修士論文　　平成 17 年度 (2005)

3DCG コンテンツ開発における
オブジェクトプロファイラに関する研究

東 京 工 科 大 学 大 学 院
バイオ・情報メディア研究科メディアサイエンス専攻

竹内 亮太

修士論文　　平成 17 年度 (2005)

3DCG コンテンツ開発における
オブジェクトプロファイラに関する研究

指導教員　金子 満 教授，渡辺 大地 講師

東 京 工 科 大 学 大 学 院
バイオ・情報メディア研究科メディアサイエンス専攻

竹内 亮太

論 文 の 要 旨

論文題目	3DCG コンテンツ開発における オブジェクトプロファイラに関する研究
執筆者氏名	竹内 亮太
指導教員	金子 満 教授，渡辺 大地 講師
キーワード	リアルタイム 3DCG　オブジェクト指向　クラスライブラリ フレームワーク　開発環境　プロファイラ

[**要旨**]

近年、パーソナルコンピュータやゲーム機において、3 次元コンピュータグラフィクス (以下 3DCG) を利用した数多くのゲームコンテンツが開発されており、その開発環境の整備が重要な課題となっている。

3DCG では表示する対象、内容をオブジェクト単位で管理し、複数のオブジェクトを制御する必要がある。表示する形状、色、位置といったオブジェクトの表示機能そのものについては、既存のライブラリによって構造化が進められ、扱いは比較的容易になっている。一方、変形や移動のようなオブジェクトの挙動については、個々のオブジェクトに対する逐次的な命令の発行で個別に制御されており、情報として管理し、構造化する手段はこれまであまり考慮されてこなかった。よって、複数のオブジェクトが非同期的に挙動した場合、状況把握が困難なものになってしまい、望ましい挙動を得るために、非常に効率の悪い試行錯誤を余儀なくされている。

本研究では、既存の 3DCG 用クラスライブラリを用いて作成したプログラムの挙動を、オブジェクトへの操作履歴を取得し、管理することで、個々のオブジェクトの状態を巻き戻したり、条件を変化させて再生するなどの制御を可能にする機能を提案する。この機能をオブジェクトプロファイラと呼ぶ。オブジェクトプロファイラ機能を用いることで、ライブラリのユーザが作成したプログラムの挙動を容易に分析することができ、望ましい挙動を得るための試行錯誤や、不具合の原因解析などが容易に実現できる開発環境を提供する。本論文では、実際にオブジェクトプロファイラを実装したライブラリを作成し、開発者に提供することでその有用性を検証した。

A b s t r a c t

Title	Reserch on a object profiler in 3DCG contents programming
Author	Ryota Takeuchi
Advisor	Professor Mitsuru Kaneko, Lecturer Taichi Watanabe
Key Words	Realtime 3DCG, Object Oriented, Class Library, Framework, Development Evironment, Profiler

[**summary**]

In recent years, a lot of computer games using three dimension computer graphics (3DCG) are developed, and the construction of the development environment becomes an important problem in the personal computer and the game machine.

In order to display contents in 3DCG, it is necessary to manage several information of many objects such as shape, color, transformation and movement. It became easy to manage visual information (ex. shape, color and position) by structurize on some existing library. But managing the behaviors of the object is still complicated. Each of these must be individually controlled by a each command for individual object. On the other hand, structurize of behavior information had not been considered over the situation to date. Therefore, in cases where multiple objects behave asynchronously, it is difficult to understand their behaviors. For this reason, a very inefficient trial and error is necessity for making desired behaviors.

In this study, a managing system for the behavior history if the object is presented. This system allows to back wind and re-act the behaviors of the objects. The object condition can be changed in this operation. This system is called the object profiler. By using the object profiler, the behavior if the program becomes easily analyzed, the root cause analysis of unforeseen result become easily understandable and the trial and error to make desired behavior become easy. In this thesis, the utility of the object profiler was verified by making the library with which the object profiler is equipped and offering it to be developer.

目 次

図 目 次

表 目 次

第 1 章

はじめに

1.1 研究の背景と目的

近年ハードウェアの処理能力向上と共に、3 次元コンピュータグラフィクス (以下 3DCG) は我々にとってより身近なものとなり、至る所で目にするようになった。これに伴い、パーソナルコンピュータやゲーム機上で動作する、リアルタイム 3DCG コンテンツが盛んに開発されており、その開発環境の整備は重要な課題となっている。

リアルタイム 3DCG プログラミングは、高速な処理が要求される性質上、グラフィクスハードウェアを直接制御して処理を行う必要がある。しかし、ハードウェアを制御するプログラムコードは、一般のプログラマが最初から記述するのは困難な上、仕様の異なるハードウェアごとに異なる処理が必要になるため、生産性が非常に悪くなる。そのため現在では、不特定のグラフィクスハードウェアを統一的に扱うことのできる、アプリケーションプログラミングインタフェース (以下 API) を利用するのが一般的である。API とは、特定の目的のプログラムを作成する際に必要となる命令の集まりのことで、ハードウェアやプラットフォームのベンダーが、開発者に向けて公開していることが多い。

しかしベンダーが提供する 3DCG 用の API [1] [2] は、リアルタイム 3DCG におけるポリゴンやテクスチャなどの基本的な表示要素の描画機能を提供するものでしかなく、実際のコンテンツ開発に用いるにあたっては、十分な生産性を確保することが難しい。このため、ベンダーが提供する API を組み合わせて、より高レイヤーの機能をライブラリとして構成し、そのライブラリの API を用いて開発を行うことで、生産性の向上を図る開発手法が増えてきている。このように、特定の API を利用して用途に合わせた機能の構造化を行うことをラッピングと呼び、それを実現するためのクラスやメソッドの集合を、ラッパーライブラリと呼ぶ。本論文中では、単にライブラリと呼称した場合は、このラッパーライブラリのことを指すものとする。また、ライブラリを提供するだけではなく、開発するコンテンツプログラム上で扱う 3 次元形状モデルやテクスチャ、サウンドなどのデータ

の処理を、外部のコンテンツ作成ソフト [3] [4] と連携して管理するツールなども包含した、統合的な開発支援環境 [5] [6] も存在する。このような環境は、主に家庭用ゲーム機をターゲットとした開発を行うソフトベンダーに対して、製品として提供されている。

現在、ゲームコンテンツの開発は、ライブラリの整備が進んでいることにより、これらを積極的に利用して行う事例が多くなっている。しかし、近年のコンテンツの大規模化、複雑化により、ライブラリによる機能の構造化だけでは、開発の効率を向上させることが難しくなってきた。それは、これまでのライブラリが機能を命令として提供するだけで、命令の呼び出しを管理する責任がライブラリのユーザに任されていたことに起因する。つまり、ライブラリはユーザの作成したプログラムに沿って逐次処理を行うだけで、プログラムからどのような命令が、どのような順番で呼ばれたかは、ライブラリのユーザが自ら解析しなければ把握できない状況にある。このため、ライブラリの高機能化や、開発するコンテンツの複雑化によって、ライブラリのユーザが把握しなければならない情報量が増加の一途を辿っており、ライブラリの設計や開発手法の見直しの必要が生じている。

3DCG プログラミングに特有の問題として、ライブラリの設計段階で高レイヤーな機能を API として提供していたとしても、その API を利用して記述したプログラムが、実際の画面上でどのような挙動を示すかを類推することが難しいという点が挙げられる。単純な数値計算を目的としたプログラムの場合は、計算過程の数値をデバッガで追跡するなどの手法によって挙動を解析することができるが、3DCG における画面上での挙動を詳細に分析していくためには、プログラムによって制御している挙動を実際に反復して実行し、確認するしか手段がない。このため、意図した通りの挙動を得るために過度の試行錯誤が必要となり、この作業が開発の効率を著しく低下させる要因となっている。

本来であれば 3DCG コンテンツの開発環境下では、上記のような挙動の分析をスムースに行うための枠組みが用意されていることが望ましい。しかし昨今の 3DCG 技術は、新しい表現技術の開拓に傾倒しがちで、開発効率化のような問題

は軽視される傾向にあった。これまでのライブラリの整備は、新しい技術を機能として提供することに注力して進められていたため、動作の解析を行うための作業やコーディングは、ライブラリのユーザ自身が行わなければならない。このように、ライブラリを高機能に拡張していくだけでは開発効率は向上せず、むしろ膨大な機能を管理するために、開発者は多大な労力を割く羽目に陥っているのが現状である。

本研究では、3DCG コンテンツの開発を目的としたライブラリを設計する際に、ライブラリが提供する機能をオブジェクトとして構造化することに留まらず、そのオブジェクトの挙動自体を情報として管理し、構造化する手法を提案する。この手法は、挙動情報をライブラリ側で把握し、管理することにより、ライブラリのユーザが負う命令の管理責任を緩和して、動作の解析とその結果のプログラムへのフィードバックをスムースに行える環境を構築し、過度な試行錯誤を削減することが目的である。具体的には、各オブジェクトに対して実行した処理内容の履歴を取得しておくことで、履歴に沿ってオブジェクトの状況を巻き戻して確認する機能や、履歴上の処理内容を変更して再生する機能などの、インタラクティブな制御を可能にするものである。

このような処理は、リアルタイムではないプリレンダリングの 3DCG アニメーション製作ツールでは、タイムライン編集という機能で実現している。しかし、リアルタイム 3DCG プログラミングの場合は、そもそもタイムラインという概念自体が存在しない。何故なら、リアルタイムコンテンツにおけるオブジェクトの挙動は、時間軸に沿ってあらかじめ決定できるものではなく、様々な要素の計算、具体的にゲームコンテンツで言うならば、プレイヤーの操作や物理計算、AI の処理などを経て決定した結果だからである。このため、プリレンダリングのアニメーションのように、ある時間を指定すればその時点での状態が復元できるものではない。

既存のライブラリの設計には、前述したような動作確認処理を内包するという視点が欠けていた。そのため、オブジェクトの動作をテストするためには冗長な

コーディングによって、オブジェクトの状態を保持するための情報を収集し、管理する必要があった。しかし、このようなアプローチによる動作テストは、多大な労力が必要となる手法である。この機構をあらかじめライブラリ側で備えて、特殊なコーディングを必要とせずに使えるようにすることで、非常に有用な開発環境が提供できる。

1.2 本論文の構成

本論文の構成は以下の通りである。次の第 2 章では、本研究において議論の対象とする 3DCG コンテンツ開発環境の概観をレビューするとともに、現状の開発環境に対する本研究の位置付けを述べる。第 3 章で、本研究において提案するオブジェクトプロファイラの概念と設計について解説する。第 4 章では、オブジェクトプロファイラの機能を実質的に制御するインターフェースの仕様について解説する。第 5 章では、本研究で開発したオブジェクトプロファイラを実際に運用し、その成果の検証と考察を行う。第 6 章では本研究の成果と意義をまとめて、今後の展望について述べる。

第 2 章

3DCG コンテンツ開発環境の現状

本章では、既存の 3DCG コンテンツ開発環境を階層的に分類し、その概観をレビューする。更に、現在プログラミングを行う際に直面している問題点を整理し、本研究がその問題点を解決するにあたって定めた方針を提示する。

