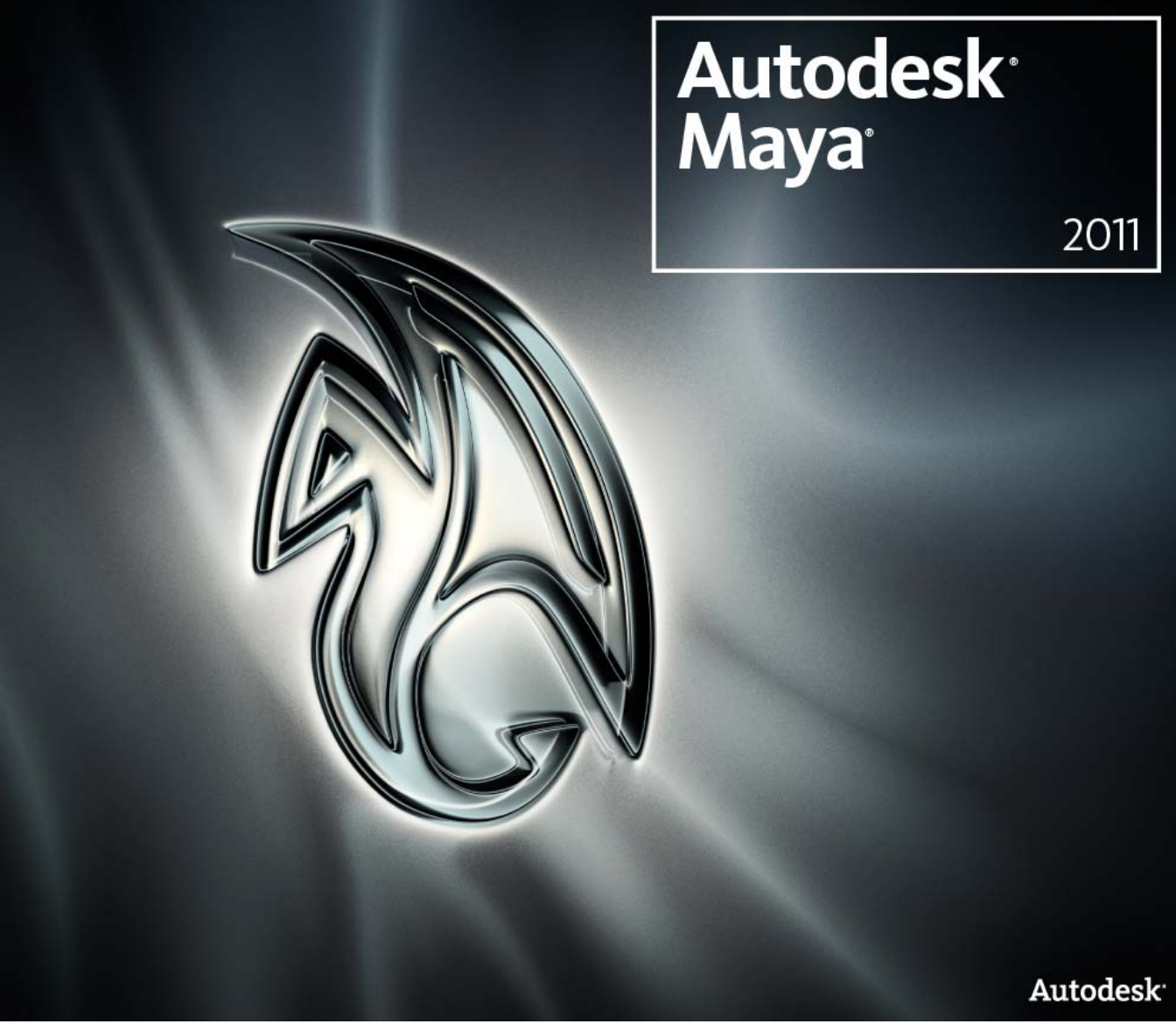


ファイル フォーマット

著作権の注意事項

目次

Maya ASCII ファイル フォーマット

1

ファイル フォーマットについて

本書では、プログラミング経験のある上級ユーザが次の作業を行うときに必要となる情報を提供します。

- Maya ファイルを編集する
- Maya ファイルとほかのファイル フォーマット間の変換を行うトランスレータ（コンバータ）を作成する

注: .pda ファイル フォーマットと .pdb ファイル フォーマットについては、コマンド リファレンスの「dynExport」を参照してください。

Maya ASCII ファイル フォーマット

Maya ASCII ファイル フォーマットの概要

Maya シーン ファイルでは、ジオメトリ、ライティング、アニメーション、レンダリングなど、シーンのさまざまな構成要素や属性を定義します。

Maya シーンは、バイナリ ファイルか ASCII ファイルとして保存されます。 ASCII ファイルは簡単に編集できます。

Maya の MEL® プログラミング言語を使用してスクリプトを作成されていれば、すでに Maya ASCII ファイル フォーマットのことはよくご存知です。Maya ASCII ファイルで使用されるのは、MEL 言語を構成する数百コマンドのうち、次の 11 種類に限られています。

- file
- requires
- createNode
- setAttr
- addAttr
- parent
- connectAttr
- disconnectAttr
- select
- currentUnit
- fileInfo

Maya ASCII ファイル内での動作が保証される MEL コマンドは上記に限られています。

MEL にあまり詳しくない場合は、作業の前にマニュアルに目を通しておいてください。詳細については、『MEL とエクスプレッション』マニュアルの「MEL の概要」を参照してください。

MEL スクリプト内の各ステートメントは、1 つのキーワードで始まり、一連のオプションと引数が続き、最後はセミコロンで終了します。1 つのステートメントは、ファイル内の複数の行にわたって記述することができます。

- キーワードは、常にステートメントの最初の単語になります。
- オプションは、ステートメントの特定の情報（属性）を指定します。
- 引数は、各オプションの内容を詳しく定義します。

この章の例では、次のタイプフェース（字体）を使用して、キーワード、オプション、引数を表しています。

キーワードとオプションは太字で表記します（例: **bump**、**-s**）。 オプションの前には常にハイフン文字（-）が付きます。

引数は斜体で表記します（例: u、file.txb）引数の名前は任意のラベルです。 スクリプト内では実際の数値または文字列を使用してください。

ファイル トランスレータを書き込む

Maya ASCII フォーマットに変換する

プログラムを作成して、別の種類のファイルを Maya ASCII ファイル フォーマットへ変換するのは比較的簡単です。主な作業は、外部データ型に相当する Maya のノードとアトリビュートを見つけることです。 サンプルの Maya ASCII ファイルの作成方法などプログラミングの詳細については、『ノード』リファレンスを参照してください。

