2002年度　卒　業　論　文

リアルタイム３ＤＣＧツールキット上での絵画調レンダリングの実装とその効果に関する研究

指導教員：渡辺 大地 講師
若林 尚樹 助教授

メディア学部３ＤＣＧアプリケーション構築プロジェクト
学籍番号　99p273
丹治　宏文

２００３年3月

2002 年度　　卒　業　論　文　概　要

論文題目

リアルタイム３DCG ツールキット上での
絵画調レンダリングの実装とその効果に関する研究

メディア学部 **学籍番号:** 99P273	**氏名**	丹治　宏文	**主査**	渡辺 大地 講師
			副査	若林 尚樹 助教授

キーワード	リアルタイム３DCG、ノンフォトリアリスティックレンダリング ツールキット、プログラマブルシェーディング

近年リアルタイム 3DCG はハードウェアの進歩によって、今まで処理に時間が掛かっていた表現を用いることが可能になってきている。その中でもノンフォトリアリスティックレンダリングについては研究が盛んになるとともに、様々な CG システムなどで実用化も行われており目にする機会も増えてきた。しかし実際にノンフォトリアリスティックレンダリングを使用したアプリケーションを制作しようとした場合 3D のコア API やプログラマブルシェーダーと呼ばれる特有のアセンブリを用いる必要があり、現状では実現のための技術的な障壁が高い。本研究では、これらの機能をカプセル化したクラスライブラリツールキットを構築し、そのツールキットを用いることによりノンフォトリアリスティックレンダリングの実装が非常に容易となったことを示す。併せて、ツールキットを用いた開発の効率化についても調査を行う。

目次

第１章　はじめに……1
第２章　問題の分析……3
2.1　既存技術の問題点……3
2.2　問題の解決策……4
第３章　NPR ツールキットの開発……6
3.1　NPR ツールキットの設計……6
3.2　NPR ツールキットの構成……8
3.3　NPR ツールキットの機能……12
3.3.1　頂点データとオブジェクトクラス……13
3.3.2　テクスチャークラス……14
3.3.3　フィルタークラス……15
3.3.4　レンダークラス……15
3.4　実装環境について……16
3.4.1　DirectX について……16
3.4.2　VertexShader と PixelShader……16
第４章　NPR ツールキットの評価……19
第5章　まとめと今後の展望……23
謝辞……24
参考文献……25

第1章　はじめに

近年リアルタイム３DCG はグラフィックスハードウェアの高性能化に伴って研究が多く行われ、技術的に大きく進歩している。そのなかでも特に注目され研究開発が盛んである分野にノンフォトリアリスティックレンダリング（non-photorealistic rendering：ＮＰＲ）がある。今まで３DCG の主流は、プリレンダリングにおいてもリアルタイムレンダリングにおいても、より写実的な画像を生成するフォトリアリスティックレンダリング（photorealistic rendering）であったが、1990 年の SIGGRAPH において NPR がセッションとして取り上げられ、そのセッションの論文[1,2,3]により広く知られるようになった。その後 NPR のメリットである情報の抽象化、誇張、特徴の抽出を行うことによって、情報の送り手が伝えたいものをより協調することが出来るという点から多くの研究が行われるようになってきている。そしてリアルタイムでの NPR についての研究も盛んに行われるようになり、輪郭線を抽出したシルエットのレンダリングや[4]、技術書の挿絵風のレンダリング[5]、スケッチ風のレンダリングといった NPR が登場するようになる。またトゥーンレンダリング[7]というアニメーションなどのセル画風の NPR はゲームで実装されており、よく目にするようなってきている。

しかし実際にリアルタイム NPR を使ったプログラミングを行おうとした場合、現在開発環境の選択肢として、DirectX か OpenGL を３D のコア API として、それとともにアセンブラにより記述をおこなうプログラマブルシェーダーと呼ばれるものを使用しなければならない。アセンブラはコンピュータが実際に処理する機械語により近い低水準言語であり、用意された１つ１つの命令はごく単純な処理を行う。それらを組み合わせて処理を記述するため、プログラムを見ても実際にプログラマブルシェーダーとしての動作が分かりづらく、またプログラミングを行う際にもコーディングし辛いという欠点があり、それが要因となり技術的な敷居を高くしていると考えられる。そのため開発者が目的のアプリケーションを開発する前にこれらの技術的な学習をしなければならず、それは場合によってはプログラミングに要する期間よりも多く掛かってしまうこともあり、大幅な開発効率のダウンになってしまう。

一方でプログラミングにおいて開発効率を高める手法としてツールキットというものがある。ツールキットとは高度な処理を行う関数を用意しカプセル化して開発者に提供するクラスライブラリ群である。ツールキットのメリットとして、提供されている高度な機能

を利用する際に、内部の構造を理解したり学習したりせずに開発を行うことが出来るため、目的のアプリケーション制作までに必要な学習時間を省くことができ、効率よく開発を行うことが出来るという点がある。またツールキットを使うことにより高度な機能が少ない行数で使用できるためソースコードの量を減らすことが出来、プログラムの開発において大きな変更が行われるときでも、少量の変更で済むという点も挙げられる。リアルタイム３DCG プログラミングの分野においても以前からいくつかのツールキットが開発されてきた。しかし今までのリアルタイム３DCG ツールキットでは、形の定義やオブジェクトの動作に関して容易に扱えるようにするものが主であり、レンダリングの部分に重点を置き特に NPR をサポートするツールキットは存在しなかった。そこで本研究ではリアルタイム３DCG ツールキットに対して NPR を実装することにより、開発者がプログラマブルシェーダーを意識することなく、NPR を使用したアプリケーション制作を行えるようにすることを目的とする。そして実際に開発した NPR ツールキットによって、NPR をつかったアプリケーションの開発に対する敷居を下げ、また開発効率に対しての効果について考察を行う。