2.1 現在提供されている開発環境

本節では、3DCG プログラミングにおいて主に利用されている 2 つの API について解説し、その用途と特徴を紹介する。またこれらの API を利用して、コンテンツ作成により直結したコーディングが可能なライブラリ、ミドルウェアについても紹介する。

2.1.1 DirectX

DirectX [1] は Microsoft 社が提供する API で、Windows 環境上での統一的なマルチメディア処理を実現する。3DCG を取り扱う Direct3D をはじめ、サウンドや入力デバイス、ネットワーク処理などの、ゲームに関わる機能を包括的にサポートする。現在は Windows 環境だけではなく、XBox [7] などの Windows ベースのゲーム機でも用いられている。

DirectX の設計理念は、常に最新のハードウェアの機能を利用できる環境を提供することにあり、ハードウェアのパラダイムシフトにあわせて仕様の再設計がたびたび行われている。DirectX の API はクラスコンポーネントとして構造化されて提供されているが、あくまでハードウェアの仕様を優先して機能がまとめられているため、バージョンごとの再設計によってコンポーネントの構成が大きく変化する傾向にある。

2.1.2 OpenGL

OpenGL [2] は、元々は Sillicon Graphics 社 (SGI) がワークステーション向けに開発した内部開発環境をベースとし、プラットフォームやハードウェアに依存

しない、標準化されたグラフィクスライブラリとして開発が進められてきた API である。その出自により、マルチプラットフォームへの対応や、3 次元の幾何要素計算における精度の高さを強みとして、業務用 CAD のアプリケーション [8] や、ワークステーション上で実現する大規模なバーチャルリアリティシステム [9]、学術的な研究用途 [10] などで用いられることが多い。また 3DCG を扱うゲーム機の開発環境においては、OpenGL が提供している API をリファレンスとして、開発環境を提供している事例 [11] も存在する。

DirectX とは違い、グラフィクス処理に特化したライブラリではあるが、提供されているコマンド (関数) 群は基本的に環境に依存することなく扱えるため、マルチプラットフォームをターゲットとした開発に適している。また、仕様の策定を nVIDIA, ATI, Apple, INTEL などの主要ベンダーが多数参加している ARB(Architective Review Board) が取り仕切っており、新しい仕様が追加されても全てのプラットフォームで利用できることを保証するだけでなく、既存のコードの動作も保証されているのも特徴である。しかし、あくまでコマンド群を提供するだけの API であるため、DirectX よりも直接的なレベルの機能提供に留まっている。

2.1.3 ライブラリとミドルウェア

以上で紹介した 2 つの API は、いずれもリアルタイム 3DCG におけるポリゴンやテクスチャなどの基本的な表示機能を提供するものであるが、コンテンツ開発において、これらの機能を個々に直接利用するプログラミングを行うことは、生産性の観点から見て好ましい方法とは言えない。例えば、複雑な構造を持つ物体を表示する場合に、個々のポリゴンメッシュを描画する命令をその都度記述するようなプログラミングは、再利用性が低く、非効率的な開発の進め方である。実際にコンテンツを開発する際には、基本的な表示機能を組み合わせることで、開発の目的に即した一連の処理を構成し、再利用しやすいように機能を構造化しておくことが望ましい。このように、基本的な機能を持つ API を利用して、より高度で直感的な機能を構成する新たな API を作成し、用途に合わせた機能の構造化を

行うことをラッピングと呼び、これを実現するための処理をラッパーと呼ぶ。ラッパーは、新たに構造化した機能をメソッド (関数) やクラスの集合とすることによって、ライブラリとして提供される。

PC 環境上で 3DCG プログラミングを行うにあたっては、DirectX をベースとしたライブラリ [12][13][14] と、OpenGL をベースとしたライブラリ [15][16][17] がそれぞれ存在するほか、両者を統合してラッピングし、必要に応じてベースとする API を切り替え可能なもの [18] もある。ベースとする API の選択によって、使用可能なライブラリやプラットフォームが変化することになるが、PC 環境をターゲットとするゲームコンテンツ開発の場合は、DirectX をベースとして利用することが多い。これは、Windows 環境以外での PC 環境におけるエンターテイメント用途の需要が少なかったという点と、最新のグラフィクスハードウェアを利用する際には、設計方針の関係から DirectX をベースにした方が都合が良かったという事情がある。それに対して OpenGL の仕様は、これまでグラフィクスハードウェアの進歩に水をあけられがちであったが、Cg 言語や GLSL などのシェーダ言語が OpenGL 上から利用できるようになったことで、ハードウェアを柔軟に制御可能な環境が整いつつある。PlayStation3 [19] の開発環境が正式に OpenGL ベースとなった現状も踏まえると、今後は PC 環境上での開発物が容易に移植できることが期待できるため、OpenGL のエンターテイメントにおける需要は高まることが予想できる。

大規模なコンテンツ開発においては、プログラミングの環境を整備するだけではなく、プログラムで扱うモデルやテクスチャ、サウンドなどのデータの処理が大きな手間となるため、その管理が重要な問題となる。ゲームで扱うこれらのデータの処理工程をコンテンツパイプラインと呼ぶが、ライブラリだけではなく、コンテンツパイプラインを管理する支援ツールを含んだ開発環境を、まとめて提供している事例も存在する。このような統合的な開発環境をミドルウェアと呼ぶ。特にゲーム機をターゲットとした開発を行う場合、3DCG に関する以外の処理を実現する際にも、ハードウェアの仕様に依存した制御が大量に発生するため、それら

を自力で処理することは生産性を低下させる要因となる。このためゲーム機上での開発においては、ミドルウェアを利用することがほぼ必須の条件となっている。

ミドルウェアは商業製品として企業向けに販売されているものが多く、ゲーム機上で採用が多いものとして RenderWare [5] や CRI ミドルウェア [6] などが挙げられる。またミドルウェアを業務上ではなく、開発者の育成において利用できるようにする目的で Lamp [20] が開発されており、かなり実践的なコンテンツ開発が体験できる環境が整備されてきている。更には、多少乱暴な言い方にはなるかもしれないが、ツクールシリーズ [21] も広義な意味でのミドルウェアの要件を満たしており、ゲームにおける RAD(Rapid Application Development) ツールとしての意義を確立している。

2.2 開発環境における階層的なフレームワーク

本論文ではゲーム開発環境を議論するにあたり、前節で解説したそれぞれの環境を、以下の 4 階層からなるフレームワークに分類する。

- Level1：プリミティブな表示機能
- Level2：オブジェクトとして機能を構造化したコンポーネント
- Level3：作成するコンテンツの目的に即したライブラリ＆ルーチン
- Level4：コンテンツ上の演出的な動作、フローを規定するスクリプト

Level1 はポリゴンメッシュやテクスチャマッピングなど、グラフィクスハードウェアに直結した、基本的な描画機能が該当するレイヤーである。Level2 は Level1 の機能をラッピングし、3 次元形状データに対する姿勢制御や変形などの柔軟な制御、複雑に交錯する座標系やシーンに表示するモデルリストの管理など、構造化した機能を提供するレイヤーとする。Level3 は Level2 によって提供される機能を用いて、開発するコンテンツのシステム的な動作を構築するレイヤーである。例えばロールプレイングゲーム (RPG) の場合、マップの表示やキャラクターの移動処

理、戦闘シーンやエフェクトの処理などを実際に行う部分が該当する。Level4 は、Level3 で構築した処理を制御するためのスクリプトが該当する。再び RPG を例にして述べるが、イベントシーンにおいて表示するメッセージやサウンドの制御、Level3 で作成したエフェクトの呼び出しなどは、プログラム上に直接記述するのではなく、データとして外部から読み込んだスクリプトに沿って処理するのが一般的であり、コンテンツの形態によっては、非常に大きなウェイトを占めるレイヤーとなる。

Level1 に相当するのが OpenGL や DirectX であるが、DirectX はクラスコンポーネントとしての機能提供が行われており、本節で示した階層構造においては Level1.5 とでも言うべき位置づけにあるとも言える。Level2 に相当するのがラッパーライブラリであるが、実際には Level2 と Level3 の概念を区別せずに進めてしまう開発方式がとられていることが多い。これは、使用する既存のライブラリの選択次第では、Level3 の段階で実現できる事柄に制約が生じてしまうため、結果的には Level3 の開発の度に Level2 に相当する部分から開発するケースが多いためである。

ミドルウェアは、Level2 までの要素に加えて、ステレオタイプなコンテンツにおける Level3 に該当する機能を提供したり、コンテンツパイプラインを管理するツールをパッケージした統合的な開発環境である。ミドルウェアの多くは、Maya [3] や 3ds Max [4] などのコンテンツクリエイションツールとのリンクを最大のセールスポイントに据えており、プログラマが内部向けのツールを作成する手間を軽減できる点が、大きなメリットである。しかし、中途半端に Level3 の階層が整備されている関係上、コンテンツ部分の開発においては各ミドルウェア特有の制約が生じがちで、ライブラリ上で開発を行うケースに比べるとその自由度は下がる。

ツクールシリーズは、Level3 までの要素を完全に隠蔽し、Level4 に相当する部分を編集する環境をユーザに提供することで、最低限の労力でステレオタイプなゲームコンテンツを作成できる開発環境として、主に趣味でゲーム制作を楽しむアマチュアのユーザに親しまれている。しかし、あくまでゲーム開発の疑似体験

ができる環境としての意味合いは払拭出来ず、その自由度は高くない。近年のシリーズでは、Level3 に相当する部分をユーザが編集可能なスクリプトとして提供し、大胆なシステムのカスタマイズが可能なものも存在するが、処理速度などに難がある場合が多く、ツクールが元々想定している形態のコンテンツから大きく逸脱したものを作成することは難しい。

既存の開発環境を利用する際には、その環境が実現する機能がどの階層に該当するかを把握し、的確に運用する必要がある。高レベルな機能を提供する環境では、実現できる仕様の自由度は低くなり、低レベルな環境から機能を積み上げて構築するには、膨大な量の作業が必要になる。また低レベルな環境から開発していく場合には、コンポーネント階層とコンテンツそのものを構成する階層との機能的な分離を行うことによって、再利用性を高めることが重要である。

2.3 開発環境を利用する上での問題点

前節で述べたように、ゲームコンテンツの開発環境は様々な形態で提供されているが、それぞれの環境が目的としているのは、開発に必要な機能を効率的に構造化することにある。機能の構造化とは、その環境が該当する階層以下の低レベルな命令を組み合わせて、目的に即した処理を構成し、低レベルな命令を隠蔽した上で、環境の利用者に提供することで達成される。この流れはゲームコンテンツ開発に限ったものではなく、一般的なプログラミングにおいても推し進められてきた流れである [22]。