注:

- Maya ASCII ファイルの最初の 6 文字は常に「//Maya」です。
- 作成されていないファイルは参照できません。

Maya ASCII フォーマットから変換する

プログラムを作成して、Maya ASCII ファイルを別のファイル フォーマットへ変換するのは困難です。変換処理を正しく実行するためには、Maya ASCII ファイルだけではなくそのファイル内で参照されているリファレンス ファイルも読み込む必要があります。 MEL の参照設定にはさまざまな MEL コードが含まれている可能性があります。それらの処理を独自に実行するコードを書くか、完全な MEL インタプリタを作成しなければなりません。

この問題をより簡単に解決する手段が、Maya Developer's Tool Kit で提供されています。このツールキットの MPxFileTranslator クラスには、ファイル トランスレータ用プラグインのサンプルや各種マニュアルが用意されています。 このマニュアルを使用して、サンプル（lepTranslator）をベースにすると、Maya ファイルを任意のフォーマットに変換するプラグインを効率的に作成できます。

MEL を使用してデータを書き出すこともできます。 この方法は規模の大きなトランスレータにはあまり向いていません。しかし、比較的小さくて単純な ASCII データのセットを簡単にすばやくエクスポートできます。 必要なデータが MEL

を使用してすべてアクセス可能ならば、fopen、fprint、fclose コマンドを使用して、データをファイルに書き込むことが出来ます。

埋め込みコメント

埋め込みコメントはMELスクリプト内の注釈テキストであり、スクリプト ファイルの読み込み時には無視されます。

スクリプト ファイル内の行に「//」（2つの連続したスラッシュ）が記載されている場合、その位置から行の末尾まではコメントとみなされます。 一般的に、これは「C++ スタイルのコメント」と呼ばれます。

スクリプト ファイル内の行に「/*」（1 つのスラッシュに続けて 1 つのアスタリスク）が記載されている場合、その位置から「*/」（1つのアスタリスクに続けて1つのスラッシュ）まではコメントとみなされます。一般的に、これは「C スタイルのコメント」と呼ばれます。

注: MEL では、C スタイルのコメントはサポートされていますが、通常はあまり使用されません。

Maya データを編成する

Maya ASCII ファイルの仕組みを理解するためには、Maya 内部でデータがどのように編成されるかを理解しておく必要があります。

シーンの情報はすべてノードに格納されます。ノードとは単にデータのブロックに名前を付けたものです。ノードの種類が違えば、データの種類も異なります。個々のデータは、ノードのアトリビュートと呼ばれます。同じ種類のノードはすべて、同じアトリビュート集合を持ちます。 ノードは固有の「ダイナミック」アトリビュートを持つ場合もあります。

Mayaのノードは次の2通りの方法で相互に接続する（関連付ける）ことができます。 1 つの方法はペアレント化（親子化: parenting）であり、関連のあるジオメトリをグループ化するために使用されます。親ノードを変換操作すると、すべての子ノードが親ノードと一緒に変換操作されます。 ペアレント化によって接続できるのは、ジオメトリを表すノード（カーブやサーフェスなど）と、それらのノードをグループ化するノード（transform ノードなど）に限られます。

たとえば、MayaのNURBS球体は、「nurbsSphere1」と「nurbsSphereShape1」という 2 つのノードで表現されます。このうち、「nurbsSphere1」ノードは

「transform」というノードのタイプです。このノードのアトリビュートには、移動、スケーリング、回転など、サイズと位置に関する情報が含まれます。「nurbsSphereShape1」ノードには、球体のシェイプに関する情報（制御点の正確な位置など）が格納されます。transform ノードはほかのノードの親ノードであるため、transform ノードのアトリビュートを設定すると、球体を変換操作できます。

ノードを一緒に接続するもう 1 つの方法は、アトリビュート接続と呼ばれます。双方のデータ型に互換性があれば、任意のノードの任意のアトリビュートを相互に接続できます。アトリビュートを接続した後、接続元のアトリビュートの値を変更すると、接続先のアトリビュートの値も変更されます。

Maya には何百種類ものノードがあり、プラグインを使用してさらに種類を追加できます。これらの単純な素材によって、Maya は精巧なモデルとアニメーションを表現することができます。

Maya で利用できる各種ノードとそのアトリビュートの詳細については、『DG ノード リファレンス（DG Node Reference）』のリスト（オンラインのみ）を参照してください。

Maya ASCII ファイルの構造

Maya が生成する ASCII ファイルは、以下の 8 つのセクションで構成されます。

各セクションについて次に説明します。ファイルを正しく読み込むためには、各セクションを上記の順序どおりに記述する必要があります。

ヘッダー

Maya ASCII ファイルのヘッダーには、ファイルの名前や作成日時などを示すコメントを記述します。 一般的なコメントと同様に、Maya ファイルを読み取るコードではヘッダーのコメントは無視されます。 ただし、1 つだけ異なる点があります。ヘッダーのはじめの 6 文字は、必ず「//Maya」でなければなりません。

Maya ASCII ファイルの一般的なヘッダーは次のようになります。

```
//Maya ASCII 1.0 scene
//Name: solstice.ma
//Last modified: Sun, Dec 21, 97 10:18:26 AM
```

リファレンス ファイルを設定する

ファイルの次のセクションは、プロシージャ以外のすべてのリファレンス ファイルを指定します。つまり、このセクションで参照されるプロシージャ以外の、すべての Maya ファイルを指定します。 現在のファイル内で参照する個々のファイルごとに、そのファイルを読み込むための 1 つの file コマンドを記述します。リファレンス ファイル内のすべてのオブジェクトは現在のファイル内で利用できるようになりますが、それらのオブジェクト名の前には接頭辞（通常はファイル名）が付きます。 ほかのファイルを参照するときの file コマンドのステートメントは次のとおりです。

```
file -r -rpr prefixString fileName;
```

または

```
file -r -ns nameSpace fileName
```

-r オプションは（引数で指定される）ファイルが参照されることを示し、-rpr オプションはリファレンス ファイル内の全オブジェクトに付加される接頭辞の文字列を指定します。file コマンドの詳細については、『MEL コマンド リファレンス』を参照してください。