第2章　問題点の分析

この章では既存の技術による問題点と本研究での問題の解決策の紹介を行う。

2.1　既存技術の問題点

これまでのグラフィックスハードウェアは単一のシェーディングを行うレンダリング機能しか搭載していなかった。そのためリアルタイムにノンフォトリアリスティックレンダリング（以後 NPR）を行う場合は、グラフィクスハードウェアによりレンダリングされた画像に対して CPU における後処理によって画像を生成する方法を取っていた。しかし近年のグラフィックスハードウェアの高性能化とリアルタイムレンダリングにおける研究により、グラフィックスハードウェア上において、プリレンダリングで実用化されているプログラマブルシェーダーを動作させることが出来るようになった。プログラマブルシェーダーとは、開発者がプログラムを使ってレンダリングにおけるシェーディング処理を変更させられる手法のことである。このことにより CPU を使わずにグラフィクスハードウェアだけの処理で高速に NPR の生成を行うことが可能になった。しかし実際にプログラマブルシェーダーを使って NPR を実装しようとした場合以下のいくつかの問題点が考えられる。

- シェーディング処理を開発者が記述しなければならない
- NPR を生成するアルゴリズムを知らなければならない
- プログラマブルシェーダーはアセンブラで記述しなければならない

これまでのグラフィックスハードウェアは単一のシェーダーを持っておりシェーディングの処理は頂点情報を用いて数学的な演算を行っていた。プログラマブルシェーダーはそれに代わるものであり、開発者はこれまで行われていたシェーダー処理をプログラマブルシェーダーによって記述しなければならない。そしてシェーディング処理を開発者が変更させるためにはシェーディングの内部処理に関する知識が必要になってくる。なにより NPR は通常のシェーディング処理と違い、法線方向だけでなく奥行き情報、テクスチャー座標、色情報などを操作しなければならず、アルゴリズム的に理解が困難である。そしてプログラマブルシェーダーはグラフィックスハードウェアのシェーディング処理を変化させるその性質上、より機械語に近いアセンブラ言語を使用して記述しなければならない。

このようにプログラマブルシェーダーの実現により、NPR を用いたアプリケーションの開発を行う環境は整ったが、実際に開発を行おうとした場合はまだ問題点が残っている。

2.2 問題の解決策

プログラマブルシェーダーを使って NPR を実装しようとした時問題となるのは、アセンブラによるプログラミングとシェーディング及び NPR のアルゴリズムである。

nVidia の Cg 言語[8]はこの問題に対して１つの解決策を提示している。Cg 言語はＣ言語に似た記述によりプログラマブルシェーダーを作成することが出来る。Cg 言語で記述されたプログラマブルシェーダーは Cg 言語コンパイラによって DirectX 向けもしくは OpenGL 向けの頂点シェーダーとピクセルシェーダープログラムを出力する。開発者はこの出力された頂点シェーダーピクセルシェーダープログラムを自分のプログラムに組み込むことによってプログラマブルシェーダーを使用することが出来る。そのため Cg 言語を使用すると開発者はアセンブラよりプログラマブルシェーダーの処理内容を把握しやすくなる。

しかし実際に開発者が Cg 言語を使ったとしても、記述が容易になるだけでシェーディングにおける処理に関する技術は知らなければならず、NPR シェーダーを作る場合には NPR のアルゴリズムを知らなければならない。つまり Cg 言語はある程度アセンブラでプログラマブルシェーダーを扱える人にとっては開発効率の向上につながるが、まだ初めてプログラマブルシェーダーを扱う人にとっては、アセンブラによるプログラマブルシェーダーと同じく敷居の高い技術であると考えられる。

そこで本研究では、以前からプログラミングにおいて使われてきたツールキットと呼ばれる手法を使ってこの問題の解決法を提案する。ツールキットとはオブジェクト指向プログラミングにおけるクラス構造を使い、いくつかの高度な機能に必要なクラスを集めたクラス群のことである。クラスには処理に必要なデータと処理を行うメソッドがカプセル化されているため、開発者はメソッドに対して必要な手続きを行うだけで高度な機能が利用可能となる。そのため開発者はクラス内部の構造やメソッドの処理について知っている必要は無く、クラスに対する入力と出力だけを知っていれば良いことになる。つまり図１のように Cg 言語ではプログラミングシェーダーのアセンブラプログラミングをカプセル化しているが、ツールキットの手法を使うとプログラミングシェーダーのアセンブラプログラミングと NPR のアルゴリズムについてカプセル化することが出来る。

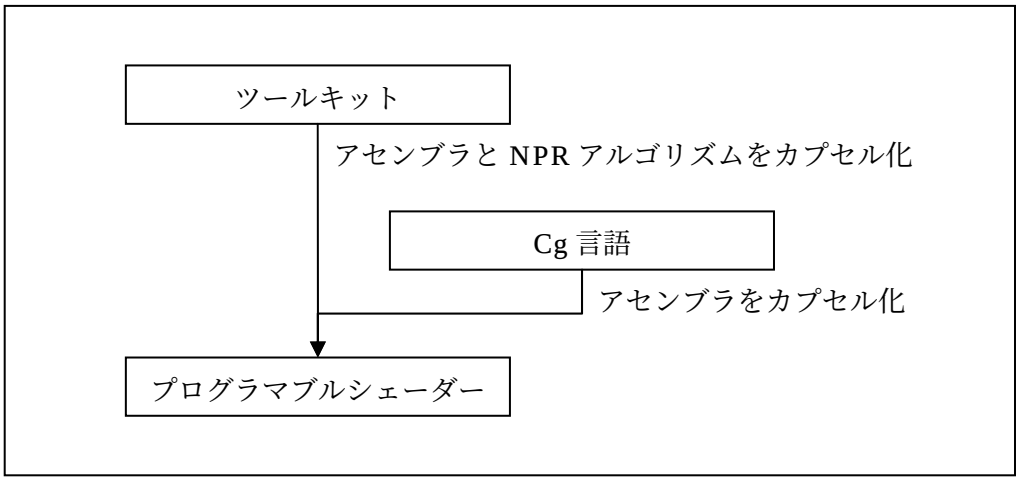

図 1　Cg 言語とツールキットのカプセル化

このツールキットにおいて NPR を使用することにより、開発者はプログラマブルシェーダーや NPR のアルゴリズムについて意識をすることなく、NPR を用いたアプリケーションの開発を行うことが出来る。このことより、NPR を用いたアプリケーションの開発に対する技術的な敷居を大きく下げることが出来ると考えられる。