3DCG コンテンツの開発環境の中でも、特に Level2 に該当するラッパーライブラリにおける機能の構造化は、ライブラリを用いたコンテンツ開発が事例として多いことも相まって、非常に重要視される要素である。実際に、現在クラスライブラリとして提供されている環境においては、表示する形状、色、位置などの表示機能そのものについては、クラスオブジェクトとしての構造化が進み、扱いは比較的容易になっている。しかし近年のコンテンツの大規模化と、3DCG の表現技術自体が高度に進化し、複雑になってきたことによって、ライブラリによる表

示機能の構造化だけでは、開発の効率を向上させることが難しくなってきた。これは、既存のライブラリが機能を命令として提供するだけで、命令の呼び出しを管理する責任がライブラリのユーザに任されていたことに起因する。つまり、ライブラリはユーザの作成したプログラムに沿って逐次処理を行うだけで、プログラムからどのような命令が、どのような順番で呼ばれたかは、ユーザが自ら把握し、管理しなければならない状況にある。

特に 3DCG のプログラミングでは、モーションなどのアニメーションを表現するために、一連の動作の過程が実行結果として重要になるケースが非常に多いため、ライブラリの命令呼び出しを慎重に管理しなければならない。例えば、物体がある地点から特定の軌跡を描いて移動する様子を表現しようとした場合、物体が最終的に到達する地点は容易に与えることができるが、実際には 1 フレームごとに自然な軌跡を描くような処理を実現する必要がある。しかし、1 フレームごとに実行した処理が意図した通りの挙動を得るように処理を記述するには、多大な試行錯誤と熟練が必要になる。更に、このように分割した挙動を複数の物体に与える制御を行う場合は、ある時点で物体同士が意図しないような位置関係に陥ったり、画面に表示されていない部分での位置関係が把握出来なくなることが多い上、意図とは異なる挙動が生じた場合に、原因となっている制御の切り分けが困難になるなどの問題を招くことが多い。

2.4 既存の動作解析手法の問題点

前述したような状況を打破するには、プログラムの動作を解析する作業が必要となる。一般的なプログラミングにおいて、プログラムの動作を解析する場合にはデバッガツールを用いることが多い。3DCG プログラミングにおいても、デバッガは強力なツールであることに変わりはないが、3DCG として表示している物体の挙動を追跡していく上では、煩雑な作業が必要となる。何故なら物体の挙動は、プログラム全体の流れの中で、断片的に呼び出している一部の命令が積み重なった結果であるため、プログラムそのもの流れを追っていても、表示している物体を

抽象化しているオブジェクトの状況の推移を把握するのは困難だからである。またデバッガツールは、コンピュータのアーキテクチャの視点から動作を解析するため、3DCG を扱う目的で抽象化したプログラム上の記述と実際の表示結果との関連性が薄く、実際の挙動が推測しづらい点も問題となる。このため、物体の挙動に注目した動作解析を行う場合は、デバッガ上で特定の命令のみをマークする作業や、プログラムソース自体に動作検証用のコードを盛り込む必要があり、非効率的な試行錯誤が開発者に要求されている。

2.5 先行研究と本研究の提案手法の比較検討

既存の 3DCG ライブラリに関する研究においては、プログラマに要求する記述を可能な限り抽象化することで、目的に直結したコーディング環境を構築する試みがなされている。具体例として、物理シミュレーションに特化した言語体系を持つ Act [23] や、プログラムの内容自体を 3DCG 上のオブジェクトとして扱い、ビジュアルなプログラミング環境の提供を目的とした SAM(Solid Agents in Motion) [24] などがある。しかし、これらの環境はオブジェクトの制御方法や表示方法を、抽象的に表現することを目的とするもので、制御の結果として挙動が複雑化する問題に対処しうるものではない。

このように、ライブラリの整備のみによるプログラミングの効率化には限界があるため、3DCG アニメーションのプログラミングをグラフィカルユーザインタフェース (GUI) によるビジュアルツールを用いて支援する研究がなされている。代表的なものとしては、統合的なビジュアルプログラミング環境である Alice [25] [26] が挙げられるほか、Java 言語によるプログラミング環境と GUI を併用するシステムを提案している peaCAT [27] などがある。これらの研究は、ビジュアルツールを用いて動作をプレビューしながらモーションをプログラミングすることによって、メソッドの動作を直感的に把握しながら作業を進められるというメリットを提供する。しかし、このようなツールを用いた支援環境は柔軟性が低く、ツールがサポートする範囲を超えた複雑な挙動を記述する段階で使えなくなることが

多い。また、ビジュアルツールによるプログラムコードの生成や制御は、実現できる機能に大幅な制約が課せられるだけでなく、生成されたコードが必ずしも可読性が高いとは言えず、編集に耐えうるものにならない点も大きな問題となる。ツールによってコードを管理するシステム上では、本来自由な記述が許されるはずのプログラミングという行為に、大きな制約が発生することを留意しなければならない。

ここで本研究で提案するのが、記述されたコードが引き起こす挙動の構造化である。これは、ツールからコードを生成したり制御するのではない、これまでとは異なる視点からのアプローチによる支援手法として提案するものである。本研究における挙動の構造化とは、ライブラリが提供したオブジェクトに対して、ライブラリのユーザが作成したプログラムが行った操作を全て把握し、管理可能にする構造を用意することである。これにより、ライブラリはユーザプログラムからの操作をただ単に実行するだけではなく、以下のような機能をプログラマに提供できるようになる。

- オブジェクトに対して行った操作履歴の閲覧
- 操作の巻き戻し、再生による挙動の確認
- 操作履歴の編集による、条件を変化させた上での挙動の確認

これらの機能を利用できることで、プログラマがライブラリを利用する上で発生する、機能を管理する責任が大幅に軽減される。これまでライブラリのユーザが自力で把握する必要があった操作履歴を、あらかじめライブラリ側で把握し、制御可能にしているため、既に記述したプログラムに対する挙動の分析や、モーションプログラミングを行う際の調整作業、挙動の不具合を起こしている操作の発見、高速性を追求するための過剰な操作の排除など、様々な用途への応用が期待できる。

第 3 章

オブジェクトプロファイラの概要と設計理念

本章では、前章で提案した「挙動の構造化手法」に基づいて実装したオブジェクトプロファイラの概要と、その設計理念について述べる。本研究では実装例として、OpenGL ベースの 3DCG クラスライブラリである FK System [28] に対して、オブジェクトプロファイラの機構を組み込む拡張を行っており、組み込む対象としたオブジェクトの仕様と、組み込む際の詳細な設計方法についても言及する。

3.1 オブジェクトプロファイラの概要

本節では、ある機能が引き起こす挙動を構造化するにあたって、オブジェクトプロファイラとして付加する処理を抽象的なレベルで解説する。本研究においてオブジェクトプロファイラの適用対象とする機能は、オブジェクト指向の概念におけるクラスオブジェクトを基本単位として構成してあるものとする。すなわち、プロファイル対象とするオブジェクトは、各々がオブジェクト自身の状態と、その状態を操作するメソッドを保持していることになる。本研究が提案するオブジェクトプロファイラは、対象とするオブジェクトに対して以下の拡張を行う。

3.1.1 履歴の取得処理の付加

オブジェクトの挙動を把握するにあたっては、個々のオブジェクトが自身の状態に対してどのような操作が行われたかを記録しておく必要がある。そのため、個々のオブジェクトに対して、操作の履歴情報を記録するデータベースを追加する。この履歴データベースの内容を参照することで、そのオブジェクトの状態遷移を情報として把握することが可能となる。この履歴データベースに操作内容を記録する処理は、オブジェクトに対する操作を行うメソッドに対してあらかじめ追加しておくものとする。これにより、プログラム上でオブジェクトの操作メソッドを呼び出す記述を行うだけで、オブジェクトの操作と同時に履歴情報の記録を自動的に行うことが可能になる。このような設計を行うことで、オブジェクトの利用者が、履歴を取得するための冗長なコーディングを行う必要がなくなる。

3.1.2　履歴の巻き戻しと再生処理の実現

履歴を取得しただけでは、それは単なる操作の記録でしかない。オブジェクトプロファイラでは、オブジェクトの状態を履歴に基づいた巻き戻しや再生処理によって、任意の時点の状態を再現する機能を提供する。そのためにオブジェクトプロファイラは、履歴の取得対象とする全ての操作に対して、ある操作がオブジェクトに及ぼした影響を取り消す処理と、操作そのものを再現する処理を定義する。この処理を呼び出すことで、オブジェクトの状態を記録してある履歴の操作の単位で巻き戻し、あるいは再生することが可能となるため、任意の状態を再現することができる。

3.1.3　履歴の制御インタフェースの提供

履歴の取得、巻き戻しと再生処理を実現することによって、オブジェクトの内部処理としてのオブジェクトプロファイラ機能は完成する。しかし、これだけではオブジェクトの内部処理を呼び出すためのコーディングが必要になり、支援環境として十分な機能が提供できているとは言えない。本研究では、任意のオブジェクトに対する履歴の制御を行うための GUI を作成し、オブジェクトプロファイラを構成する機能の一つとして提供することとした。制御インタフェースの提供によって、画面上の表示とオブジェクトの挙動を直感的に対応づけて確認できるようになるほか、インタフェースの機能自体を拡張することで、取得した履歴に対して目的に応じた活用方法を考案することが可能になる。

3.2　プロファイル対象とするオブジェクトの概要

本研究では、前節で述べたオブジェクトプロファイラの概念を「モデル」を表すオブジェクトに対して適用することとした。本論文におけるモデルとは、3DCG コンテンツのプログラミングにおいて、表示処理を行う際の基本となる単位を指すものとする。モデルを表すオブジェクトは、3DCG シーンにおいて特定の物体

を表現するための形状と、シーン中における位置、姿勢、色を表現するマテリアルなどのステータスを持つものとして定義する。この定義は 3DCG プログラミングにおいては一般的なもので、多くのライブラリでもモデルオブジェクトを介したコーディングによって、低レベルな命令を逐次発行する必要のない、直感的な制御命令によるモーションやアニメーションのプログラミングが可能になっている。本研究で実装を行った FK System において、モデルの概念は fk_Model というクラスとして定義されている。FK System を用いた 3DCG プログラム中では、シーン内に表示する物体や視点を決定するカメラは全て、この fk_Model クラスのオブジェクトとして定義し、制御することになる。

モデルオブジェクトが提供する機能のなかでも、モデルの位置と姿勢の制御は、3 次元空間中における物体の見え方や挙動を決定する上で、非常に重要な要素である。3DCG におけるアニメーション表現は、その大半がこの位置と姿勢の制御によって成り立っていると言っても過言ではない。本研究で行った実装では、モデルに対する位置と姿勢及びスケールを制御する命令を、オブジェクトプロファイラの適用対象として選択した。これらの制御を 3 次元座標変換操作と呼ぶ。3 次元座標変換操作は、大別すると以下の 3 つの制御によって成り立っている。

- スケーリングの変更による拡大縮小制御
- 任意軸に対しての回転移動による姿勢制御
- 平行移動による位置制御

これらの制御は、通常の 3DCG API では難解な行列の演算が必要となる処理である。fk_Model クラスでは、方向ベクトルやオイラー角表現を取り入れた姿勢制御を提供し、すべての操作を直感的な名前のメソッドで制御可能にすることによって、難解な数学的演算を行う必要のない処理の記述が可能になっている。しかし前章で述べたように、プログラム上の記述と実際の挙動の対応は、実際にプログラムの動作を確認しなければ分かりにくいことには変わりがなく、動作の反復によ