例

```
file -r -rpr "solar" "/u/sally/work/solar.ma";
```

たとえば、ファイル「solar」に「sun」というオブジェクトが含まれている場合は、「solar_sun」という名前でそのオブジェクトを利用できます。

リファレンス ファイルの読み込みを遅延させる

リファレンスファイルの読み込みを遅延させる場合は、-drオプションを使用します。

必要条件

次のセクションは、必要条件を指定します。 指定するには、一連の requires コマンドを作成します。ファイルのこのセクションは、そのファイルを正しく読み込むのに必要な Maya ソフトウェアを指定します。すなわち、Maya のバージョン、プラグインなどです。

このセクションのステートメントは次のとおりです。

```
requires productName version
```

一般的な requires コマンドの例を下記に示します。

```
requires maya "2.0";
requires specialPlugIn "1.2";
```

（requires コマンドの詳細については、オンライン マニュアル『MEL コマンド リファレンス』を参照してください。）

単位

ファイルのこのセクションは、このファイルで使用されている単位を示す 1 つの currentUnit コマンドで構成されます。この指定によって、ファイル内の数値がどのように解釈されるかが決まります。

例:

```
currentUnit -l cm -a deg -t ntsc;
```

この例では、長さの単位をセンチメートルに設定し（他の選択肢はミリメートル、メートル、キロメートル、インチ、フィート、ヤード、マイル）、角度の単位を度に設定し（他の選択肢はラジアン）、時間の単位を NTSC に設定しています（currentUnit コマンドとそのオプションの詳細については、オンライン マニュアル『MELコマンド リファレンス』を参照してください）。

重要: 長さの単位をセンチメートルからフィート/マイルに変更した場合、予期せぬ結果になることがあります。

ファイル情報

次のセクションは、ファイルに関する情報を提供する複数の行から成ります。これらのうち最初の 5 行は、Maya によって次のように定義されます: アプリケーション名（Maya）、製品（Complete または Unlimited など）、バージョン、確認者（日付と時間）、OS とそのバージョン。ご使用のファイルに固有の fileInfo コマンドをファイルに追加すると、追加されたコマンドもこのセクションに表示されます。

```
fileInfo "application" "./maya.bin";
fileInfo "product" "Maya Unlimited 4.5";
fileInfo "version" "4.5";
fileInfo "cutIdentifier" "200111121041";
fileInfo "osv" "IRIX 6.5 04151556 IP32";
```

ファイルを保存するたびに、Maya はこれら 5 つの定義済み fileInfo ステートメントの現在の値を保存します。 ファイルをロードするときに見つかった、あるいは Maya のセッション中に発行された他の fileInfo ステートメント（あるいは同等のバイナリ）は、セッション中は保持され、ファイルの保存と同時に書き戻されます。

ノード、アトリビュート、ペアレント化

ファイルのこのセクションには、大量のデータが含まれます。

この部分に createNode コマンドで新しいノードを作成し、setAttr コマンドを使用して新しいノードのアトリビュートを設定します。

リファレンス ファイル内のノード（またはグローバルに定義されたノード）を選択するには select コマンドを使用し、それらのノードのアトリビュートを設定するには setAttr コマンドを使用します。

addAttr コマンドを使用すると、新しい「ダイナミック」アトリビュートをノードに追加することができます。

ノード間のペアレント化を設定するには、parent コマンドを使用します

（これらのコマンドの詳細については、オンライン マニュアル『MEL コマンド リファレンス』を参照してください）。

新しいノードを作成するステートメントは次のとおりです。

```
createNode nodeType -n nodeName;
```

このノードがペアレント化できるノード（すなわち、ジオメトリまたはジオメトリのグループを表す）であれば、すでに作成された親ノードがあります。ペアレント化はコマンドで指定することもできます。

```
createNode nodeType -n nodeName -p parentNodeName;
```

デフォルトのカメラなど一部のノードは、すべての Maya シーンで共通に使用されます。 そのようなノードを作成するときは、-s オプションを指定してください。このオプションを指定すると、同じ名前と親を持つノードが既に存在する場合には、新しいノードは作成されません。 このような状況は、他のファイルを参照しているときによく発生します。

新しいノードが作成されると、そのノードが自動的に選択されます。 通常、createNode コマンドを実行した後は、setAttr コマンドを何回か実行して新しいノードのアトリビュートを設定します。ノードが既に選択されているため、setAttr コマンドではアトリビュートの名前と値を指定するだけです（ノード名は不要です）。

setAttr コマンドのステートメントは次のとおりです。

```
setAttr attributeName value;
```

または

```
setAttr attributeName -type typeName value;
```

すべてのアトリビュートにはデフォルト値があるため、デフォルト以外の値を使用する場合、setAttr コマンドを実行してください（またはリファレンス ファイル内のアトリビュート値を変更する場合）。球体のノードを作成してアトリビュートを設定する例を下記に示します。

```
createNode transform -n "sphere";
 setAttr ".s" -type "double3" 2.44 2.44 2.44;
 setAttr ".t" -type "double3" -6.96 0 6.9;
createNode nurbsSurface -n "sphereShape" -p "sphere";
 setAttr ".tw" yes;
 setAttr ".rtw" yes;
 setAttr ".ipo" no;
```

リファレンス ファイル内で作成されたノードのアトリビュートを設定するときも、上記と同様の手順を実行します。 ただし、リファレンス ファイル内のノードは既に読み込まれている（作成済みである）ため、createNode コマンドの代わりに select コマンドを使用します。たとえば、「lunar」というファイルがオブジェクト「sun」を含む「solar」というファイルを参照しているとします。「lunar」というファイルでは、このオブジェクトのスケールが 3 に変ります。

これは、「lunar」という Maya ASCII ファイルではこのように見えるということです。

```
select -ne solar_sun;
 setAttr ".s" -type "double3" 3.0 3.0 3.0;
```

addAttr コマンドを使用すると、新しいダイナミック アトリビュートをノードに追加できます。 addAttr コマンドのステートメントは、setAttr コマンドと似ています。たとえば、新しい球体のノードを作成した後、-1～+1 の値を取る「squish」という浮動小数点型アトリビュートを球体に追加し、そのアトリビュート値を 0.3 に設定するとします。この場合は、次のようなコードを記述します。

```
createNode transform -n "sphere";
 addAttr -ci true -sn "squish" -ln "squish"
  -min -1 -max 1 -at "double";
 setAttr -k on ".squish";
 setAttr ".squish" 0.3;
// etc...
```

createNode コマンドの -p オプションでは、ノード間の階層構造を簡単に設定できない場合があります。 たとえば、リファレンス ファイル内の他のノードとの階層構造を設定するような場合です。 そのような場合は、次の parent コマンドを使用して階層構造を設定します。

```
parent childNodeName parentNodeName;
```

parent コマンドを使用して、ノードをインスタンス化することもできます（すなわち、2 つのノードの間で子ノードを共有させたい場合）。 その場合は、次のように -add オプションを使用します。

```
parent -add childNodeName parentNodeName;
```

