社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－CG－128 (13)
2007／8／17

MELスクリプトデバッガの研究開発

石川 雄介
e-mail : yusukei@mac.com
web : http://web.mac.com/yusukei/
神奈川工科大学大学院情報工学専攻

Mayaは3DCGソフトウェアの代表的なものの一つである．MayaはMEL(Maya Embedded Language)というスクリプト言語をサポートしている．これを使いデザイナーは複雑なアニメーションを制御することができる．しかし，開発環境は十分な環境を備えていない．たとえば，MELを実行中にブレイクし，そのときの変数の値を参照したり，コールスタックの表示，ステップ実行，実行の中断などといった機能を持ち合わせていない．このため，古典的な伝統的な方法でしかデバッグを行うことができない．そこで，MELスクリプトの開発を支援すべくデバッガの開発をおこなった．

Research and Development of MEL Script debugger

Yusuke ISHIKAWA
Infomation and Computer Sciences,
Graduate School of Kanagawa Institute of Technology

Maya is one of the most famous 3DCG software. Maya supports a script language whose name is MEL(Maya Embedded Language). The 3DCG designers can make fertile expression of 3DCG animation by using MEL. But, the developmental environment of MEL Script is not sufficient. For example, it supports not checking the values of a variables, not expressing call stack, not running in step by step, and not breaking the run. The traditional print debug is only the method to debug the MEL Script source codes. Thus The author developed a debugger system for MEL Script.

1. まえがき

近年の3DCG，VFXが急速に進化して，使用されるようになってきた．これらはとても高度な技術で作られており，また，複雑な計算の基にあらわされている．ただしその分，開発には時間がかかり，高度なスキルを要求しているのも現実である．特に，映画などの大規模なプロジェクトではアニメーションなどの制御にスクリプトを使用する．これらのおかげで非常に多彩な表現が可能になるものの，その開発期間はとても大きいものとなっているのが現状である．本研究の目標として，3DCG分野におけるスクリプト開発の補助とした．研究開発の対象とする3DCGソフトウェアとしてMayaを選んだ．これはMayaがハリウッドにおける主要な3DCGソフトウェアの代表的なものの一つであることと，MELというスクリプト言語に対応していること，また研究室にも導入されているといった点から選んだ．まず，この分野における開発支援をするツールなどの調査を行い，現業での問題を洗い出した．そうしたところ，Mayaにはデバッガとしての機能を備えておらず、スクリプトの開発環境は十分な機能を備えているとはいえないことがわかった．そこで，スクリプトの開発時間を短縮するために．標準の機能で提供されていないデバッガの研究開発を行った．また，本研究の方針としてはユーザーに使いやすいものであり．デバッグに必要な情報が十分に取り出せるものであることとした．

2. デバッガ実装方法

2.1 デバッガの仕様

デバッガを作るにあたって基本的な機能要求の提示を行う.

・ブレイクポイントを設定
・ブレイクポイントから再実行
・ブレイクポイントからステップ実行
　ステップイン
　ステップアウト
　ステップオーバー
・コールスタックを参照
・コールスタックの位置を変更
・変数を参照

それ以外として
・デザイナーが使えるようにメニューを作成
・マルチプラットフォームを意識

これらの機能要求を基本として開発していくことにした.

2.2 デバッグ手法

まず, MEL スクリプトのコードをデバッグするために必要な機能を洗い出した. その結果, 以下のようなものが必要だと判断した.

(1) 変数の Watch
　・現在のスコープ内の変数の名前と値
　・スコープの管理
(2) ブレイクとステップ実行
　・コールスタック
　・現在実行しているファイル内の位置
(3) コールスタック
　・スタックを積む位置
　・スタックを降ろす位置
(4) 3 までの操作を MEL のコードに埋め込む

デバッガを作る際に重要なのがコールスタックおよび変数のスコープである, この両者はほぼ同一の性質をもつ. この部分ができなければデバッガの機能の多数が実装不可能になってしまう. そして, 最も重要なのが, それらの操作を MEL スクリプトのコードに埋め込む部分である. そして, 開発に一番時間がかかり, 苦労した部分でもある.

2.3 Watch の実装

実装の簡単な変数の Watch を実現するためのコマンドから開発に取りかかった. 変数を Watch するためには変数の名前と値が必要である. そのため著者は変数の名前と値を受け取るコマンドを作成した. このコマンドは dbgValue というコマンド名で -set フラグをとり, その後ろに変数名, さらに後ろに変数の値をとるコマンドである. $val という変数があったとき, このコマンドは,

```
dbgValue -set " $val" $val;
```

という呼び出しをすることで, 変数名と変数の値を受け取ることができるようにした.