また開発効率についても NPR に必要な機能をカプセル化して使用することが出来るので、単純にソースコードを見ても行数を大きく減らすことが出来ると考えられる。さらにアセンブラ、もしくは Cg 言語を用いて NPR 実装したアプリケーションを開発した場合、最初に実装した NPR シェーダーから別の NPR シェーダーに切り替えようとした場合、ソースコードの大幅な改変やプログラムの再設計が必要になる。だがツールキットを用いて NPR を実装した場合、シェーダーの変更による大幅なソースコードの改変やプログラムの再設計は必要なく、少量のソースコードを書き換えるだけで実現することが出来る。これらのことからツールキットを用いて NPR を実装することにより、NPR を用いたアプリケーションの開発における作業効率について大幅な向上が得られると考えられる。

第3章NPR ツールキットの開発

ここではツールキットにおいて NPR を実装する際におけるクラスの設計と、実際に DirectX を用いて実装したモデルについて述べる。

3.1 NPR ツールキットの設計

ツールキットに実装を行う NPR には以下の２種類の手法が存在している。

- 一度レンダリングされた画像に対してフィルターを掛ける
- レンダリング時に NPR 画像を生成する

まず１つ目に、一度シェーダーによりレンダリングを行って画像を生成し、その生成された画像に対して２DCG で使われているフィルター処理をプログラマブルシェーダーによって行うことにより NPR 画像を生成するものである。これは既存の２DCG において使われている技術を応用することが可能である。だが処理に時間の掛かるフィルターや他のピクセルを多くサンプリングするフィルターなどはまだ実装することが困難であるが、他のピクセルをサンプリングしないモノクロ化やセピア調変換といった色調補正や、エンボスのような少ないピクセルをサンプリングするフィルターなどについて作成することが出来る。

２つ目は３D によって得られる幾何情報やテクスチャーを使うことにより、レンダリング時にプログラマブルシェーダーによって NPR 画像を生成するものである。この手法による NPR では、奥行き情報を用いたエッジの検出による輪郭線表示や、法線方向を使ったトゥーンシェーディングなどがある。

以上のようにプログラマブルシェーダーによって実装可能な NPR には以上のような２種類の方法があり、ツールキットの開発ではその２種類について実装する必要がある。またそれぞれの NPR 画像を生成する際に必要となる機能として以下のようなものがある。

- 画像データを保持する機能
- NPR に必要な幾何情報を持つ頂点構造

フィルター型の NPR では一度レンダリングした画像に対してプログラマブルシェーダー

によるフィルター処理を行う。そのためレンダリングした結果をすぐにディスプレイに出力せず、一旦溜め込んで保持しておく必要がある。これにはリアルタイム３DAPI で用意されているテクスチャーを用いる。テクスチャーとは色情報の集まりで、多くの場合はポリゴンにマッピングすることによりポリゴンの表現を深めるために使われる。だが多くのリアルタイム３DAPI ではこのテクスチャーをレンダリングの出力ターゲットとすることで、レンダリングの結果をディスプレイではなくテクスチャーに出力することにより、レンダリングデータを保持することが出来る。この機能を使いテクスチャーをレンダリングの出力先として使用するためのクラスの設計を行う。まずレンダリングデータを保持するテクスチャーのクラスには次の機能が必要となる。

- フレームバッファのサイズ、または指定したサイズのテクスチャーを生成する
- テクスチャーをレンダリングターゲットとして使用する
- レンダリングターゲットを元のバックバッファに戻す
- 保持するデータをディスプレイに出力する

レンダリングターゲットに指定されたテクスチャーに保持されるデータは最終的にはフレームバッファに出力されるため、画像のサイズとしてフレームバッファと同じかもしくはそれより大きなサイズのテクスチャーを生成する機能を持つ必要がある。そして生成されたテクスチャーを実際にレンダリングターゲットに指定する機能と、レンダリングターゲットを指定する前のバックバッファを保持しておきレンダリングが終わった後に元の状態に戻す機能が必要である。最後にフィルターを掛けるため、保持しているテクスチャーを受け渡すための機能が必要となる。
レンダリングする機能とこれらの機能を備えたテクスチャーのクラスを作成する必要がある。

また NPR に必要な頂点の幾何情報とは以下の通りである。

- 位置情報
- 法線方向
- 複数のテクスチャー座標

位置情報はプログラマブルシェーダー内の演算により、スクリーン投影座標や奥行き情報に変換される。それと法線方向、テクスチャー座標を複数使用する場合が幾何情報を用いた NPR にはある。これらの情報を持った頂点データが必要になる。またリアルタイム３DCG では形状の宣言が必要になるため、この頂点情報を用いた形状の定義が必要となる。

以上のような機能を持ったツールキットとして、本研究では NPR ツールキットの開発を行う。

3.2 NPR ツールキットの構成

この節では前節で行った設計に基づいて開発した NPR ツールキットのクラス構成について述べる NPR ツールキットでは以下の機能を持つクラスを用意した。

- アプリケーションクラス･･･ウィンドウの作成と表示、３D コア API の初期化作業を行うクラス
- ウィンドウクラス･･･････カメラの役割をする１つのモデルと、シーン、NPR を生成するレンダー、フィルターを保持するクラス
- レンダークラス･･･････プログラマブルシェーダーを使い、頂点の幾何情報を用いて NPR 画像を生成するクラス
- フィルタークラス･･･････プログラマブルシェーダーを使い、テクスチャーに対してフィルターを掛けて NPR 画像を生成するクラス
- テクスチャークラス･･･････レンダリングの出力先として使用できるテクスチャーの生成とその保持をするクラス。
- シーンクラス･･･････････複数のライトとモデルを保持するクラス
- オブジェクトクラス･･･････オブジェクトの定義を行うクラス
- モデルクラス･･･････････オブジェクトの移動、回転、拡大を行うクラス
- ライトクラス･･･････････ライトの定義と動作を行うクラス