特殊なケースとして、最上位レベルのオブジェクト（world とも呼ばれます）をインスタンス化する場合は、-w オプションと一緒に -add オプションを使用します。

```
parent -w -add childNodeName;
```

（親ノードはすべてのノードを包含する world であるため、親ノードを指定する必要はありません。）

スクリプト ノード

スクリプト ノードは MEL コードを、シーン ファイルの一部として保持します。スクリプト ノードはまた、ファイルからロードされた後、ファイルを閉じる前、ノードが削除される直前などに実行されるようセットアップされます。「before」

スクリプトはファイルがロードされたときに実行されます。「after」スクリプトは、ファイルが閉じられたとき、またはノードが削除されたときに実行されます。 シーンがいくつかのスクリプト ノードを含んでいる場合、保証されている実行順序はありません。つまり、スクリプトは特定の実行順序に依存してはなりません。スクリプト ノードを作成する方法の詳細については、『MEL コマンド リファレンス』の「scriptNode」の項と『ノード & アトリビュート リファレンス』（英語）のスクリプト ノードの項を参照してください。

MEL を使用すると、Maya 内のファイルのほぼすべての機能を実行できるため、スクリプト ノードの可能性は無限です。 とくに、スクリプト ノードはシーンのクリーンアップや UI のカスタマイズなどに便利です。

アトリビュート接続を解除する

リファレンス ファイルを含むファイルでは、このセクションで、次のようなステートメントの disconnectAttr コマンドを使用して、リファレンス ファイル内のノードとのアトリビュート接続を解除します。 これは、次のようなステートメントの 1 つ以上の disconnectAttr コマンドを使って実行します。

```
disconnectAttr sourceAttributeName destinationAttributeName;
```

例:

```
disconnectAttr "sphere.tx" "cone.ry";
```

disconnectAttr コマンドの詳細については、オンライン マニュアル『MELコマンドリファレンス』を参照してください。

アトリビュート接続を設定する

Maya ASCII ファイルの次のセクションは、作成および参照されたすべてのノード間のアトリビュート接続を確立します。 これは、connectAttr コマンドを使用して実行されます（connectAttr コマンドの詳細については、オンライン マニュアル『MEL コマンド リファレンス』を参照してください）。

コマンドのフォーマットは次のとおりです。

```
connectAttr sourceAttributeName destinationAttributeName;
```

例:

```
connectAttr "sphere.tx" "cone.ry";
connectAttr "sphere.squish" "sphere.sz";
```

Maya ASCII ファイルの編集に関する制限事項

Maya ASCII ファイルでは上記で説明したステートメント（MEL コマンド）だけがサポートされます。

注: Maya ASCII ファイルは、編集や MEL コマンドの追加を行うことができます。ただし、下記の理由から、ファイルの編集やコマンドの追加は積極的にはお勧めできません。

独自の MEL コードを Maya ASCII ファイルに直接追加する場合は、次の点に留意してください。

- Maya ASCII ファイルを編集して任意の MEL コマンドを追加する場合、ファイルの読み込み後、追加されたコマンドは保存されません。
  Maya ASCII ファイルに直接追加した MEL コマンドのテキストは、追加したコマンドを実行して作成された構成要素の記述（ノード、アトリビュート、接続などの記述）で置き換えられます。コマンドのテキストをそのまま保持したい場合は、拡張子 .MEL のスク リプト ファイルにコマンドを記述し、**ファイル（File）メニューのリファレンスの作成...（Create Reference...）**または file -r コマンドでそのファイルを参照してください。

- 任意の MEL コマンドを Maya ASCII ファイルに直接追加すると、一部のコマンドが正常に機能しない可能性があります。
  すべてのコマンドが正しく実行されるという保証はありません。たとえば、パフォーマンス上の理由から、Maya ASCII ファイルまたはバイナリ ファイルの読み込み中は、すべての undo 操作が一時停止されます。コマンドの中には内部的に undo 機能に基づいて処理を行っているものがあります。そのようなコマンドは、（外部の）.MEL ファイルの内部からしか実行できません。.MEL ファイルの読み込み中は undo 操作が中断されません。

アニメーション カーブ 2

アニメーション カーブの概要

Mayaには別のファイルとの間でアニメーションのインポートとエクスポートを行うためのプラグインがあります。このプラグインを使用すると、シーン間で簡単にアニメーションをコピーできます。たとえば、大まかな外観の（軽いラフな）キャラクタを使ってアニメーションをつけ、本番用のキャラクタを含む別のファイルにコピーして最終的な仕上げ作業を行うことができます。

アニメーション カーブをインポートする

プラグインによる内部処理では、API クリップボートとの間でアニメーションのインポートとエクスポートが行われます。インポート/エクスポート処理では、MEL コマンドの copyKey と pasteKey がプラグインによって実行され、API クリップボートと Maya オブジェクト間でアニメーション カーブが転送されます。プラグインを使用しても、Mayaアニメーション用のクリップボードの内容は変更されません。

インポートでは、選択したオブジェクトにアニメーションを適用するときに使用される編集 > キー > キーのペースト（Edit > Keys > Paste Keys）と同じオプションを設定できます。

アニメーション カーブをインポートするには

1. animImportExport プラグインを読み込みます。
2. アニメート対象の Maya オブジェクトを選択します。

3 ファイル > インポート（File > Import）を選択します。

4 インポートする .anim ファイルを選択します。

アニメーション カーブをエクスポートする

アニメーション カーブをエクスポートする場合、MEL コマンドの copyKey でアニメーション キーが API クリップボードにコピーされます。次に、API クリップボードの内容がアニメーション（.anim）ファイルに書き出されます。ファイルのエクスポート処理には、上記のすべての操作が含まれます。

注: インポート/エクスポート処理では MEL コマンドが使用されるため、MEL コマンドの制限事項がプラグインにも適用されます。たとえば、アニメーション カーブがアトリビュートよりも上流にあるが、アニメーション カーブとアトリビュート間にブレンド ノードが存在する場合は、エクスポート処理を実行できません。