2.4 コールスタックの実装

次に重要な機能の一つであるコールスタックの部分を作った. このコールスタックであるが, 実装上スコープのスタックになるように実装した. それはスコープのスタックはコールスタックを含むためである. ある変数が宣言された場合, その変数の生存時間は宣言されたスコープと同一である. スコープがスタックされるタイミングを考えると, 関数に入ったとき (コールスタックが積まれたとき), if 文のブロックに入ったとき, for 文のブロックに入ったとき, while 文のブロックに入ったとき, do 文のブロックに入ったとき, switch 文のブロックに入ったときである. スタックを降ろす場合は同じことが言える. あとは { があればスコープのスタックを積み, } でスコープのスタックを降ろせばよい. ここで return 文については特別な処理をしなければならない. return 文はコールスタックを一つ降ろすと同等の意味を持つ. コールスタックの動作を図 2.1 に示した.

コールスタックが完成したら, これと先に開発した変数の Watch 部分を結合する. 受け取った変数が同一のコールスタック内になければ新たに現在のスコープ内に変数を登録する.

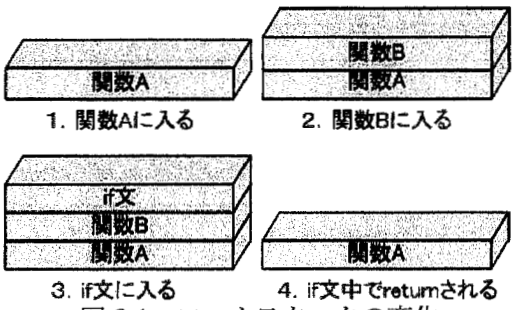

図 2.1　コールスタックの変化

2.5　ブレイクポイントの実装

ブレイクポイントを実装する上で重要なのが位置の取得である．ブレイクポイントはMELスクリプト名と行数の対で設定する．そのため，実行位置の取得は必須である．そこで，パース時に1行ごとに行の頭にコマンドを埋め込み何行目を実行しているのかをデバッガが判断できるようにした．また，コマンドを埋め込められない場所があった場合，埋め込めるようになった場所にまとめて埋め込むようにすることで，見た目には設定した通りの行数で止まったかのように見せかけている．

MELスクリプトには行頭にコマンドを配置できない場所がある．それは以下の通りである．

(1) proc の前
(2) case の前

また，行末が；で終わっているが，次の行にコマンドを配置できない場合がある．それはUI関係のコマンドである．その場合，行頭がーで始まるため，ーで始まる行についてはコマンドを埋め込まないようにしている．

2.6　メッセージの処理

ブレイクポイントで止まっている間はプラグインの中で無限ループしている状態となる．そして，状態が変わればループを抜けるという動作になる．このとき長い時間 Maya は OS からのメッセージを処理しないため「応答なし」と判断される．これにともなって，ウインドウの再描画もされない．そこで，この問題に対し、ブレイクしているループの中でメッセージをディスパッチすることで対処を行った．

2.7　ステップ実行の実装

ステップ実行には3種類を実装した．

(1) ステップイン

ステップインはブレイク中の行が関数だった場合，その関数に入り関数の頭でブレイクする．つまり，MEL スクリプトでの最小の行単位での実行と等しい．そのため，実装はブレイクポイントの設定とは関わらず，次に行数を受け取ったときに必ず再度ブレイクするという動作である．

(2) ステップアウト

ステップアウトは現在のコールスタックを抜けるというものである．実装ではコールスタックのスタックの積み重なる数が減ったときに再度ブレイクするという動作である．

(3) ステップオーバー

ステップオーバーは現在のコールスタックの位置で次の行にきたらブレイクするものである．実装としては現在のコールスタックと同じ深さで次の行数を受け取ったときに再度ブレイクする．

2.8　パーサとコマンドの埋め込み

トークンナーを使用し，パーサが MEL スクリプトを構文を解釈していく．パーサが解釈した結果を用い，デバッガはデバッグを行うためのコマンドを埋め込んでいく．パーサを開発する際に工夫した点はパースをきっちりとは行わず，必要な情報を抽出していくだけという形をとった．これは，もし途中で構文エラーがあったとしても，それは Maya が MEL スクリプトをコンパイルする際にエラーとして処理されるのと，ある程度あいまいにしておくことで，将来 MEL スクリプトの構文が拡張，修正された際に，何もしなくても動作する可能性があるためである．

2.9　MEL スクリプトの書き換え

まずは MEL スクリプトをデバッグするためには2通りの考え方ができる．1つは実行時に MEL のプログラムを書き換える．

２つめは実行される前に MEL のプログラムを書き換える．

１つめの方法を実現するためには Maya から実行されている情報を取得することが不可欠である．Maya から実行中の MEL に対する情報を取得するには MComandMessage という API クラスがある．このクラスの addCommandCallback メンバ関数を呼び出すことで Maya がコマンドを実行する前に登録した関数を呼び出すようになる．しかしながら，コールバックされた関数内でコマンドを書き換えるなどと言ったことはできないので，この方法での実装は無理と分かった．