このツールキットにおけるクラス間のつながりは以下の通りである。

・ アプリケーションクラスはすべてのクラスの上位にあり、開発者はアプリケーションクラスを必ず１つだけ宣言する。

・ アプリケーションクラスは複数のウィンドウクラスを登録することが可能。

・ ウィンドウクラスは１つのシーンクラスとカメラの役割をする１つのモデルクラス、NPR レンダリングを行うレンダークラス、フィルタークラスを登録することが可能。

・ シーンクラスは複数のモデルクラスとライトクラスを登録することが可能。

・ モデルクラスは１つのオブジェクトクラスを登録することが可能。

・ オブジェクトクラスは複数のモデルクラスに対して登録することが可能。

これを図２で表す。ボックスはクラスを、矢印は登録先のクラスを指している。

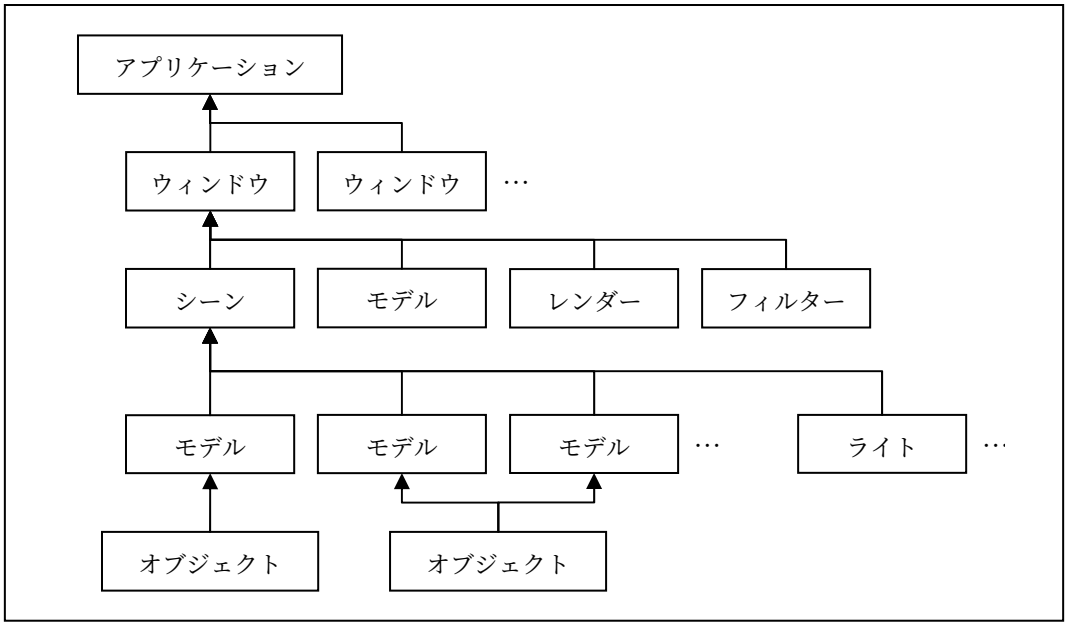

図２　ツールキットのクラス関係

このように各クラスには登録された別のクラスがあり、それぞれ登録されたクラスに対してレンダリングプロセスを辿っていくことによりこのツールキットでは NPR 画像の生成を行う。次にそのレンダリングプロセスの流れについて記述する。

レンダリングプロセスはフィルター型の NPR か幾何情報を使った NPR かで流れが異な

る。まずフィルター方の NPR では、開発者がプログラムの中でアプリケーションのレンダリングメソッドを呼び出したタイミングでレンダリングプロセスが始まる。

アプリケーションクラスは登録されているウィンドウクラスに対してレンダリングメソッドを呼び出す

- ウィンドウクラスではテクスチャークラスをレンダリングターゲットにして、登録されているシーンのレンダリングメソッドを呼び出す
- シーンクラスでは登録されている複数のモデルクラスに対してレンダリングメソッドを呼び出す
- モデルクラスでは登録されているオブジェクトクラスのレンダリングメソッドを呼び出す
- オブジェクトクラスは保持している頂点データをプロシージャルシェーダーに渡してポリゴンのシェーディング処理を行い、レンダリングターゲットに指定されているテクスチャーにレンダリング結果を保存する
- 全てのオブジェクトがレンダリングされたら、ウィンドウクラスはフィルタークラスに対してレンダリングターゲットに指定されていたテクスチャーを引数にしてレンダリングメソッドを呼び出す。
- フィルタークラスは入力されたテクスチャーに対してプログラマブルシェーダーを使ってレンダリングを行い、NPR 画像を生成してフレームバッファに対して出力する。

以上のレンダリングプロセスを図３に表す。プロシージャルシェーダーは NPR ではない既存で用意されているシェーダーを、フレームバッファは最終的に出力する画面を指す。黒矢印はレンダリング命令の流れを、白矢印と吹出しはデータの流れとその内容を表している。