ファイル > 選択項目のエクスポート（File > Export Selection）オプション ウィンドウでは、選択したオブジェクトからアニメーションをエクスポートする場合に使用される編集 > キー > キーのコピー（Edit > Keys > Copy Keys）と同じオプションを設定できます。

アニメーションをエクスポートすると、API クリップボードの内容が新しいアニメーション カーブで置き換えられます。

アニメーション カーブをエクスポートするには

1 animImportExport プラグインを読み込みます。

2 エクスポートするアニメーションを含む Maya オブジェクトを選択します。

3 ファイル > すべてエクスポート（File > Export All）またはファイル > 選択項目のエクスポート（File > Export Selection） を選択します。

4 ファイルのタイプを animExport に設定します。

5 エクスポートする .anim ファイル名を入力または選択します。

アニメーション ファイルのフォーマット

Maya には、アニメーション カーブのエクスポートと編集用のファイル フォーマットが用意されています。このファイル フォーマットは、Maya API を使用しなくても外部アプリケーションを通じてアニメーションの読み取り/書き込みを簡単に行えるような様式で定義されています。

```
Anim File Format
// A description of the anim file format.
// August 16, 1998
//
// The .anim file format (version 1.0):
// // and # are both valid comment characters.
//
// All of the lines in the file that do not contain curly braces
// ('{' or '}') should end with a ';' After the ';' character, start
// a new line.
//
// The keywords and data are whitespace delimited.
//
// Version 1.1 changes:
// April 20, 1999
// new weighted keyword for animData
// new breakdown flag for keys
//
// The version of the file format. This is a required line.
//
animVersion string
// The Maya version. The string is the value of
MGlobal::mayaVersion()
mayaVersion string

// The following two lines are optional. If they are not included,
// the clipboard is set to the range defined by the anim curves
// contained in the clipboard.
//
// These are used by anim curves that have time inputs.
//
startTime [float] // The starting frame for the clipboard.
endTime [float] // The ending frame for the clipboard.
// The following two lines are optional. If they are not included,
// the clipboard is set to the range defined by the anim curves
// contained in the clipboard.
//
// These are used by anim curves with unitless inputs.
//
startUnitless [float] // The starting value for for the clipboard.
endUnitless [float] // The ending value for the clipboard.
// The following three keywords are used to set the units for the
 file.
```

```
// Each anim curve may have its own units, but these are the
default
// units if the anim curve units are not given (see the animData
section).
//
// If the units are not given, then the ui units are used.
//
timeUnit [game|film|pal|ntsc|show|palf|ntscf|hour|min|sec|millisec]
linearUnit [mm|cm|m|km|in|ft|yd|mi]
angularUnit [rad|deg|min|sec]
// All of the keywords described above can only be in the header
section
// of the file. As soon as anim curve information is encountered,
 the
// header section is completed and the body of the file is begun.
//
// The string is the name of the attribute the anim curve is
connected to.
// The next three ints are the row, child, and attr values used
by the
// clipboard. See the documentation for MAnimCurveClipboard for
more
// information.
//
// If the anim curve is not connected to any attributes, the string
// is not needed, but the following ints should be 0 0 0.
anim [string] [int] [int] [int]
// The second form of the anim line has three strings which list
what
// the anim curve was connected to.
//
// The strings are: the full attribute name, the leaf attribute
name,
// and the node name. The row, child, and attr ints are still
required.
//
anim [string] [string] [string] [int] [int] [int]
// The third and final form of the anim line is used for clipboard
// place holder objects. These are used to skip node which do not
// contain any anim curve data, but are positioned in a hierarchy

// with nodes that have attached anim curves.
//
```

```
// In this case, the string is the node name and the three ints
are the
// same as the other two formats.
//
anim [string] [int] [int] [int]
// The animData must follow a line with a valid anim statement.
//
animData {
 // The input type of the anim curve. Defaults to time.
 input [time|unitless]
 // The output type of the anim curve. Defaults to linear.
 output [time|linear|angular|unitless]
 // Whether or not the anim curve has weighted tangents. Defaults
 to false.
 // This is available with animVersion >= 1.1
 weighted [1|0]
 // The unit of the anim curve input, if it is a time input.
 // The units default to the time units specified in the file
header.
 inputUnit
[game|film|pal|ntsc|show|palf|ntscf|hour|min|sec|millisec]
 // The unit of the anim curve output. The output unit should match
 // the output type of the curve. These default to the units
specified
 // in the header.
 outputUnit
[game|film|pal|ntsc|show|palf|ntscf|hour|min|sec|millisec]
 outputUnit [mm|cm|m|km|in|ft|yd|mi]
 outputUnit [rad|deg|min|sec]

 // The unit of the tangent angles, if there are any fixed
tangents.
 // The units default to the angular units specified in the file
header.
 tangentAngleUnit [rad|deg|min|sec]

 // The pre-infinity type. Defaults to constant.
 preInfinity [constant|linear|cycle|cycleRelative|oscillate]

 // The post-infinity type. Defaults to constant.
 postInfinity [constant|linear|cycle|cycleRelative|oscillate]
 // The start of the actual keyframe data. Each key is a row in
the
```

```
 // braced section.
 keys {
 [float] [float] [in tan] [out tan] [tan locked] [weight locked]
 // animVersion 1.1 adds breakdown information
 [float] [float] [in tan] [out tan] [tan locked] [weight locked]
[breakdown]
 .
 .
 .
// The first two values are the input and output values in the
 // units defined by the inputUnit and outputUnit keywords.
 // The in and out tangents should be valid tangent types.
 // These are followed by three int values for tangent locking,
 // weight locking and the breakdown flag. If they are 0, the
values
 // are unlocked, or not a breakdown, otherwise they are locked.
 //
 // If either, or both, or the tangents are fixed, then additional

 // information is needed: a tangent angle and weight.
 // These two values, per fixed tangent, are added at the end of
 // the above line.
 //
 // For example:
 // 1.0 2.0 fixed linear 1 1 0 62.345 0.04;
 //
 // In the above case, 62.345 is the tangent angle for the first
 // tangent and the tangent weight is 0.04.
 //
 // An example with two fixed tangents:
 // 1.0 2.0 fixed fixed 1 1 0 62.345 0.04 45.3 0.023;
 }
}
```

The pattern of an anim line followed by animData should be used until all of the anim curves are described.

