DirectX ゲームグラフィックスプログラミング, ソフトバンクパブリッシング株式会社,2004 年,

3. 大川善邦, 大澤文考, 登大遊, 成田卓郎:DirectX ９実践プログラミング,I/OBOOKS, 2004 年,

参考 Web ページ

1. Tokio's Web Site

   http://www.tgws.fromc.jp/tokio6/

2. t-pot

   http://imagire.zive.net/t-pot/

3. Masafumi's Laboratory

   http://masafumi-t.cool.ne.jp/

4. MSDN ライブラリ

   http://www.microsoft.com/japan/msdn/default.asp