る挙動の解析にまで配慮した設計には至っていない。そこで本研究では、fk_Model クラスに対する 3 次元座標変換操作の履歴を取得して、インタラクティブな制御環境を提供することにより、3DCG シーンの状況把握を補助する環境を構築した。

3.3 ヒストリーフック機構の設計

本節では、fk_Model に対して履歴を取得する処理を追加する際の設計について述べる。既存のクラスコンポーネントに対して機能を追加する場合、対象とするクラスを継承し、派生したクラスで追加する機能を実装するのがセオリーである。しかし、既存のライブラリに対してプロファイラ機能を付加する場合、ライブラリの使用者に冗長なコーディングを要求することは好ましくない。本研究では、プロファイラ機能を利用するにあたって可能な限りコードに対して変更を加えずに済むように、以下のような継承関係を用いた。

今回オブジェクトプロファイラ機能を付加するにあたり、ライブラリのユーザが利用するインターフェースは変更せずに、基底クラス→実装クラス→プロファイラクラス→インターフェースというように、インターフェースの前に追加する機能のクラスを挟んだ継承関係を構築した。このような継承関係を設計に用いることで、既存のライブラリのコードに対する変更を最小限に抑えると共に、利用者側は既存のライブラリと同じ利用方法のまま、プロファイラ機能を利用することが可能となる。以下の図 3.1 は、プロファイラを導入する際のクラスの継承関係を示す。左側が、既存のライブラリにおける 3 次元モデルを表すクラス図で、右側はプロファイラを導入したクラス図である。

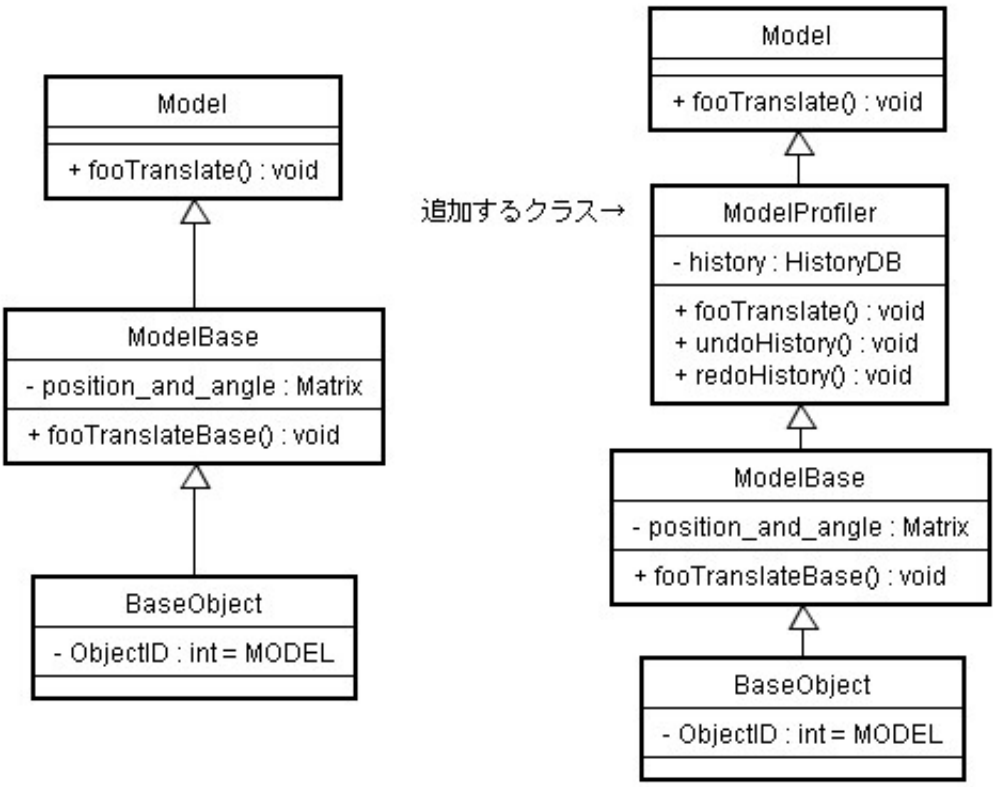

図 3.1: プロファイラを導入したクラスの継承関係

プロファイラクラスでは、ユーザプログラムが呼び出した命令の情報をデータベースに記録し、実際の処理を行う実装クラスのメソッドを呼び出している。これにより、ユーザはデータベースへの記録作業を意識することなく、モデルの制御を記述できることになる。このように命令の情報を取得する仕組みを、ヒストリーフック機構と呼ぶ。

3.4 コマンドベースによる履歴のデータベース設計

本節では、ユーザプログラムが行った処理の情報を記録する手法について述べる。3 次元座標変換操作のメソッドは、モデルの状態に対して変化量を与えるタイプと、座標や方向ベクトルを直接代入するタイプに大別することができる。このため、代入操作を行うメソッドの場合は、引数と共に書き換える前の値も保存

する必要がある。以下の表 3.1 は、実際に記録するメソッドの処理内容と引数の一覧を示す。実際のライブラリ上における仕様では、各々の処理をグローバル座標を基準に処理するバージョンと、モデルのローカル座標を基準に処理するバージョンを用意している。

表 3.1: 履歴情報として記録するメソッドの一覧

メソッドと引数	処理内容
Scale(スケール)	スケール代入
Rotate(座標, 軸, 回転角)	任意軸回転移動 (姿勢は不変)
RotateWithVec(座標, 軸, 回転角)	任意軸回転移動 (姿勢も変化)
Focus(位置ベクトル)	注視点の指定
Angle(オイラー角)	姿勢の直接代入
Translate(移動ベクトル)	平行移動
MoveTo(位置ベクトル)	座標の直接代入

履歴情報を記録するにあたっては、各メソッドに固有の ID を割り当てておき、前述したヒストリーフック機構によってメソッドの呼び出しを捕捉して、実行したメソッドの ID と引数、またメソッドによっては状態の復元に必要なパラメータを、配列に蓄積していく。記録するパラメータの種類は、FK System において 3 次元のベクトルを表現する fk_Vector 型の変数の他に、回転処理の角度などを保存する実数型 (double) や、整数型 (int) の数値にも対応している。その他、将来の機能拡張やライブラリの内部処理に対応するため、論理型 (bool) と文字列型 (string) の値も格納することが可能になっている。

履歴情報をデータベースとして管理するにあたっては、実行したメソッドの ID と、そのメソッドが受け取った引数を、まとめてコマンドオブジェクトとして定義し、これを配列として保存することで、任意の時点での操作を参照することが可能になった。このようにして記録した履歴を、コマンドヒストリーと呼ぶ。本研究で行う拡張により、全ての fk_Model クラスのオブジェクトが、各々のコマンドヒストリーを保持することになる。

例えば、fk_Model クラスのオブジェクトに対して、以下のような操作を記述し

たとする。ソース中のメソッド名は、グローバル座標を基準とする操作名には gl、ローカル座標を基準とする操作名には lo という接頭辞を付加している。また、fk_Y とは Y 軸を示す定数を表し、プログラム中では int 型の整数として扱われる。

```
fk_Model    testModel;

testModel.glMoveTo(0.0, 0.0, -10.0);
testModel.setScale(5.0);
testModel.glFocus(-1.0, 0.0, 0.0);

for(int i = 0; i < 5; i++) {
    testModel.loRotateWithVec(0.0, 0.0, 0.0, fk_Y, 0.05);
    testModel.glTranslate(0.0, -0.1, (double)i*0.2);
}
```

以下の表 3.2 は、上記のプログラムを実行した場合に、testModel のコマンドヒストリーとして記録する内容である。表中の getPos(), getAngle(), getScale() とは、モデルが現在保持している位置、姿勢、スケールを取得するメソッドを表し、代入操作を行う際に代入前の値を取得し、保存するために使用しているものである。

表 3.2: testModel のコマンドヒストリー

メソッド ID	保存してある引数
GLMOVETO	fk_Vector(0.0, 0.0, -10.0), getPos()
SETSCALE	double(5.0), getScale()
GLFOCUS	fk_Vector(-1.0, 0.0, 0.0), getAngle()
LOROTATEWV	fk_Vector(0.0, 0.0, 0.0), int(fk_Y), double(0.05)
GLTRANSLATE	fk_Vector(0.0, -0.1, 0.0)
LOROTATEWV	fk_Vector(0.0, 0.0, 0.0), int(fk_Y), double(0.05)
GLTRANSLATE	fk_Vector(0.0, -0.1, 0.2)
LOROTATEWV	fk_Vector(0.0, 0.0, 0.0), int(fk_Y), double(0.05)
GLTRANSLATE	fk_Vector(0.0, -0.1, 0.4)
LOROTATEWV	fk_Vector(0.0, 0.0, 0.0), int(fk_Y), double(0.05)
GLTRANSLATE	fk_Vector(0.0, -0.1, 0.6)
LOROTATEWV	fk_Vector(0.0, 0.0, 0.0), int(fk_Y), double(0.05)
GLTRANSLATE	fk_Vector(0.0, -0.1, 0.8)

3.5 逆操作の定義

前節で解説したコマンドヒストリーには、モデルに対して行った全ての操作が記録されているため、この履歴の昇順通りの操作を行うことで、一連の処理を再現することができる。それとは逆に、ある操作が及ぼした影響を無かったことにする操作を定義しておき、それを履歴の降順に実行していくことで、モデルの状態を操作単位で巻き戻すことが可能になる。この「ある操作の影響を取り消す操作」のことを逆操作と呼ぶ。コマンドベースによる逆操作は元々業務用 CAD の分野で用いられており [29]、可逆な操作の構造化に適しているため本研究でも採用した。本手法では、オブジェクトプロファイラの処理対象とした操作全てに対して逆操作を定義しており、コマンドとして管理している操作に関しては、履歴に沿って自由に処理を巻き戻したり、進めなおしたりすることが可能である。

実際の逆操作の定義方法について述べる。平行移動や回転移動処理の場合は、移動ベクトルや回転角の符号を反転させることで逆操作とする。スケールや方向ベ

クトルを代入する操作の場合は、あらかじめ保存してある代入前の値を復元することで逆操作とする。特に注視点を代入する Focus メソッドの場合、注視点を与えるだけではアップベクトルの値が不定になるので、あらかじめオイラー角による姿勢の状態を保存しておき、それを復元することで逆操作を実現している。以下の表 3.3 に、各メソッドに対する逆操作の定義例を示す。

表 3.3: 逆操作の定義例

メソッドと引数	逆操作の内容
Scale(スケール)	Scale(保存済みのスケール)
Rotate(座標, 軸, 回転角)	Rotate(座標, 軸, -回転角)
RotateWithVec(座標, 軸, 回転角)	RotateWithVec(座標, 軸, -回転角)
Focus(位置ベクトル)	Angle(保存済みのオイラー角)
Angle(オイラー角)	Angle(保存済みのオイラー角)
Translate(移動ベクトル)	Translate(-移動ベクトル)
MoveTo(位置ベクトル)	MoveTo(保存済みの位置ベクトル)