The following example is an animated joint chain consisting of 4 joints. The first three joints are animated and the fourth joint is not animated.

```
animVersion 1.1;
mayaVersion 2.0;
timeUnit ntsc;
linearUnit cm;
angularUnit deg;
```

```
startTime 1;
endTime 30;
anim rotate.rotateX rotateX joint1 0 1 0;
animData {
 input time;
 output angular;
 weighted 0;
 preInfinity constant;
 postInfinity constant;
 keys {
 1 0 linear linear 1 1 0;
 30 0 linear linear 1 1 0;
 }
}
anim rotate.rotateY rotateY joint1 0 1 1;
animData {
 input time;
 output angular;
 weighted 0;
 preInfinity constant;
 postInfinity constant;
 keys {
 1 0 linear linear 1 1 0;
 30 0 linear linear 1 1 0;
 }
}
anim rotate.rotateZ rotateZ joint1 0 1 2;
animData {
 input time;
 output angular;
 weighted 0;
 preInfinity constant;
 postInfinity constant;
 keys {
 1 0 spline spline 1 1 0;
 10 -16.774359 spline spline 1 1 0;
 15 -1.6493069 spline spline 1 1 0;
 22 -3.064691 spline spline 1 1 0;
 30 0 spline spline 1 1 0;
 }
}
anim rotate.rotateX rotateX joint2 1 1 0;
animData {
```

```
 input time;
 output angular;
 weighted 0;
 preInfinity constant;
 postInfinity constant;
 keys {
 1 0 linear linear 1 1 0;
 30 0 linear linear 1 1 0;
 }
}
anim rotate.rotateZ rotateZ joint2 1 1 1;
animData {
 input time;
 output angular;
 weighted 0;
 preInfinity constant;
 postInfinity constant;
 keys {
 1 0 spline spline 1 1 0;
 10 60.962438 spline spline 1 1 0;
 15 106.06094 spline spline 1 1 0;
 22 33.259896 spline spline 1 1 0;
 30 0 spline spline 1 1 0;
 }
}
anim rotate.rotateX rotateX joint3 2 1 0;
animData {
 input time;
 output angular;
 weighted 0;
 preInfinity constant;
 postInfinity constant;
 keys {
 1 0 spline spline 1 1 0;
 10 0 spline spline 1 1 0;
 15 0 spline spline 1 1 0;
 22 0 spline spline 1 1 0;
 30 0 spline spline 1 1 0;
 }
}
anim rotate.rotateY rotateY joint3 2 1 1;
animData {
 input time;
```

```
 output angular;
 weighted 0;
 preInfinity constant;
 postInfinity constant;
 keys {
 1 0 spline spline 1 1 0;
 10 0 spline spline 1 1 0;
 15 0 spline spline 1 1 0;
 22 0 spline spline 1 1 0;
 30 0 spline spline 1 1 0;
 }
}
anim rotate.rotateZ rotateZ joint3 2 1 2;
animData {
 input time;
 output angular;
 weighted 0;
 preInfinity constant;
 postInfinity constant;
 keys {
 1 0 spline spline 1 1 0;
 10 0 spline spline 1 1 0;
 15 0 spline spline 1 1 0;
 22 0 spline spline 1 1 0;
 30 0 spline spline 1 1 0;
 }
}
anim joint4 3 0 0;
```

Maya イメージ ファイル フォーマット（IFF）

3

Maya IFF の概要

Maya では、イメージは IFF（Interchange File Format）というフォーマットで保存されます。IFF は汎用の構造化ファイル（structured file）アクセス機構であり、イメージの保存以外の用途にも使われます。たとえば、AIFF（Audio Interchange File Format）はオーディオ データ用の IFF です。IFF イメージでは、RGB の各チャンネル（およびオプションのアルファ チャンネル）ごとに 8 ビットまたは 16 ビットを使用できます。また、オプションで 32 ビット浮動小数点型のデプス マップ（Z バッファ）も含めることができます。

解像度の制限事項

次の表に、各イメージ ファイルの解像度に関する制限事項を示します。

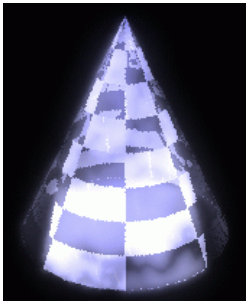
イメージ ファイル タイプ	解像度の制限
ファイル テクスチャ	8192 x 8192
イメージ プレーン	8192 x 8192
インタラクティブにレンダーしたイメージ	4096 x 4096
バッチ レンダーしたイメージ	8192 x 8192
BOT ファイルへ変換	8192 x 8192

Maya で IFF ファイルを生成する

レンダーしたイメージは IFF ファイルとして保存することができます。デフォルトでは、これらの IFF ファイルには、デプス マップを含まない RGBA データが保存されます。例:

レンダーしたイメージ

1. シーン（cone.ma など）を読み込みます。
2. **レンダリング**（**Rendering**）メニュー セットに切り替え、**レンダー > カレント フレームのレンダー**（**Render > Render Current Frame**）を選択します。
3. レンダリングが完了したら、右マウス ボタンで**イメージ**をクリックし、**ファイル > イメージの保存...**（**File > Save Image...**）を選択します。
4. ファイルブラウザにファイル名を入力します。cone.iff などの出力イメージを表示します。



シャドウ マップ

Maya の光源は、シャドウ マップの読み込みと書き込みができます。シャドウ マップは、IFF ファイル内ではデプス マップとして保存されます。デプス マップは従来の Z バッファまたはミップマップです。ミップマップの格納された値は最初のサーフェスと 2 番目のサーフェス間の距離の半分です。ミップマップには、不正に発生するセルフ シャドウを抑える効果があります。

生成されるシャドウ マップは、光源のタイプに応じて異なります。広角が 90 度未満のスポット ライトでは、シャドウ マップが 1 個生成されます。広角が 90 度以上のスポット ライトでは、5 個のシャドウ マップが生成されます。ポイン