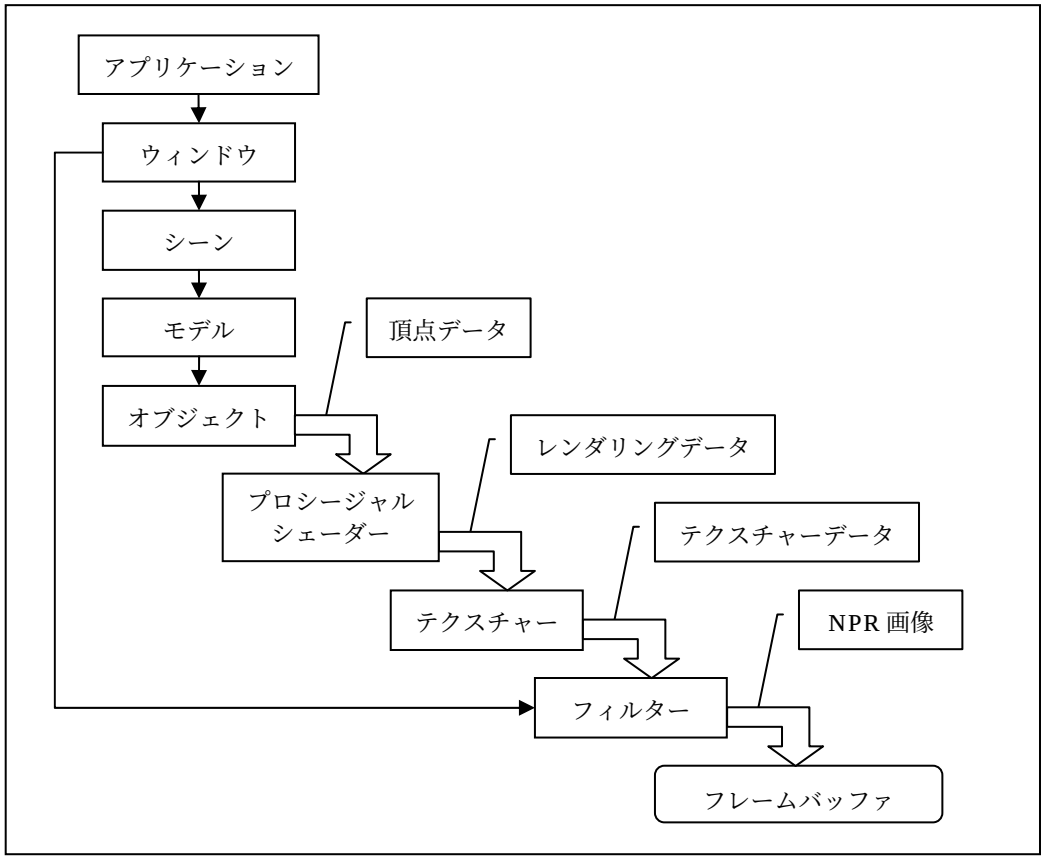

図３　フィルターを使った NPR 処理の流れ

また幾何情報を用いた NPR のレンダリングプロセスでも、開発者がプログラムの中でアプリケーションのレンダリングメソッドを呼び出したタイミングでレンダリングプロセスが始まり。

- アプリケーションクラスは登録されているウィンドウクラスに対してレンダリングメソッドを呼び出す
- ウィンドウクラスは登録されているシーンクラスを引数にレンダークラスに対してレンダリングメソッドを呼び出す
- レンダークラスでは内部でシーンクラスのレンダリングメソッドを呼び出す
- シーンクラスは登録されているモデルクラスに対してレンダリングメソッドを呼び出す

・ モデルクラスは登録されているオブジェクトクラスに対してレンダリングメソッドを呼び出す

・ オブジェクトクラスは保持する頂点情報をレンダークラスのプログラマブルシェーダーを使ってレンダリングを行い、NPR 画像を生成してフレームバッファに出力する

以上の処理を図４によって表す。フレームバッファは最終的に出力する画面を指す。黒矢印はレンダリング命令の流れを、白矢印と吹出しはデータの流れとその内容を表している。

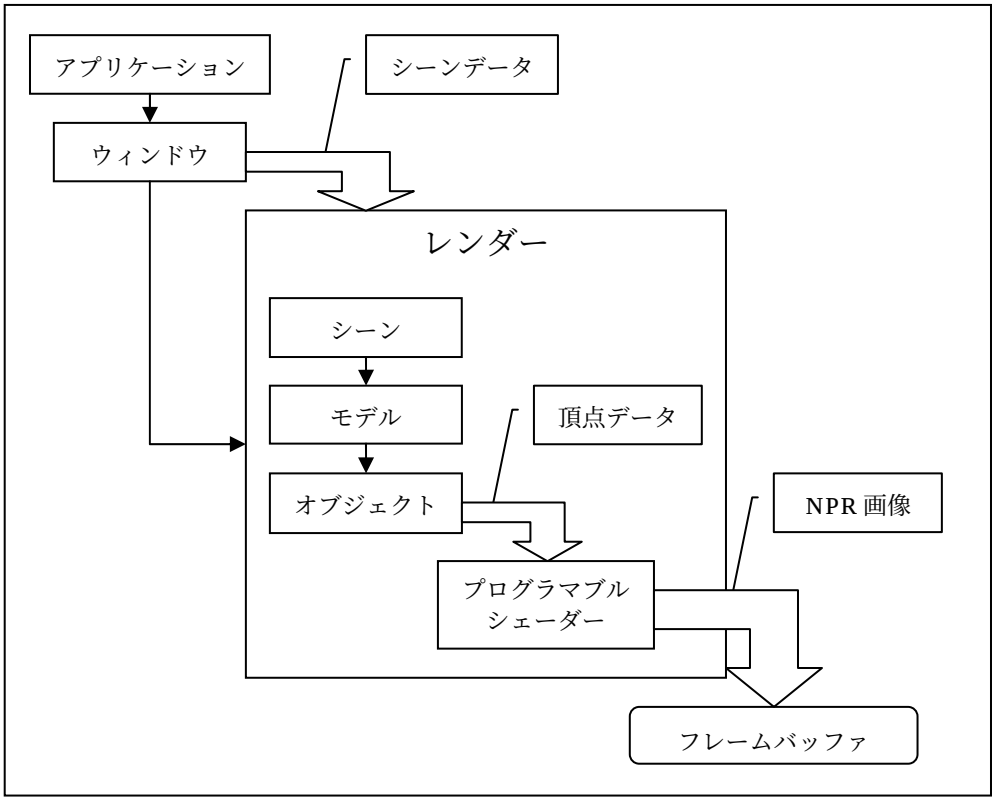

図４　幾何情報を使った NPR 処理の流れ

3.3　NPR ツールキットの機能

ここでは前節の構成の中から NPR に関するクラスの機能について述べる。

3.3.1 頂点データとオブジェクトクラス

NPR に必要な幾何情報をもった頂点データとして NPR ツールキットでは以下の幾何情報を持った頂点データについて定義する。

```
float x, y, z;        // 位置ベクトル
float nx, ny, nz;     // 法線ベクトル
float tu1, tv1;       // テクスチャー座標１
float tu2, tv2;       // テクスチャー座標２
```

またこの頂点データフォーマットによって形状を保持するオブジェクトクラスについて定義する。

```
class object
{
        virtual void render() = 0;
}
```

オブジェクトクラスは純粋仮想関数でインスタンスを作ることは出来ない。実際のデータや動作は図５で示すようなオブジェクトクラスを継承した派生クラスである BoxObject や SphereObject、XfileObject などで記述する。動作としては render()メソッドによって保持されている形状のレンダリングを行う。