第 4 章

オブジェクトプロファイラの利用方法

本章では、オブジェクトプロファイラによって取得した履歴に対して、インタラクティブな制御機能を提供する GUI の仕様について解説する。この GUI をヒストリーブラウザと呼ぶ。また、ヒストリーブラウザを用いた履歴の制御を行うために、ユーザプログラム上で必要な作業についても述べる。

4.1 オブジェクトプロファイラ機能の制御

本研究では、既存のライブラリである FK System に対して拡張を行う形で実装を行った。このため FK System を利用して開発したプログラムは、ライブラリのファイルセットを拡張を施したものに差し替えることで、即座にオブジェクトプロファイラ機能を用いることができる。ただし、強制的にオブジェクトプロファイラを有効にしてしまうと、動作のパフォーマンスが低下する恐れがあるため、標準の動作としてはオブジェクトプロファイラ機能は有効にせず、機能の有効化と無効化を制御するメソッドを提供することとした。既存のプログラムに対してオブジェクトプロファイラを有効にする場合は、ヒストリーブラウザを利用するための制御メソッドと、各 fk_Model オブジェクトに対して履歴の取得を開始する制御メソッドの呼び出しを記述する必要がある。また本研究での実装においては、取得する履歴に対して任意の名称を付加することで、プロファイル対象としたメソッドの呼び出し元を識別するための制御メソッドも提供している。本節では以上に挙げたような、オブジェクトプロファイラを利用する上で、ユーザが制御する外部仕様について解説する。

4.1.1 プログラム全体に対する制御メソッド

FK System を用いたプログラムには、ウィンドウメッセージや描画イベントを処理する命令を含む、メインとなるループが存在する。以下に示すのは、サンプルから引用したループ中に記述する描画処理などの諸命令のコーディング例である。これらの一連の処理は、FK System を用いたどのプログラムでも、原則として必ず記述する必要がある。

```
        // メインウィンドウが最小化されている場合の処理
        if(main_win.visible() == 0) {
            if(Fl::wait() == 0) {
                break;
            } else {
                continue;
            }
         }
        // OpenGL による 3D 描画処理の呼び出し
        if(fk_win.drawWindow() == 0) break;
        // ウィンドウのメッセージ処理
        if(Fl::check() == 0) break;
        if(fk_win.winOpenStatus() == false) continue;
```

ヒストリーブラウザを利用するために、この部分の末尾に対して以下のメソッド呼び出しを追加する。プログラム上のセマンティクスとしては、ウィンドウを描画するタイミングと同期して、1 フレームの処理の中で 1 度実行されるように記述することになる。

```
        if(fk_HBUpdate() == 0) break;
```

fk_HBUpdate() は、ユーザプログラムの実行中にモデルの履歴を制御するブラウザを呼び出すために必要なメソッドである。このメソッドを追加した状態で、プログラムの動作中に**左 Ctrl+左 Shift+Tab** キーを押すことで、ブラウザのウィンドウが開くようになる。また、このメソッド内でブラウザに表示する情報を随時更新する処理や、ウィンドウメッセージの処理なども行っており、返値として 0 を返した場合はユーザプログラムの終了操作が発生したことを表している。この場合はメインループから処理を離脱することで、プログラムを速やかに終了する

ようになっている。

4.1.2 個々のモデルに対する制御メソッド

次は個々のモデルに対して、オブジェクトプロファイラ機能を有効にする命令を追加する。例えば testModel という名前の fk_Model のオブジェクトを追加したい場合は、履歴の取得を開始したい場所に対して以下のように記述する。

```
testModel.setHistoryMode(true, "Test");
```

fk_Model::setHistoryMode() は一番目の引数に true を設定し、二番目の引数にはそのモデルに対する名前を文字列で指定する。この文字列は、ブラウザ上でモデルを識別するための名前なので、半角英数字であれば分かりやすい名前を自由に付けてよい。この命令を実行した以降、モデルの位置や姿勢を制御する 3 次元座標変換操作の命令を呼び出すと、自動的に履歴が保存されてブラウザから操作することができるようになる。その後で引数に false を設定してメソッドを呼び出すと、オブジェクトプロファイラ機能を無効にし、これまでに取得した履歴を破棄する処理を行う。これにより、各モデルが履歴の取得を開始するタイミング、終了するタイミングはユーザプログラムによって自由に制御可能となる。

4.1.3 実行位置情報の付加

オブジェクトプロファイラによって取得する履歴には、メソッドを呼び出したプログラムのソース上の位置情報を付加することができる。これにより、累積した個々の制御がプログラム内のどのルーチンから実行されたかが把握できるので、不具合を招いている原因となっている箇所を容易に判別することができる。情報を付加するにあたっては、モデルの制御命令の前後に付加する文字列を引数に取っ

た追加の命令を記述する必要がある。この手法は本来、プログラム中の特定のスポットにおいて処理に要する時間を計測し、その結果をパフォーマンスのチューニングのために利用する、コードプロファイリングで用いられている手法である[30]。以下にその記述例を示す。

```
fk_PushProfile("hoge-");
// ここに fk_Model のオブジェクトへの処理が入る
fk_PopProfile("hoge-");
```

このように、fk_Model に対する命令を fk_PushProfile(“hoge-”) と fk_PopProfile(“hoge-”) で挟むことで、その中で実行された命令に対しては、その名前が履歴に付加できる。この PushProfile と PopProfile は、正しく対になっていれば入れ子の構造にすることもできるため、ループ構造やユーザーの作成したメソッドが途中で呼ばれても正しく名前を記録することができる。

プログラム上のメソッド名や変数名などのシンボル情報は、Ruby や C# などの言語ではメタ情報として取得し、プログラム上で利用することが可能であるが、本研究における実装では C++ を使用しているため、実行位置情報に関してはライブラリのユーザに手動で付加させるものとした。この作業に関しては、今後トランスレータなどを利用して自動化することを検討している。

4.1.4 描画タイミングと実行ステップ数の記録

3DCG のプログラム構造は、基本的に大きなループ構造になっていて、その中でフレームごとの描画と表示するモデルの状態を更新していくスタイルが一般的であり、FK System も同じ形態を採っている。このため、ループ内に記述した命令は描画のたびに繰り返し処理されて、どこからどこまでが 1 回のループで行った処理なのかが判別しづらくなる。そこで今回の実装では、描画を行うタイミン

グと同期して、全てのモデルに対して描画を行ったことを通知するコマンドを発行することとした。更に、このコマンドを発行した回数をカウントし、コマンドのパラメータとして付加することで、プログラム上で何回描画処理を行ったかを示すステップ数を記録し、履歴情報とあわせて参照することが可能になった。

4.2 ブラウザを用いたプロファイル情報の利用方法

ヒストリーブラウザは、オブジェクトプロファイラによって管理している履歴情報を制御するための機能であり、GUI として提供する。前節で述べたように、描画処理のタイミングにあわせて fk_HBUpdate() を呼び出すことで、実行中にユーザが任意のタイミングで呼び出すことができる。ブラウザを起動した時点でユーザプログラムの実行は一時中断し、プロファイリングの対象として選択しているモデルの履歴を操作するモードに移行する。以下の図 4.1 は、本研究で開発したヒストリーブラウザのスナップショットである。

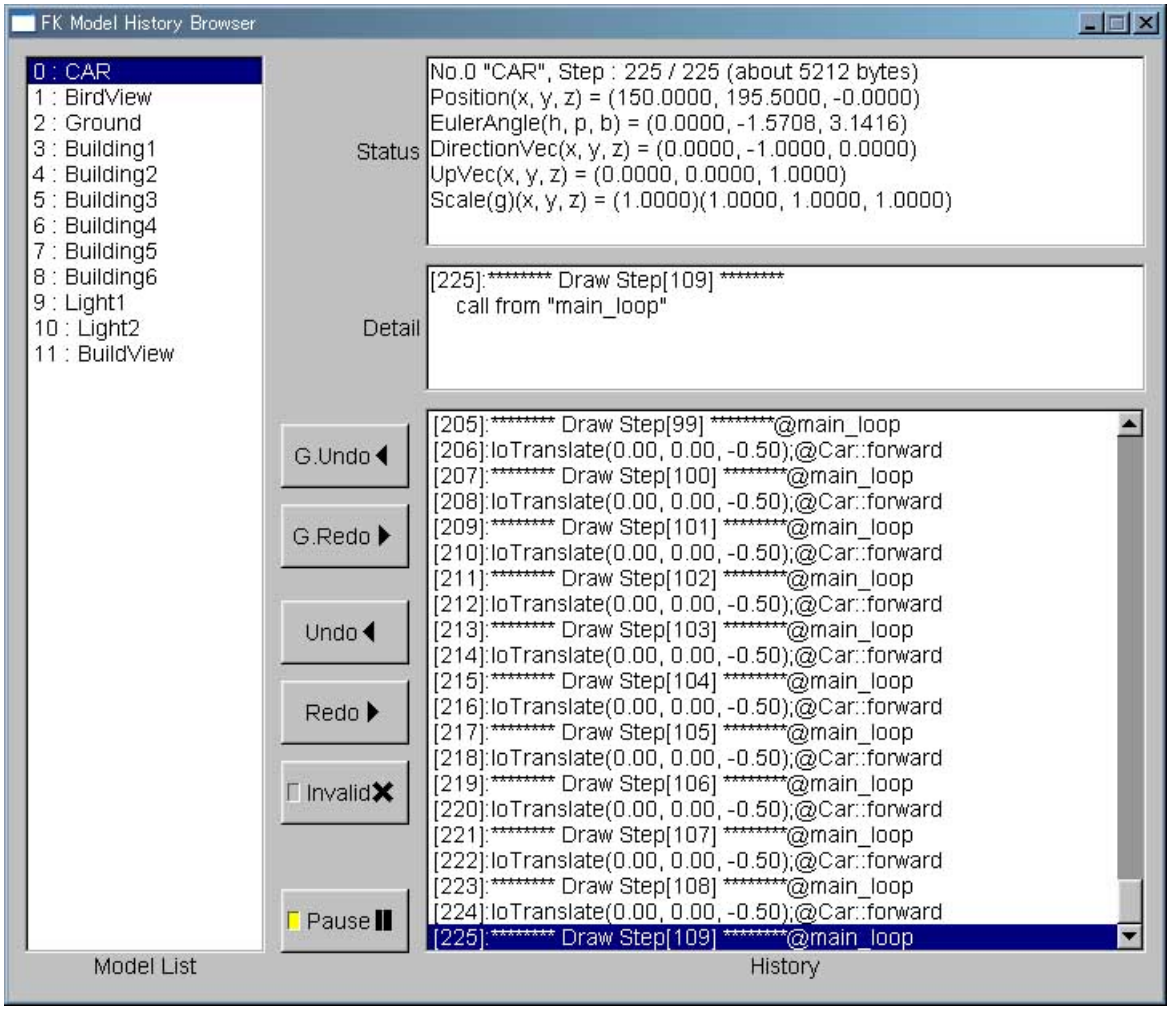

図 4.1: ヒストリーブラウザのスナップショット

ブラウザウィンドウ内の各表示内容について解説する。ModelList には、現在プログラム上で setHistoryMode(true) が呼ばれてオブジェクトプロファイラが有効になっているモデルの一覧を表示している。この一覧から、ブラウザによって履歴を操作するモデルを選択可能である。現在選択しているモデルの情報を、ブラウザウィンドウの右側に表示している。オブジェクトプロファイラによって取得した全履歴の内容は History に表示している。履歴の内容としては、プログラムが呼び出した fk_Model に対する 3 次元座標変換操作を処理するメソッド名と、引数として渡したパラメータを表示しているほか、プログラムが画面を描画したタ