ト ライトを使用すると、単純なシーン 1 つにつき 1 つのシャドウ マップが生成され、より複雑なシーンでは最大 6 個のシャドウ マップが生成されます。

次の例を試してみてください。

1. ライト（光源）を含むシーン（torusShadow.ma など）を読み込みます。
2. ライトを選択して、**アトリビュート エディタ**（**Attribute Editor**）を開きます。
3. アトリビュート エディタの**シャドウ**（**Shadows**）セクションを展開します。
4. **デプス マップ シャドウ アトリビュート**（**Depth Map Shadow Attributes**）サブセクションの**デプス マップ シャドウの使用**（**Use Depth Map Shadows**）チェック ボックスをオンに設定します。
5. ミッドマップを使用する場合は、同じセクションの**中間距離のデプス マップの使用**（**Use Mid Dist Dmap**）チェック ボックスをオンに設定します。
6. 同じセクションの**シャドウ マップの書き込み**（**Write shadow map**）チェック ボックスをオンに設定します。**ダンプの名前**（**Dmap name**）フィールドにファイルのプリフィックスを入力します。
7. シーンをレンダーします。
8. ここで、手順 6で選択したプリフィックスで始まるファイル* が 1 つなければなりません。拡張子は、光源の名前、生成されたシャドウ マップのタイプになります。torusShadow の例では、ステップ 5 での選択に応じて、shadowmap_pointLightShape1_int.SM.iff または shadowmap_pointLightShape1_int.MIDMAP.SM.iff のいずれかのイメージ ファイルが作成されます。

IFF イメージを表示、変換する

Maya に含まれる fcheck というスタンドアローン型プログラムで、IFF イメージとデプス マップを表示することができます。imgcvt プログラムは、IFF ファイルを別のフォーマットに変換することができます（Linux と Windows のみ）。Linux と Windows の場合、imgcvt プログラムを使用して IFF ファイルを別のフォーマットに変換することができます。

ILライブラリとFLライブラリでプログラミングする

カスタム ソフトウェアで IFF イメージを使用するには、IL ライブラリと FL ライブラリの使用をお勧めします。FL ライブラリは、IFF ファイルを扱うための汎用的なライブラリです。ほとんどの場合、IFF ファイルを手作業で直接操作する必要はありません。IL ライブラリは FL ライブラリに基づいて IFF イメージ用に特化されているため、効率的に読み書きできます。

IL ライブラリには IFF ファイルのフォーマットや内容が記述されています。イメージ データを取り出すフォーマットを指定するだけで、その他の必要な操作はライブラリを通じて実行されます。ライブラリの使用方法は極めて簡単です。Maya Developer's Tool Kit に用意されたサンプル プログラム（下記参照）を見れば、必要な情報はほとんど得られます。

注: スタンドアローン型のプログラムを作成する場合は、コンパイル/リンク時に IL ライブラリと FL ライブラリを直接リンクしてください。Maya にはこれらのライブラリが組み込まれているため、Maya プラグインについては、特別なライブラリをリンクする必要はありません。

サンプル プログラムを使用する

下記のサンプル プログラムとその説明については、開発者向けリソースの『API ガイド』マニュアルにある「サンプル プラグイン」の「その他のプラグイン」を参照してください。

iffInfoCmd: イメージに関する情報を取得するプラグイン（iffreader が必要）

iffPixelCmd: イメージのピクセルすべての値を取得するプラグイン（iffreader が必要）

iffPpmCmd: IFF から PPM に変換するプラグイン（iffreader が必要）

その他の参考情報

IL と FL のマニュアル ページ

IL と FL のマニュアル ページでは、IFF ファイルを操作するためにプラグイン内部からコールできる各関数について説明しています。詳細については、**IFFmanPages** を参照してください。

fcheck のマニュアル ページ

Maya に付属の fcheck プログラムを使用すると、IFF ファイルを直接表示できます。詳細については、『レンダリング ユーティリティ』マニュアルの「FCheck の概要」を参照してください。

imgcvt のマニュアル ページ（Linux と Windows のみ）

Maya に付属の imgcvt プログラムを使用すると、IFF ファイルをほかのフォーマットに変換できます。詳細については、『レンダリング ユーティリティ』マニュアルの「imgcvt」を参照してください。

IFF フォーマットの概要

flib ライブラリの簡単な説明を次に記します。ここでは、IFF フォーマットに基づく汎用的な構造化ファイル アクセス機構を実装する flib ライブラリについて簡単に説明します。IFF は現在 Maya のイメージとテクスチャ用に使用されているフォーマットです。

カーネル

ファイル タイプからの独立性

flib ライブラリの基本的な概念は、すべてのファイル アクセスを同じ方式で表現することです。ディスク ファイル、パイプ、メモリ セグメントなどが論理的にはすべてファイルとして表され、同じ関数集合を通じて操作されます。flib カーネルは、8 つの関数（FLopen、FLreopen、FLclose、FLread、FLwrite、FLseek、FLtell、FLflush）から構成されています。これらのロー レベルの IO ルーチンを libc ライブラリ ルーチン（open、fopen、read、fread、write、fwrite...）の代わりに使用することができます。

flib ライブラリのもう 1 つの利点は、ファイルのオープン モードによる特定の制限がないことです。たとえば、読み取りモードで開かれたパイプにも（例: 「pipe:cat file」）書き込むことが可能です。

FLopen 関数では、次の命名規則に従って、操作対象とする論理ファイルのタイプが判定されます（open、popen、fopen、socket などに応じて別々の方式を指定する必要はありません）。現在のところ、認識されるファイル名の形式は次のとおりです。

ファイル名	説明
name	通常のディスク ファイル
name.Z	圧縮されたファイル
mmap:name	メモリ マップされたファイル
pipe:cmd [args]	cmd の標準入力（出力）
fd:#	ファイル記述子番号（# = 0:stdin、1:stdout、2:stderr）
stdin, stdout, stderr	ファイル記述子 0,1,2 の別名
host:name	リモート ホスト上のファイル
user@host:name	リモート ホスト上のファイル（ユーザ アカウントを通じてアクセスされる）
mem:addr	アドレス addr にあるメモリ セグメント

開いたファイル オブジェクトの特性に応じて、特定の制限が課せられる場合があります（たとえば、パイプに対しては FLseek を実行できません）。

各ファイルは必要に応じてバッファリングされます。メモリ内の移動を最小限に抑えると、それに比例して転送速度が上がります。FLbgnread、FLendread、FLbgnwrite と FLendwrite の各関数を使用すると、ライブラリ内の読み取り/書き込みバッファに直接アクセスできます。メモリ マップされたファイルでこれらの関数を使用すると特に効率的です。

データ フォーマットからの独立性

flib ライブラリに基づくファイル アクセスでは、FLfilter 関数が使用されており、論理ファイルが外部フィルタに渡された上で実際の読み取りや書き込みが行われます。このため、ファイルに保存されたデータのフォーマットをほとんど気にすることなく、ファイルを操作できます。