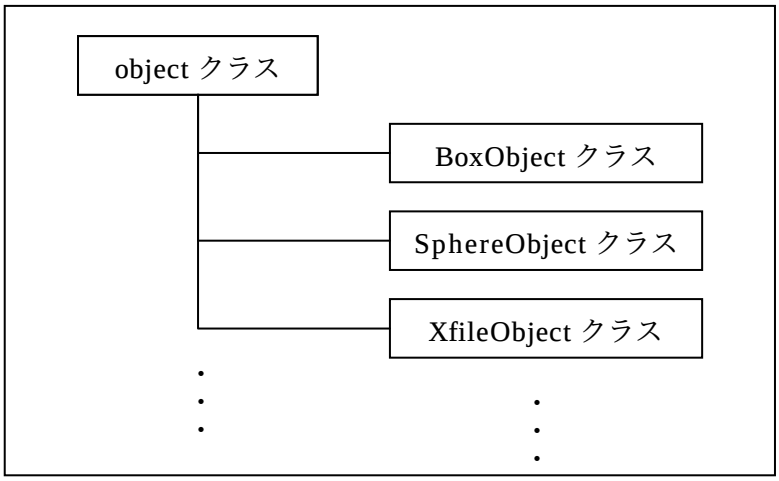

図５　オブジェクトクラスの派生クラス

3.3.2 テクスチャークラス

レンダリングターゲットとして使用できるテクスチャーの生成と、そのテクスチャーのレンダリングターゲットへの指定を行うクラスについて定義する。

```
class renderTarget
{
        void createRenderTarget();
        void setRenderTarget();
        void reset();
        texture getTexture();
}
```

createRenderTarget()によってフレームバッファサイズもしくは指定の大きさのレンダリングターゲットに指定できるテクスチャーを生成する。setRenderTarget()でテクスチャーをレンダリングターゲットに指定し、reset()によってレンダリングターゲットをバックバッファに戻す機能を持つ。getTexture()では保持しているテクスチャーを返すことによりポリ

ゴンにマッピングしてディスプレイに出力することが出来る。

3.3.3 フィルタークラス

フィルターによる NPR 画像を生成するクラスについて定義する。

```
class filter
{
        virtual void render(texture) = 0;
}
```

純粋仮想クラスでインスタンスは生成できない。実際の処理はフィルタークラスを継承した派生クラスに記述する。しかし動作としてはテクスチャーを入力し、そのテクスチャーに対してプログラマブルシェーダーによりＮＰＲ画像を生成して画面に出力する機能を持つ。

3.3.4 レンダークラス

レンダリング時に NPR 画像を生成するクラスを定義する。

```
class render
{
        virtual void render(scene) = 0;
}
```

レンダークラスは純粋仮想クラスでインスタンスは生成できない。実際の処理はレンダークラスを継承した派生クラスに記述する。動作としては、オブジェクトやライトをまとめて保持するシーンクラスを入力とし、レンダリング時に NPR 画像を生成して画面に出力する機能を持つ。

3.4 実装環境について

本研究ではツールキット制作における下位リアルタイム 3 DCG ライブラリとして DirectX を使用することにした。その理由として、まず一つに現在のパーソナルコンピュータ向けビデオカードが DirectX に最適化されており、DirectX を使ってプログラミングすることによりハードウェアの機能を最大限引き出すことが出来るということである。二つ目に DirectX8.0 からサポートされるようになったプログラマブルシェーダーを使用することが NPR を実現するのに最も適した方法であると考えられるからである。

3.4.1 DirectX

DirectX[9]とは Microsoft 社が Windows プラットフォーム上でハードウェアデバイスに高速にアクセスするために開発したメカニズムの総称である。2D、3D グラフィックス処理を行う DirectX Graphics やサウンドや音楽を再生する DirectX Audio、ジョイスティックなどの入力デバイスを制御する DirectInput といったライブラリ郡からなる。現在は主にマルチメディアやゲームなどリアルタイムにグラフィックス描画や音声を再生するアプリケーションに使用されている。DirectX の機能としてデバイスを取得することは出来るが、ウィンドウの作成や表示といった命令は含まれておらず、Windows API を使用する必要がある。

3.4.2 VertexShader と PixelShader

DirectX8.0 には以前からの固定されたレンダリング処理を行うプロシージャルシェーダーに加えて、ここで紹介するプログラマブルシェーダーが追加された。プログラマブルシェーダーとはその名の通り、プログラミング可能なシェーディング処理である。パーソナル向けハードウェアでは nVidia 社の GeForce3 や ATI 社の RADEON8500 から対応するようになり、GPU において高速に処理することが可能になってきた。今までのレンダリングでは DirectX8.0 のプロシージャルシェーダーのように、開発側で用意されたシェーダーを使用することしか出来なかったが、そこに開発者がプログラムを解して自由にシェーディング処理を変更させることが出来るようになった。この考え方は 1987 年の論文で発表され、その後 Pixar の開発した RenderMan により実現されている。DirectX で実装されているプログラマブルシェーダーも RenderMan のプログラマブルシェーダーを模範として作られ

ている。1998 年の論文[10]ではハードウェア上でプログラマブルシェーダーを動作させる試みがなされ、その後ハードウェアにおけるプログラマブルシェーダーの実行に関する研究[11,12]を受け、一般の開発者が使用可能なプログラマブルシェーダーとして DirectX8.0 に実装されて公開された。

DirectX8.0 のプログラマブルシェーダーは VertexShader と PixelShader という 2 つによって実装された機能である。それぞれ VertexShader と PixelShader の具体的な処理は以下の通りである。

- VertexShader：座標変換や照明計算などの頂点データに対する処理をプログラム可能とする。
- PixelShader：テクスチャマッピングやテクスチャブレンディングなどのピクセルデータに対する処理をプログラム可能とする。

つまり VertexShader は図 6 の●で表されるポリゴンの頂点に対して、PixelShader は図 6 の○で表されるポリゴン内にあるピクセルに対して処理を行う機能のことである。