イミングと回数を、Draw step という名前のコマンドとして表示している。更に、履歴の現在位置に関する詳細な情報を Detail に表示している。ここには、メソッドに渡している引数をより高精度で表示しているほか、状態を代入する操作の記録に際して、引数以外に保存してある代入前の値などを、History で表示している内容に加えて確認することができる。Status には、モデルの現在位置、姿勢のパラメータ、親子関係にあるモデル名などの情報を表示している。

4.3 モデルごとの Undo と Redo

ブラウザウィンドウ内の Undo/Redo ボタンで、個々のモデルで実行した命令一つずつに対して、巻き戻しと再生を行うことができる。また、HistoryView 上で任意の履歴をクリックすると、自動的にその地点までの Undo/Redo が行われる。この処理は前章で述べたように、各モデルが保持している履歴に沿って、逆操作や順操作を呼び出すことによって実現している。

4.4 プログラム全体に対する Undo と Redo

ブラウザウィンドウ上で、カーソルキーの↑↓を押すことにより、ブラウザに登録してあるモデルすべての動作を巻き戻したり、再生させることができる。カーソルキー以外に、ブラウザウィンドウ内の G.Undo/G.Redo ボタンでも同じ動作が 1 ステップずつ行える。この動作は、プログラムでウィンドウを描画したタイミングにあわせて各モデルに発行している Draw Step という名前のコマンドを区切りとして、各モデルの巻き戻しと再生を行っているもので、コンテンツ全体の状態遷移を把握するのに役立つ。

4.5 特定の操作の無効化

ブラウザウィンドウ内の Invalid ボタンを押すことにより、履歴の現在位置にある操作を無効化することができる。この機能は、表示上の挙動に異常が見つかっ

た場合などに、特定の操作を一時的に無効化することで、以降の挙動がどのように変化するかをシミュレーションしたい場合に役立つ。無効化によるシミュレーションで挙動が望ましい方向へ変化した場合は、その結果をプログラムへフィードバックすることにより、動作検証とプログラムの修正を効率よく反復することができる。

4.6 Undo 後の動作再開

ブラウザウィンドウ内の Pause ボタンを押すか、ブラウザウィンドウ自体を閉じると、ユーザプログラムの一時停止を解除し、動作を再開させることができる。この時モデルの履歴を巻き戻してあった場合は、一時停止の解除後もモデルの状態が巻き戻ったままで、ユーザプログラムの動作が再開する。巻き戻した履歴の内容は消滅し、再開後の動作による履歴によって上書きされる。

この場合、ヒストリーブラウザの管理下にあるモデルについては巻き戻った状態になるが、ヒストリーブラウザの管理外にあるモデルや、その他のプログラム中で使用している変数などの値は、ヒストリーブラウザを開いて動作を一時停止させた状態のままであるため、状態が巻き戻っているモデルとそれ以外の変数で、状態の時系列的な乖離が起きることになる。このため、ユーザプログラムの仕様によっては正しい動作に復帰しない場合が起こり得るので、挙動を観察する際には注意が必要である。

第 5 章

検証と考察

本章では、本研究で作成したオブジェクトプロファイラを内包するライブラリを用いて、実際にコンテンツの動作を解析する際の有用性を検証する。検証にあたっては、本学メディア学部の演習授業及び卒業研究において作成されたプログラムを使用した。また、演習授業を履修中の 3DCG プログラミング初学者に対して、オブジェクトプロファイラ機能を提供し、その効果についても考察を行った。

5.1 ビリヤードゲームにおける動作検証

3DCG を用いたコンテンツでは、現実の物体の運動を表現するために、物理学に基づいたシミュレーションを行ってモデルの動きを制御することが多い。本節で紹介するビリヤードゲームのようなコンテンツも、2 次元の平面上ではあるが、衝突や摩擦を考慮した簡単な剛体シミュレーションによって構成している。以下の図 5.1 は、今回動作検証を行ったビリヤードゲームの動作イメージである。この作品は、本学メディア学部の演習授業において最終課題として作成されたものである。

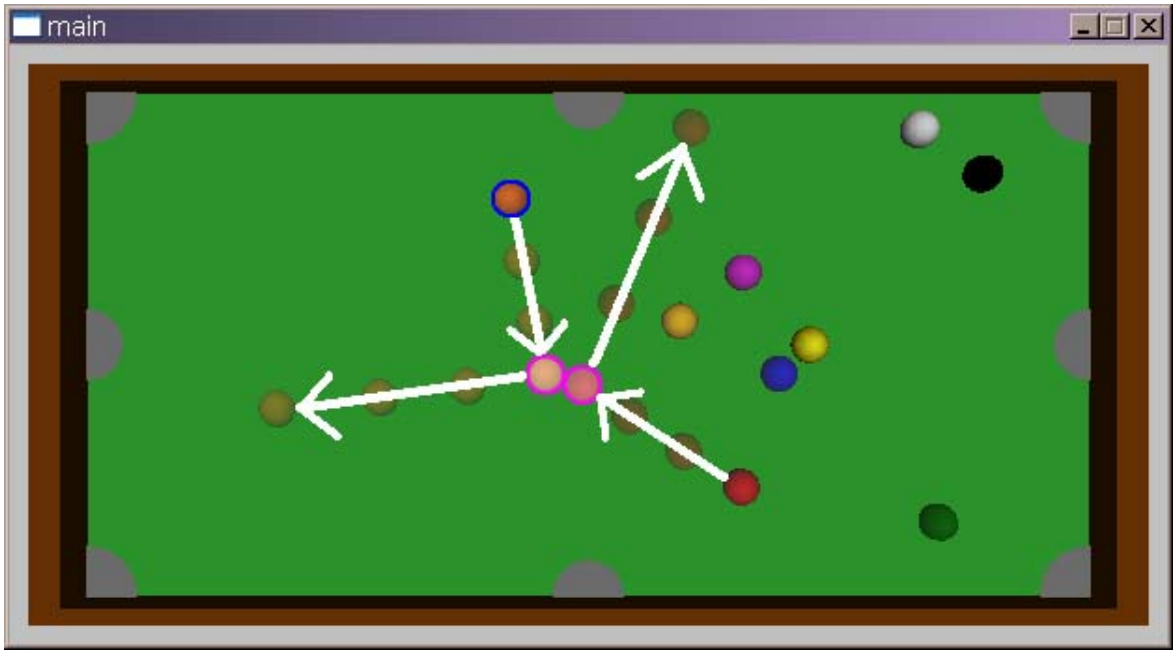

図 5.1: ビリヤードゲームの動作イメージ

このようなシミュレーションに基づくモデルの制御を行う場合は、まさにライブラリによる直感的な制御が非常に有用であるが、意図とは異なる挙動が生じた場合の動作検証は手間のかかる作業となる。特に上の図で示したような、衝突の瞬間に各モデルがどのような状態にあり、どのような操作を行っているのかを把握するのは難しい。

そこで本研究で作成したライブラリ上でこのゲームプログラムをコンパイルし、ボールを表現している各モデルの挙動を解析した。結果、わずかなプログラムコードの追記によって、ボールの動作を任意の時点で停止し、その瞬間の各ボールの状態や実行している処理を容易に確認することができた。衝突が発生した瞬間の再現も、衝突後にヒストリーブラウザを開き、プログラム全体に対して 1 ステップずつ Undo を行うことで、容易に実現できた。

しかし、プログラマのコーディングスタイルによっては、プロファイラの機能が十分に発揮できないことも確認できた。このビリヤードゲームの場合、各ボールの運動する方向や位置を全てユーザプログラムの中で処理してしまい、fk_Model の内部状態を一切使用せず、表示位置を MoveTo 命令によって直接代入する方法で処理を実現していた。このため、ヒストリーブラウザによって解析が可能なのは、位置を直接代入する履歴の羅列だけで、その時点でボールが持っている速度のベクトルを直感的に分析することは困難であった。このようなコーディング例は、モデルに対する相対的な操作の感覚がつかみづらい初学者に多く見られるが、モデルの状態とプログラム内部の変数との情報の乖離をどのように吸収するかが、本手法における今後の課題となることを顕著に示した事例である。

5.2 魚群の遊泳シミュレーションにおける動作検証

3DCG の重要な用途として、様々なシミュレーションの可視化という目的が挙げられる。本学における卒業研究でも、FK System を使用した群集シミュレーションを題材とした研究が数多く見られる [31][32][33][34]。本節では、2005 年度の卒業研究 [35] において学部 4 年生が製作した、魚群の遊泳シミュレーションプログ