次に２つめの方法であるが，これを実現するにはファイルにかかれた MEL スクリプトはいつ呼び出されるのかということを調べなければならない．そしてそれは Complete Maya Programming [1]にかかれている．これによると，MEL ファイルが読み込まれるタイミングは source コマンドで呼び出された時である．つまり，MEL スクリプトを source コマンドで明示的に呼び出されたときか，source されていない MEL スクリプトのコマンドが呼ばれたときである．このタイミングを正確に知り得ることは難しい．従って，Maya が拡張子 mel と付くファイルを開いたときに，その MEL スクリプトを書き換えるという方法をとった．つまり，Maya が拡張子 mel を開くタイミングとは，Maya が Win32API の CreateFileA を呼び出したときである．このタイミングで Maya に割り込みをかければいいわけである．これをプログラム的に考えると，CreateFileA の関数アドレスを書き換えてやればよい．ただし，関数アドレスの書き換えは関数ポインタを持つ変数があって初めて可能なので，この場合は不可能である．今回，これを実現するために Advanced Windows [2]に記載されている API フックというものを参考にした．これは DLL のインポートセクションのアドレスを書き換えることで API コールをフックすることができるようになるものである．今回の API フックの対象は kernel32.dll のエクスポート関数 CreateFileA である．フック前の挙動を図 2.2 に示す．

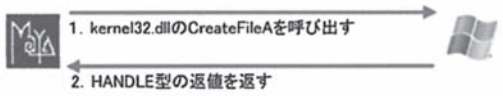

図 2.2　フック前の CreateFileA の挙動

Maya が API を呼び出すと kernel32.dll の CreateFileA が呼び出される．そして，引数に基づいた結果を HANDLE 型で返す．フック後の挙動は図 2.3 のようになる．ここで mdbg.dll というのは本研究で開発を行った DLL である．この DLL 内にある関数が呼び出されるようになる．

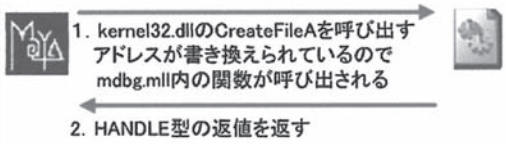

図 2.3　フック後の CreateFileA の挙動

2.10　API フック後の処理

API をフックした後，CreateFileA が呼び出されると，DLL 内の関数へ呼び出しが来る．その後，関数では CreateFileA で操作されるファイルが MEL スクリプトかどうかを調べ，MEL スクリプトだった場合，そのスクリプトをデバッグできるように，作成したコマンドを埋め込んでいく．そして，書き換えられた MEL スクリプトのコードを，一時的にテンポラリディレクトリへ作ったファイルに書き込み，そのファイルのハンドルを Maya へと返すようにする．このとき，ハンドルが閉じられたときにファイルを削除するように設定しておくことで，書き換えられたスクリプトのファイルが残ったり，後から削除する手間が省ける．このときの動作を図 2.4 に図示した．また，MEL スクリプト以外であれば通常通り CreateFileA を呼び出し，返値のハンドルを返す．このときの動作の様子を図 2.5 に図示した．

2.11　エラーが起きたときの処理

MEL スクリプトの実行中にエラーが起きた場合，MEL スクリプトは実行を中断する．このため，このままでは，コールスタックがエラーが起きたところで壊れてしまう．そのため，エラーが起きた場合