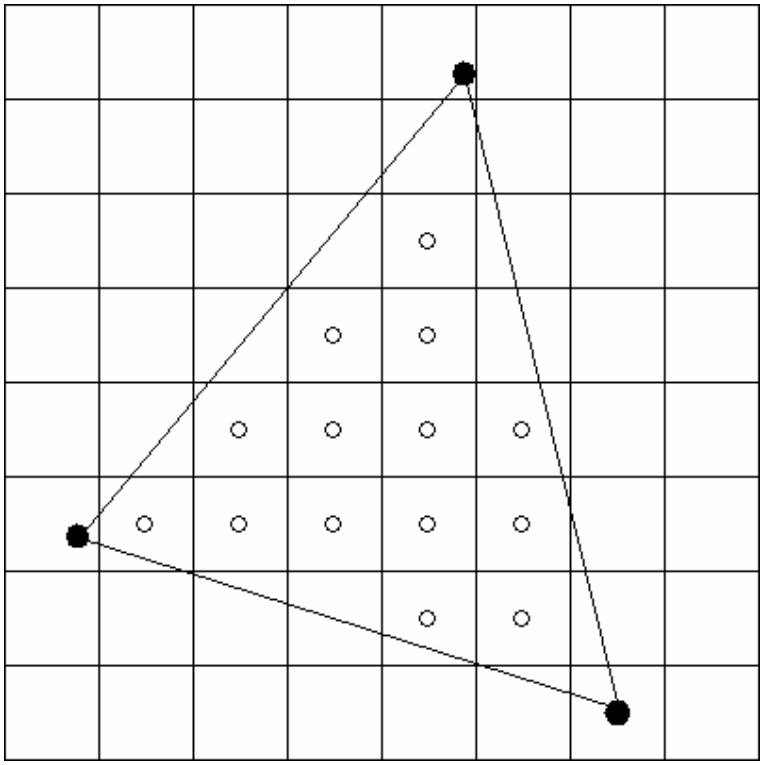

図６　VertexShader と PixelShader の実行個所

このことにより、従来ポリゴンの頂点ごとに行っていた照明計算をポリゴン内の１ピクセル単位で計算を行うことがより簡単に行うことが出来るようになった。具体的にはフォトリアリスティックレンダリングでポリゴン表面の細かな凹凸感を表現するために使われるバンプマップ処理なども、プログラマブルシェーディングを使うと簡単に実装することが出来るようになる。

第4章　NPR ツールキットの評価

本研究では NPR ツールキットの機能としてフィルター型 NPR のエンボスフィルターと幾何情報を用いた NPR のトゥーンレンダリングを実装している。このレンダリング処理を記述する部分から NPR ツールキットにおける開発の効率化に対する評価を行う。

まずエンボスフィルターによる NPR では、ツールキットを使わずにプログラムを開発した場合、以下の手順が必要になる。

・フレームバッファの大きさのテクスチャーを生成

・テクスチャーをレンダリングターゲットにする

・プロシージャルシェーダーを使ってレンダリングを行う

・プログラマブルシェーダーを使って、テクスチャーを元に NPR 画像を生成する

それぞれの項目について十数行のコーディングが必要になる。特にプログラマブルシェーダーを用いたフィルターの処理部分ではアセンブラによって図７のようなプログラムを書かなければならない。

```
vs.1.0

mov oPos, v0
add oT0, v7, c10
add oT1, v7, c11


ps.1.0

def c0, 0.299f, 0.587f, 0.114f, 0.0f
def c1, 0.4f, 0.4f, 0.4f, 0.0f
tex t0
tex t1
dp3 r0, t0, c0
dp3 r1, t1, c0
add r0, r0, -r1
```

図７ エンボスフィルターのアセンブラコード

だが、NPR ツールキットを使って開発を行う場合、

```
texture.createRenderTarget();

texture.setRenderTarget();

scene.render();

embosFilter(texture.getTexture(), 2.0, 2.0);
```

のように、各ステップにつき１つのメソッドを呼び出すだけである。エンボスフィルターに与えるパラメータはフィルターを掛けるテクスチャーと X 軸方向と Y 軸方向の移動距離という、最小限のパラメータを指定することにより図８のような効果を得ることが出来る。

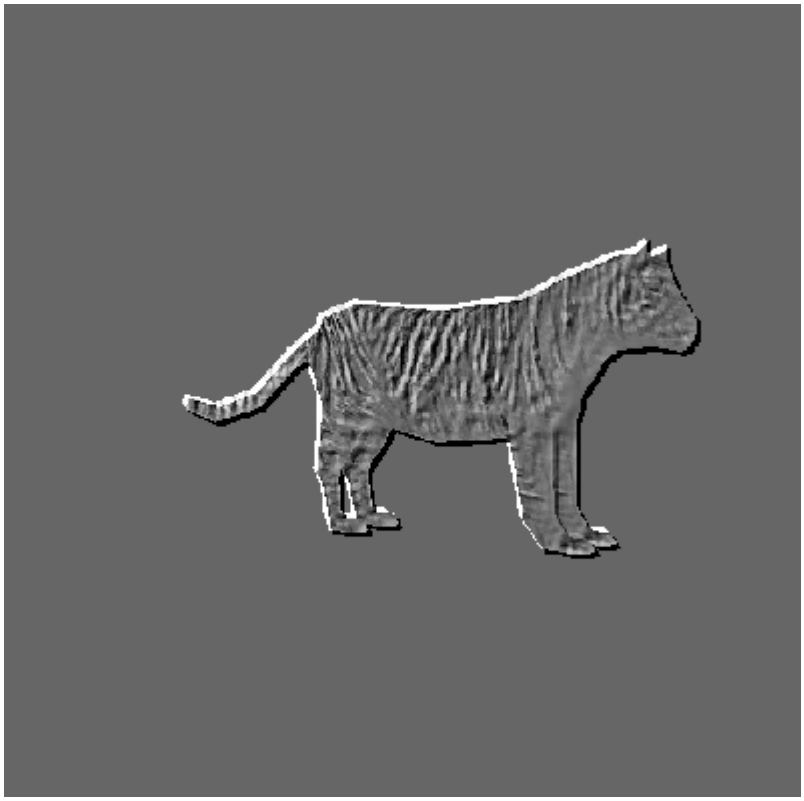

図８ フィルターを使ったエンボスフィルター

トゥーンシェーダーではプログラマブルシェーダーによって２段階のプロセスが必要になる。

・頂点を法線方向に押し出し、面を反転させてレンダリング

・ライトベクトルとの内積によって１次元テクスチャー座標を取り、テクスチャーカラーと頂点カラーを乗算する

この処理も NPR ツールキットでは１プロセスで実装することが可能である。

```
toonRender(scene, color, 5.0);
```

トゥーンレンダークラスに対して、形状とライトを保持するシーンクラス、縁取りの色情報、縁取りの押し出し量という、必要最低限のパラメータを指定することにより図９のようなトゥーンレンダリングを実装することが可能となる。