ラムに対して、オブジェクトプロファイラ機能を用いた事例を紹介する 。以下の図 5.2 は、今回検証に用いたシミュレーションプログラムの実行画面である。

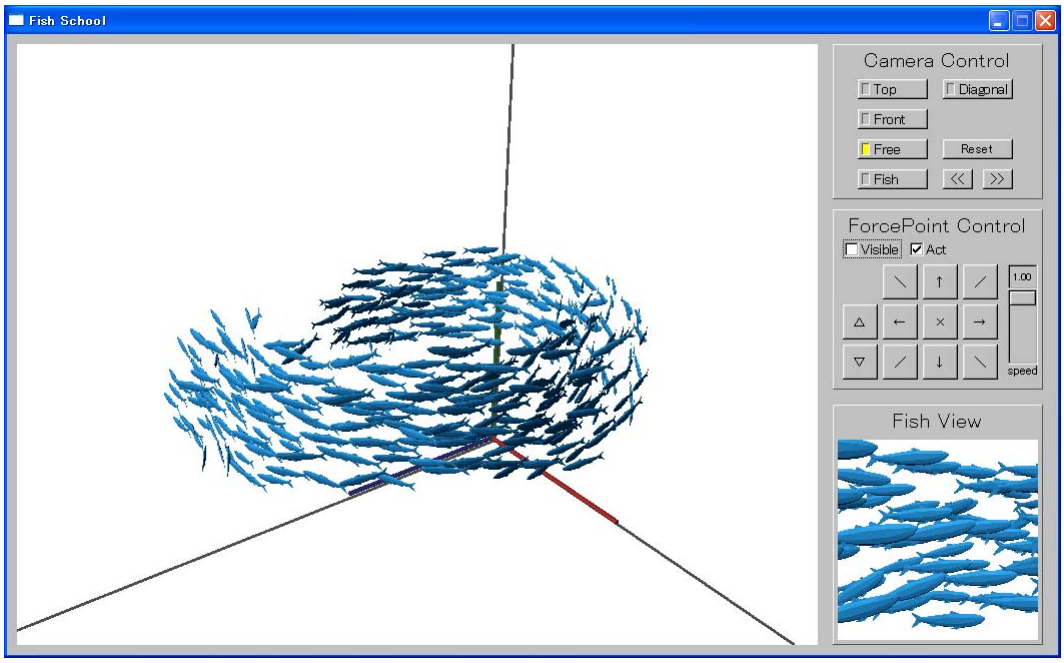

図 5.2: 魚群の遊泳シミュレーション

このシミュレーションでは 400 体にも及ぶ魚のモデルに対して、一定の法則に基づいた姿勢や速度の制御を行うことで、より自然に魚群が遊泳している様子を表現することを目的としている。生態系のシミュレーションは、生物の思考パターンを再現する関係上、様々なパラメータがモデルの挙動に影響を与えるため、望ましい挙動を得るための調整が非常に困難な作業になりがちである。そこで本研究で作成したライブラリを、魚群の遊泳シミュレーションプログラムの開発者に提供し、実際の調整作業において利用してもらうこととした。このシミュレーションプログラムは 400 体のモデルを制御対象としているため、オブジェクトプロファイラを使用することによる、パフォーマンステストとしての意味合いも兼ねて検証を行った。以下の図 5.3 は、実際にヒストリーブラウザによってシミュレーションプログラムの動作解析を行っている様子を示している。

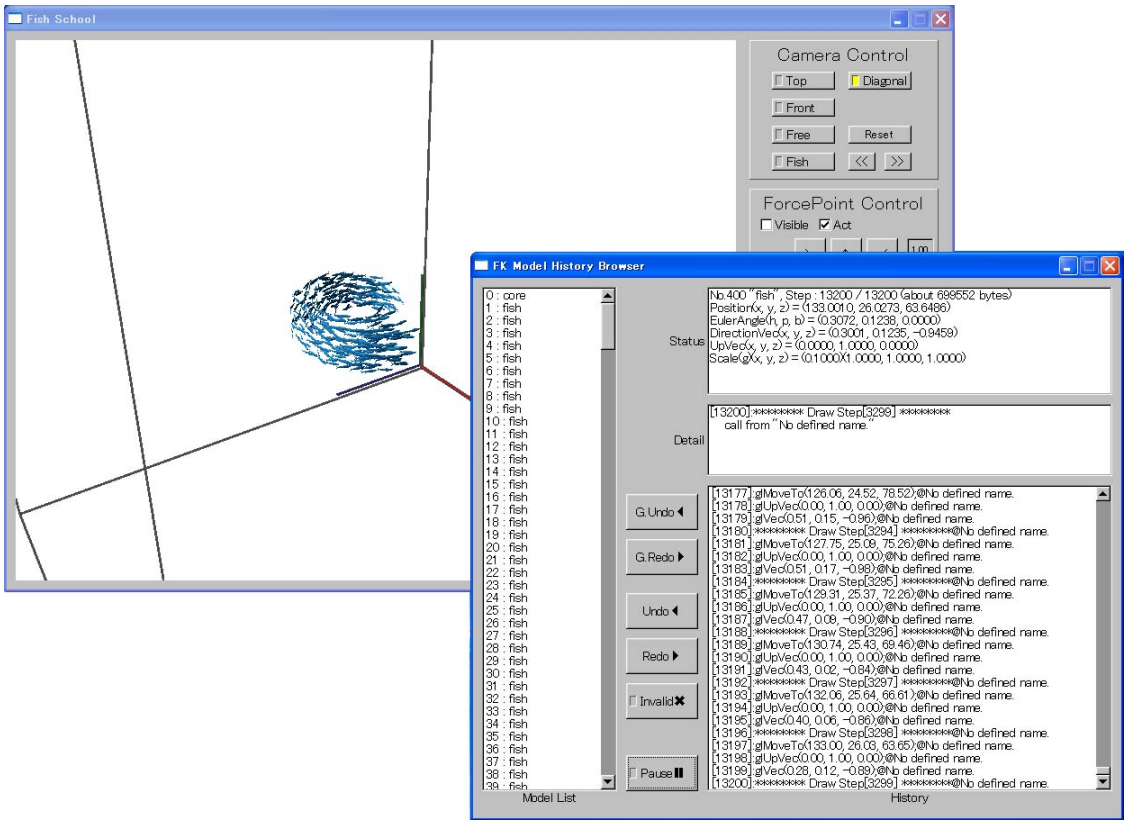

図 5.3: シミュレーション中の解析作業の様子

シミュレーションプログラムの開発者に、オブジェクトプロファイラを使用して動作の調整作業を行った場合の利点をたずねたところ、プログラムの動作解析が非常にスムースに行え、良好な結果が得られたとの回答を得ることができた。実際に作業の様子を検証したところ、魚のモデルが不自然な姿勢を取った場合に、引数として与えた数値が即座に確認でき、すぐにプログラムへの修正として反映ができた。修正作業に際しては、実行した命令がプログラムのどの部分で記述されているのかを、前章で述べた実行位置情報の付加機能を用いることによって簡単に把握できるため、シミュレーションの目的に即した調整が容易になったなどの効果が得られたことを確認した。

懸案だった動作のパフォーマンスも、オブジェクトプロファイラ機能を無効にした場合と有効にした場合で、体感的な速度差を感じることなく動作することが

確認できた。オブジェクトプロファイラ機能を有効にした場合、履歴を配列に蓄積する処理が純粋にオーバーヘッドになるが、400 体のモデルに対して履歴の取得処理を実行した場合でも処理速度に体感差が出なかったことから、本手法には、処理速度面のコストを考慮した上でも有用性が確認できた。

ただし、オブジェクトプロファイラ機能を利用したプログラムを長時間動作させていると、各モデルの履歴の件数が増加し、使用するメモリの容量は増加していく。この検証で用いたマシンはメモリを 1GB 搭載しており、検証中にメモリ不足に陥ることはなかったが、環境によっては動作の継続が困難になったり、パフォーマンスの大幅な低下を招く可能性は十分にある。この問題を回避するため、ある程度の履歴の件数に到達した時点で履歴の内容をディスクに退避するなどの対応処理を付加する必要がある。また今回の実装では、履歴を記録する配列として、C++ の標準テンプレートライブラリが提供する可変長配列の vector クラスを使用しており、メモリ確保などの管理を全て委任しているが、今後仕様を検討する際には、インテリジェントなメモリの管理制御を独自で行うことも視野に入れる必要がある。

更に、このシミュレーションプログラムの開発者からは、GUI の機能に対して多くの追加の要望が挙げられた。具体的には、一連のシミュレーションの実行結果をディスクに保存し、そのデータを読み込んで再現する機能や、プロンプトベースのコマンド入力による、履歴の編集機能などが意見として挙げられた。これらの機能は、オブジェクトプロファイラによって取得した履歴を有効に活用する上で、非常に有用な機能になり得る。本節で検証したシミュレーションには乱数の要素が含まれており、実行するたびに異なる結果が得られることになるが、ある特定の条件下で生じる現象に対して挙動を解析したいと思った場合、その条件を任意に発生させることは困難である。このような場合に一連の挙動を保存して再現することが出来れば、非常に容易に分析が可能となるであろう。また、本研究においては GUI からの制御機能を最低限のものしか用意出来なかったため、履歴の編集機能は一時的な取り消し処理しか実装しておらず、十分整備されていると

は言えない。今回の運用によって得られた意見のように、モデルの履歴を動的に編集してその挙動をシミュレーションする機能は、今後是非とも実現していきたい要素である。

5.3 プログラミング初学者に対する効果の検証

本研究で拡張を施した FK System は、本学メディア学部のコア演習科目である、“3DCG プログラミング” において元々使用されているものである。本研究をまとめるにあたり、目的の一つであった「3DCG プログラミング初学者に対する効果」を検証するべく、2005 年度後期に開講された当該の演習において、本研究の成果物であるライブラリとマニュアルを履修者である学部 3 年生に提供し、その利用実態を調査し、効果を検証した。

結論から言うと、本当の意味での初学者にとっては、あまりメリットの分からない機能という印象が強いようで、積極的に利用していた学生は散見される程度しかいなかった。オブジェクトプロファイラという機能は、モデルという抽象化したオブジェクトを操作する、という概念を明確に意識できてこそ有用なのであり、3DCG やオブジェクト指向に不慣れな初学者に対して即効的な効果は期待できないことが分かった。

しかしながら、学部 3 年生の中でもプログラミングや 3DCG に造詣が深く、FK System を既にある程度使いこなしていたり、積極的に理解しようとしている学生は、オブジェクトプロファイラ機能に興味を示し、実際に使用している場面を観察することができた。中でも、物体のアニメーションを表現するために試行錯誤を繰り返していた学生に対しては、ヒストリーブラウザによる履歴の表示が、非常に有効に機能していた場面があった。以下に、その学生が行っていたコーディング例を示す。

```
while(true) {
    // ウィンドウの描画とメッセージ処理
```

```
        .
        .
        .
        // ここまで

        // アニメーションを表現しようとして記述したコード
        for(int i = 0; i < 100; i++) {
         // 指定した座標への移動命令
            ball.glMoveTo((double)i, (double)i*i, 0.0);
        }
    }
```

このコーディングでは、ball という fk_Model のオブジェクトに対して $y = x^2$ の関数によって導かれる放物線を描くアニメーションを実現しようとしているのだが、これでは期待した通りの動作は得られない。モデルの制御命令は 1 フレームの描画ごとに呼び出さなければ、軌跡を描くような挙動にはならないからである。上記のようにコーディングしてしまうと、1 フレームの描画のうちにモデルの状態が $i = 99$ の時点まで推移してしまうため、アニメーション処理とはならないのである。

このようなミスは、プログラムのフローを正しく追えていれば容易に気が付くのだが、描画ループという既にある枠に対して自らの実現したい処理を変形させて記述することは、3DCG プログラミング初学者にとっては困難であり、また理解させるのも難しい。そこで、この学生に対してヒストリーブラウザによる履歴を表示させ、描画のタイミングとモデルの制御のタイミングについてレクチャーを行った。その結果、1 回の描画に対してモデルの制御命令が大量に処理されている状況を目の当たりにして、自らのコーディングの問題点に気が付かせることができた。

以上のような事例もあり、プログラムの挙動を情報として可視化することは、初学者のステップアップに対しても有用な支援になり得る。プログラムは思った通りには動かず、書いた通りに動くものであるが、自らが記述したプログラムがど

のように動くかを類推するには、知識と経験によって裏打ちされたセンスが必要である。そのセンスを初学者にも求めるのは、現在の複雑化したプログラミング環境においては酷である。今後、本手法によってプログラミング技術の習得を支援することを目的とした場合、プログラミング初学者に対して、どのように挙動を情報として提示するかを検討することは、重要な課題となる。