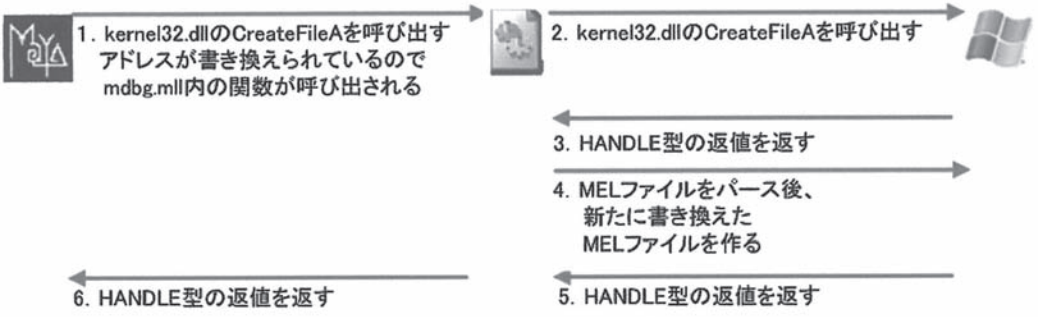

図 2.4　MEL ファイルの処理

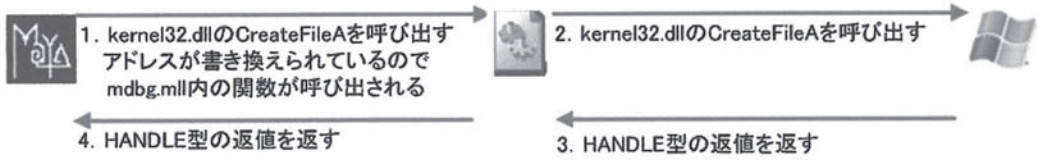

図 2.5　MEL ファイル以外の処理

は，そのことを検知し，コールスタックを破棄しなければならない．それを実現するために，MCommandMessage クラスの addCommandOutputCallback メンバ関数を使用した．これにより，スクリプトエディタに表示される各種情報をコールバックするように設定できる．この機能を使い，エラーのメッセージがコールバックされたときにはコールスタックを破棄するように機能させた．

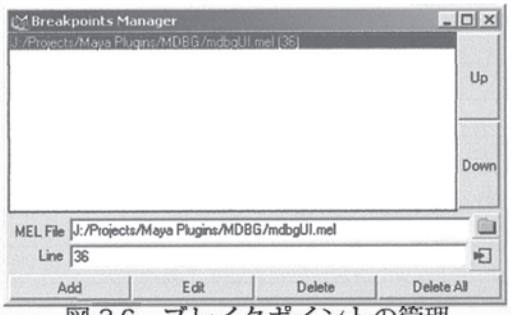

図 2.6　ブレイクポイントの管理

2.12　ブレイクポイントの設定

本研究で開発したデバッガでブレイクポイントを設定するには図 2.6 に示したようなウインドウを使い，設定を行う．設定にはファイルを指定した後，位置を指定することとなる．位置を指定する際には図 2.7 のようなウインドウを使い，グラフィカルに設定することができる．

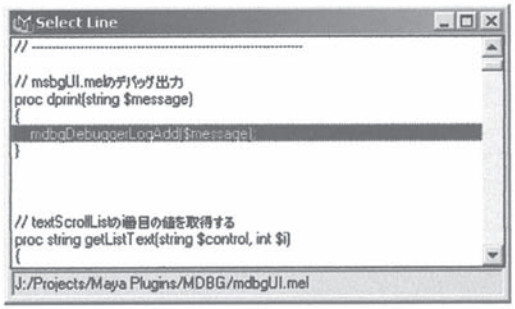

図 2.7　位置の選択

2.13　コールスタックの移動

コールスタックを表示するには図 2.8 のようなウインドウを使う．コールスタックを選択することでカレントのコールスタック位置を変更することができる．

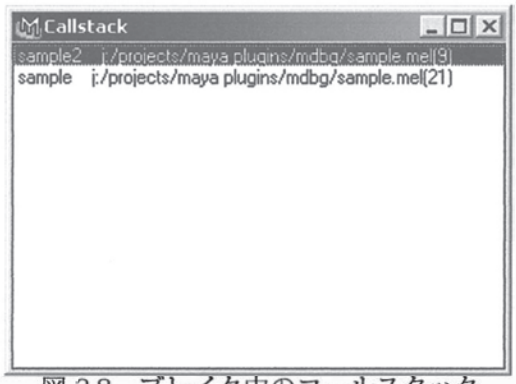

図 2.8　ブレイク中のコールスタック

2.14　ローカル変数

ブレイク中にローカル変数を表示するには図 2.9 のようなウィンドウを使う．このウインドウに表示される変数は，前述のコールスタックのウィンドウで選択されたコールスタックのスコープと連動する．

図 2.9　ブレイク中のローカル変数

2.15　グローバル変数

グローバル変数を参照するときは図 2.10 のウィンドウを使う．グローバル変数は数が多いため，上部のテキストボックスで変数名の絞り込みを行うことができるようになっている．

図 2.10　グローバル変数

3．まとめ

今回の研究で，MEL スクリプトのプログラムをデバッグすることが可能となった．また，これにより MEL スクリプトでの開発の時間が短縮できるようになった．今後の課題としては，ユーザーインターフェイスの拡充やブレイク中に Hypergraph, Graph Editor などといった一部の画面の描画がおかしいといった問題点の解決があげられる．

謝辞

今回の研究を行うに当たり，本学の尾形薫非常勤講師，および Web にて有用な情報を公開している田村公一氏に多大なアドバイスをいただきました．深く感謝いたします．

参考文献

1) DAVID A.D.GOULD, 中村達也訳：Complete Maya Programming, p.117, Born Digital, Inc. (2004)
2) Jeffery Richter, 長尾高弘訳：Advanced Windows, p.789, 株式会社アスキー (2001)