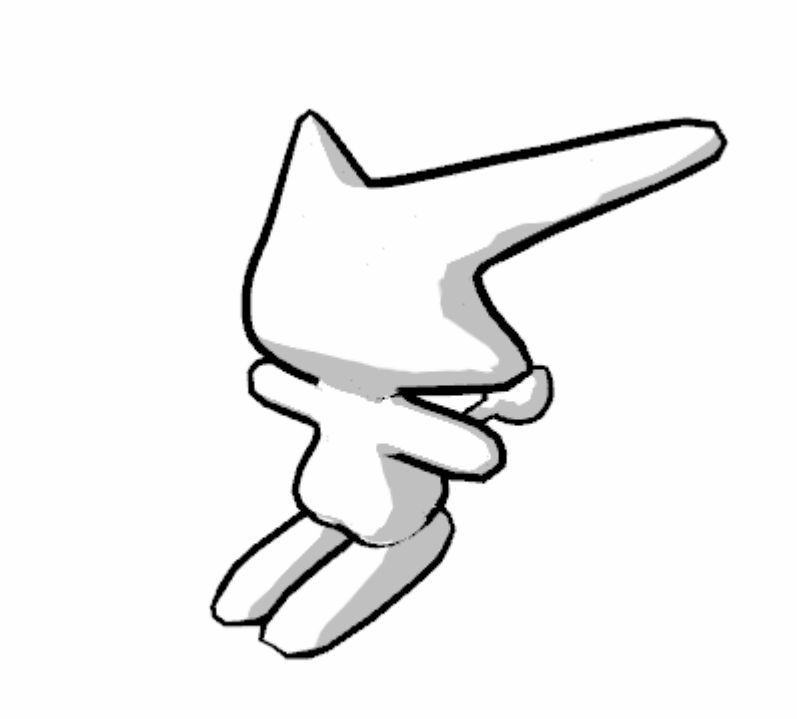

図９　幾何情報を使ったトゥーンレンダリング

本研究で開発を行った NPR ツールキットを使用して、NPR を使ったアプリケーションの開発を行うことにより特にレンダリング部分のプログラムを見ただけでも、NPR を使うために必要なのはフィルターもしくはレンダークラスの定義と必要最低限のパラメータを入力するだけである。その他に必要な機能は内部でカプセル化してあるため、開発者が意識する必要は無くスムーズに NPR を使用することが出来る。このことよりツールキットを

用いた NPR の実装により、NPR を使ったアプリケーションの開発の敷居を大幅に下げることが出来たといえる。

NPR ツールキットを用いたアプリケーションの開発効率の向上については、レンダリング処理部分の行数を見ても NPR ツールキットを使用して開発することにより約１０分の１程度に減らすことが出来るようになっている。また NPR ツールキットを用いてアプリケーションを開発することにより、NPR シェーダーの変更による大幅なソースコードの改変は必要が無くなる。シェーダーの変更で必要となるのは別のフィルタークラスまたはレンダークラスの定義と、ウィンドウクラスへの登録だけである。そのためいくつもの NPR シェーダーの効果を最低限の作業によって確認することが出来る。この点から考えてもツールキットを用いた場合の開発効率についても十分効果が得られると考えられる。

第5章　今後の展望

今回の研究ではエンボスフィルターとトゥーンレンダリングという簡単なNPRによる評価であったが、それでも十分に開発効率を上げられることが分かった。

技術が急速に進歩しているグラフィックスハードウェアではプログラマブルシェーダーの高速処理と高機能化について特に力が注がれている。またコア３DAPI においても、DirectXではバージョン9.0でプログラマブルシェーダーにおける処理機能がさらに追加され、OpenGL でもバージョン 2.0 の策定においてプログラマブルシェーダーの実装が予定されている。このようにプログラマブルシェーダーは現在リアルタイム 3DCG において最も注目されている分野である。そのプログラマブルシェーダーを使って実現される NPR についても今後多くの手法の研究と実装が行われていくであろう。

しかしそのため多くの人がアセンブラとNPRのアルゴリズムに置いて敷居の高さを感じるであろう。そのためツールキットにおける NPR の実装は今後ますます有効な手法になっていくことが考えられる。

謝辞

本研究を行うにあたり、多大なる助言を与えてくださった担当教官である渡辺 大地講師、副査を勤めてくださった若林 尚樹助教授及び、研究室において様々な面でアドバイスをしてくれた３DCG アプリケーション構築プロジェクトのメンバーに感謝の意を表する。

参考文献

[1] 斎藤隆文, 橋時市郎, “Comprehensible Rendering of 3D Shapes”, SIGGRAPH, 1990

[2] Paul Haeberli, “Paint By Numbers”, SIGGRAPH, 1990

[3] Pat Hanrahan, “Direct WYSIWYG Painting and Texturing on 3D Shapes”, SIGGRAPH, 1990

[4] Markosian, “Real-Time Nonphotorealistic Rendering”, SIGGRAPH, 1997

[5] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, Richard Riesenfeld, “Interactive technical illustration”, ACM Symposium on Interactive 3D Graphics, 1999, 31–38.

[6] Emil Praun, Matthew Webb, Adam Finkelstein, Hugues Hoppe, “Real-Time Hatching”, SIGGRAPH, 2001

[7] Adam Lake, Carl Marshall, Mark Harris, Marc Blackstein, “Stylized Rendering Techniques For Scalable Real-Time 3D Animation“, SIGGRAPH, 2000

[8] NVIDIA Developer Home, http://developer.nvidia.com/

[9] DirectX Developer Center, http://www.microsoft.com/japan/msdn/directx/

[10] Marc Olano, Anselmo Lastra, “A Shading Language on Graphics Hardware: The PixelFlow Shading System“, SIGGRAPH, 1998

[11] Mark S. Peercy, Marc Olano, John Airey, P. Jeffrey Ungar, “Interactive Multi-Pass Programmable Shading“, SIGGRAPH, 2000

[12] Kekoa Proudfoot, William R. Mark, Svetoslav Tzvetkov, Pat Hanrahan, “A Real-Time Procedural Shading System for Programmable Graphics Hardware”, SIGGRAPH, 2001