5.4 オブジェクトプロファイラに対する考察

前節までに述べたように、本論文においては 2 つのコンテンツに対する適用試験と、本学の演習における運用試験による検証を行った。これらの検証を通じ、ゲームとして完成されたコンテンツ、ある程度の大規模なシミュレーション、初学者に対する学習効果という 3 つの視点から、本手法の有用性を確認することができた。

本研究は、3DCG という分野においてはあまり研究対象とされてこなかった、開発環境に焦点を当てたものである。この分野において主に研究論文として発表されているのは、新しい表現技術に関するものがほとんどで、開発環境そのものは既に製品として提供されているものを利用することに腐心しているように見受けられる。ましてや開発の手法や効率化などは、学術的な文書の上で論じられることがほとんど無かった。本研究は、このような現状に一石を投じるべく進めたものである。

本手法が提案したのは、新しい表現技術でもなければ、真新しいデバッグの手法でもない。本手法をデバッグの一手法としか見ないのであれば、オブジェクトプロファイラを用いて実現しているような動作テストは、既に開発者は労力を費やして行っているのである。本手法の意義は、このような労力を費やす作業を肩代わりするべく、挙動を情報として管理可能にするようにライブラリを設計するという視点を提案したことにある。これにより、これまでライブラリを利用して開発を進める際に要していた、過度な試行錯誤を大幅に軽減できる可能性が生まれたのである。

プログラミングの歴史は抽象化の歴史でもある。オブジェクト指向の概念が開

発者の間に浸透しつつある今、クラスオブジェクトなどの抽象化した階層での動作の理解は、もはや基礎知識となっている。今後はプログラミングの方法論だけを論じるのではなく、プログラムの実行結果を解析する方法論も、開発の手法に応じて変化していくべきであろう。

第 6 章

おわりに

本章では本論文の締めくくりとして、まとめと今後の展望について述べる。

本研究では、既存の 3DCG クラスライブラリに対してオブジェクトプロファイラと呼称する機能を追加し、ライブラリを用いて作成したプログラムの挙動を容易に解析可能にする環境を提案し、構築した。オブジェクトプロファイラとは、3DCG シーン中で表示する 3 次元モデルに対する位置や姿勢を制御する命令をコマンドベースの履歴として記録し、プログラム上から履歴を操作可能なヒストリーブラウザと呼称する GUI によって制御することで、プログラムが実行したメソッドと実際の表示の対応をインタラクティブに確認できる環境を提供するものである。

本手法は、ライブラリのユーザが記述したプログラムの挙動をライブラリ側で把握し、制御可能な情報として構造化して提示するという、既存のライブラリやプログラミング支援環境とは異なる独特のアプローチによって成り立っている。このため、本手法が提供する機能は、冗長なコーディングを必要とせずに利用できる上、ヒストリーブラウザという GUI を通じて、メソッドが引き起こす動作を直感的に理解できるというメリットを生み出した。このメリットは、プログラミングの熟練者から初学者にいたるまで享受することができ、各々のユーザに応じたレベルでの支援が可能な開発環境を実現した。本手法を用いることが有用な状況として、既存のプログラムの動作解析、アニメーションプログラミングの動作確認支援、デバッグ、高速化のために過剰な命令を排除するためのパフォーマンスチューニングなどが挙げられ、様々な用途が期待できる。

今後の課題としては、3 次元座標変換操作以外のメソッドへの対応や、トランスレータを用いたソースに対するメタ情報の自動的な付加、履歴を巻き戻した時点から新たな操作を行った場合の分岐に対応することによる、並列的な履歴管理の実現などが挙げられる。またヒストリーブラウザに拡張を加えることにより、履歴を任意のファイルに保存し、読み込むことによる挙動全体のロード/セーブ機能や、GUI からの操作のみではなく、コマンドラインからの入力による、任意の命令の追加、編集機能など、インターフェースの改善によって、オブジェクトプロファイラは更なる真価を発揮することだろう。

なお、本研究は情報処理学会第 67 回全国大会において “リアルタイム 3DCG プログラミングにおける、ビジュアルプロファイラの有用性に関する研究” [36] として発表した内容を含む。

謝辞

本研究を締めくくるにあたり、学部時代から引き続きまして研究の指針から開発の手法、論文の執筆と幅広いご指導ご教授を頂きました、本校メディア学部の渡辺 大地 講師及び、演習講師として研究の進め方のご指導を頂くと共に、博士課程への道を示して頂きました、電気通信大学の和田 篤 氏に心より感謝いたします。また、主査をお引き受け下さいました、本校メディア学部の金子 満 教授及び、副査をお引き受け下さいました、本校メディア学部の宮岡 伸一郎 教授に感謝いたします。

在学中にメディア学の在り方や、研究者としての心得などについて、活発な議論をしていただきました、本校大学院メディア学研究科卒業生の渡邊 賢悟さんに感謝したいと思います。さらに、研究を進めるにあたって様々な意見を交換してくれた、本校大学院メディアサイエンス専攻の学友諸氏に感謝します。特に、3 年間同じ研究室で学んだ石川 朱香音さん、藤原 大三郎さんには色々とご協力をいただきました。また、本研究の成果物の運用に協力してくれた石塚 拓磨さんをはじめ、本校メディア学部の 3DCG コンポーネントプロジェクトのメンバーに感謝します。そして、いつも私を支えてくれた家族と、全ての友人たちに感謝します。

最後に、本研究にご協力いただきました全ての皆様と、この論文に目を通してくださった全ての方々に、厚くお礼を申し上げます。

参考文献

[1] Microsoft, DirectX, <http://www.microsoft.com/japan/msdn/directx/>.

[2] OpenGL.org, OpenGL, <http://www.opengl.org/>.

[3] Autodesk, Maya, <http://www.alias.co.jp/>.

[4] Autodesk, 3ds Max, <http://www.autodesk.co.jp/>.

[5] Criterion Software, RenderWare, <http://www.renderware.com/>.

[6] CRI Middleware, CRI Sofdec, <http://www.cri-mw.co.jp/>.

[7] Microsoft, XBox / XBox360, <http://www.xbox.com/>.

[8] 株式会社フォトロン, 図脳RAPID3D, <http://www.photron.co.jp/>.

[9] KGT Inc., PortableVR, <http://www.kgt.co.jp/product/avs/pvr/>

[10] エレクトロニック・テキストブック「人体ウォークスルー」, The Future of the Book of the Future, 情報処理振興事業協会, pp.46-51, 1995.

[11] Nintendo, GAMECUBE, <http://www.nintendo.co.jp/ngc/>.

[12] Truevision 3D, <http://www.truevision3d.com/>.

[13] Botchy, Easy Link Library, <http://www.twin-tail.jp/contents/el/>.

[14] 葉迩倭, Luna, <http://www.twin-tail.jp/contents/luna/>.

[15] Sam Lantinga, Simple DirectMedia Layer, <http://www.libsdl.org/>.

[16] Jorrit Tyberghein, Crystal Space 3D, <http://www.crystalspace3d.org/>.

[17] 渡辺大地, “リアルタイムグラフィックスのためのツールキットに関する研究”, 慶應義塾大学大学院政策・メディア研究科修士論文, 1996.

[18] Nikolaus Gebhardt, Irrlicht, <http://irrlicht.sourceforge.net/>.

[19] Sony Computer Entertainment, PLAYSTATION3, <http://www.playstation.jp/>.

[20] 大谷淳平, 株式会社アントラッド, “Lamp：教育的かつ実用性のあるゲームミドルウェア”, 独立行政法人 情報処理推進機構, 未踏ソフトウェア創造事業, <http://lamp.sourceforge.jp/>.

[21] Enterbrain, ツクール Web, <http://www.enterbrain.co.jp/tkool/>.

[22] Edsger Wybe Dijkstra, 「構造化プログラミング」, サイエンス社, 1967.

[23] Waldemar Celes, Jonathan Corson-rikert, “Act: an easy-to-use and dynamically extensible 3D graphics library”, Proc. of SIBGRAPI, 1997.

[24] Christian Geiger, Wolfgang Mueller, Waldemar Rosenbach, “SAM - An Animated 3D Programming Language”, IEEE 1998 Symposium on Visual Languagesm IEEE Press, Los Alamitos. CA., 1998.

[25] Randy Pausch (head), Tommy Burnette, A.C. Capeheart, Matthew Conway, Dennis Cosgrove, Rob DeLine, Jim Durbin, Rich Gossweiler, Shuichi Koga, Jeff White, “Alice: Rapid Prototyping System for Virtual Reality”, IEEE Computer Graphics and Applications, May 1995.

[26] Matthew Conway, Steve Audia, Tommy Burnette, Dennis Cosgrove, Kevin Christiansen, Rob Deline, Jim Durbin, Rich Gossweiler, Shuichi Kogi, Chris Long, Beth Mallory, Steve Miale, Kristen Monkaitis, James Patten, Jeffrey Pierce, Joe Schochet, David Staak, Brian Stearns, Richard Stoakley, Chris Sturgill, John Viega, Jeff White, George Williams, and Randy Pausch, “Alice: Lessons Learned from Building a 3D System for Novices”, CHI 2000.

[27] 西田彩, 鎌田十三郎, 瀧和男, “適応的な 3DCG アニメーションのためのプログラミング支援ツールの提案” 日本ソフトウェア科学会全国大会第 19 回大会

講演論文集, 2002.

[28] 渡辺大地, FK Tool Kit System, <http://www.media.teu.ac.jp/~earth/FK/>.

[29] 鳥谷浩志, 千代倉弘明, 「3 次元 CAD の基礎と応用」, 共立出版, 1991.

[30] Steve Rabin, “リアルタイム ゲーム内プロファイリング”, GAME PROGRAMMING Gems, 1.14, pp.118-128, Born Digital, Inc., 2001.

[31] 森江修也, “集団移動シミュレーションにおける移動障害物の回避に関する研究”, 東京工科大学メディア学部卒業論文, 2002.

[32] 石川朱香音, “人間の行動特性を考慮した雑踏における自律エージェントモデルの歩行行動”, 東京工科大学メディア学部卒業論文, 2003.

[33] 藤原大三郎, “群衆流動シミュレーションにおけるグループ歩行表現導入の有効性”, 東京工科大学メディア学部卒業論文, 2003.

[34] 中部智也, “群衆の経路プランニング”, 東京工科大学メディア学部卒業論文, 2004.

[35] 石塚拓磨, “リアルタイム 3DCG における密集した魚群の遊泳表現に関する研究”, 東京工科大学メディア学部卒業論文, 2005.

[36] 竹内亮太, 渡辺大地, “リアルタイム 3DCG プログラミングにおける、ビジュアルプロファイラの有用性に関する研究”, 情報処理学会 第 67 回全国大会, 2005.

発表論文

竹内亮太, 渡辺大地, “リアルタイム 3DCG プログラミングにおける、ビジュアルプロファイラの有用性に関する研究”, 情報処理学会 第 67 回全国大会, 2005.