コントロール

flib ライブラリには、C 言語の標準 IO 関数とシステム コールの場合と同様のエラー コントロールが組み込まれています。エラーが発生するとステータス変数 flerror の値が変更されますが、その変数値を調べるために複数の関数（FLerror、FLseterror、FLperror、FLstrerror、FLoserror、FLsetoserror）が用意されています。flib ライブラリで扱われるエラー集合は、Linux の標準的なエラー（errno、strerror、h_errno、およびシステムでサポートされている場合は hstrerror）を拡張したものです。

flib ライブラリの IO 関数はエラーが連鎖的に発生しないように設計されていますが、エラーが一度発生した後は、読み取り/書き込み処理を続行しないことを強くお勧めします。

FLconfig 関数を使用すると、flib ライブラリの特定のパラメータ（テンポラリ ファイルの作成、マッピング、自動圧縮/解凍など）を変更できます。

読み取りアクセス時のファイル名の解決に使用されるパスの定義または追加を行うときは、FLsetpath 関数または FLaddpath 関数をコールします。

FLswitchpath 関数（頻繁に行われるパス変更を最適化する関数）によってアクティブにされるパスの設定または解除を行うときは、FLbuildpath 関数または FLfreepath 関数をコールします。FLsetreorder 関数を使用すると、パスの検索を最適化できます。

構造化ファイル（structured files）

flib には IFF から派生したファイル構造の規則があります。flib のファイル構造ではタグを使用して、「チャンク」と呼ばれるデータのブロックおよび「グループ」と呼ばれるチャンクの構造体を定義します。個々のタグは最大 4 文字からなり、その後にチャンクまたはグループのサイズを表す数値（4 バイトの整数）が続きます。この構造は IFF（Interchange File Format）を少しだけ拡張したものです。すべてのデータは big endian フォーマットで記述されますが、タグは疑似文字列として取り扱われます （バイト交換はコンパイル時に処理されます）。

ブロックのサイズを明示的に指定すると、FLparse 関数など IFF パーサーで認識できないデータはスキップされます。

タグには、ファイル構造（グループ）を定義するタグと、データを含むタグの 2 種類があります。

グループ

ブロックは次の 4 つのタグを使用してグループに編成されます: FORM、CAT、LIST、PROP。サイズの後に続く最初の 4 文字は、グループのタイプを指定するのに使用されます。

FORM タグは、C 言語の struct と似たステートメントによって、グループの始まりを定義します。

```
FORM 38 TEXT
 CHAR 6 "Times"
 CHAR 12 "Hello World"
EOF
```

上記のコードは次の C コードと同じ意味です。

```
struct Text t = {
 char *f = "Times";
 char *c = "Hello World";
};
```

グループのサイズ（38）は、グループ内のデータのサイズ（6 + 12）に、ヘッダーのサイズ（TEXT 用の 4、CHAR 6 用の 8、CHAR 12 用の 8）を加えた合計値です（この例の場合は、6+12+4+8+8 = 38）。

C 言語のステートメントと同じく、次の例のようにグループをネストできます（ネストとは、入れ子にすることです）。

```
FORM 52 TEXT
 FORM 8 FONT
  CHAR 6 "Times"
  LONG 4 <12>
  LONG 4 <0>
 CHAR 12 "Hello World"
EOF
```

上記のコードは次の C コードと同じ意味です。

```
struct Text t = {
 struct Font f = {
  char *n = "Times";
  int s = 12;
  int d = 0;
 };
 char *string = "Hello World";
};
```

C 言語のステートメントとまったく同様に、データ型の異なる複数のブロックを同じグループに入れることができます。FORM タグの役割は、別々に取り扱える独立したチャンクの集合（グループ）を分離して、各グループの意味を明確に表すことです。上記の例では、FONT FORM 内の CHAR チャンクと TEXT FORM 内の CHAR チャンクは意味が異なります。すなわち、FORM タグを使用することによって、特定の順序関係を持つチャンクの集合をどのように解釈するかが決まります。

CAT タグは、相互に特定の順序関係を持たない独立したグループを連結します。CAT の主な使用目的は、グループのライブラリ（最初の例の PICT）またはクリップボード（2 番目の例の CLIP）を定義することです。

```
CAT 3632 PICT
 FORM 1234 PICT ...
 FORM 2378 PICT ...
EOF
CAT 2130 CLIP
 FORM 1234 PICT ...
 FORM 876 DRAW ...
EOF
```

通常、構造化されたファイル内部での検索は、（メンバーのグループ間に特定の順序関係が存在しない CAT の場合でも）かなり高速になります。これは、ファイルのヘッダーで各グループやチャンクのサイズが指定されているためです。

LIST タグは、特定の順序関係を持つグループ（FORM データ ブロック）の集合を定義します。また、PROP タグと一緒に使用すると、同様の特性を持つグループの集合を簡略に定義できるため、冗長性が排除されます。たとえば、同じサイズの一連のイメージは下記のように表現されます。

個々のイメージの構造は次のとおりです。

```
FORM .... PICT
 IHDR 32 [image size info]
 BODY ... [image data]
EOF
```

上記と同じサイズの一連のイメージは、ヘッダー情報が同じなので、次のように定義できます。

```
LIST ... ANIM
 PROP 44 PICT
  IHDR 32
 FORM ... PICT
  BODY ....
 FORM ... PICT
  BODY ....
 FORM ... PICT
  BODY ....
EOF
```

PROP タグで指定された情報は LIST タグの終わり（EOF）まで有効であり、LIST 内部のすべての FORM タグに適用されます。ただし、C 言語のローカル変数の場合と同様に、個々の FORM タグ内部で PROP 情報を再定義することもできます。上記の例では、独自の IHDR ブロックを持たないすべての PICT に対して、PROP タグ内の IHDR ブロックの情報が適用されます。

データ ブロック

データ ブロックは以下で定義されます。

[tag] [size] [data]

例: 単一のイメージの構造は以下のとおりです。

```
FORM 12304 IMAG
 IHDR 200 ... picture header, size, maps ...
 LINE 800 ... data from line 1 ...
 LINE 800 ... data from line 2 ...
 ...
```

ライブラリの定義

```
CAT 64200 IMAG
 FORM 12304 IMAG
  IHDR 200
  ...
 FORM 12304 IMAG
  ...
```

同じ特性を持つ一連のイメージの定義

```
LIST 64200 IMAG
 PROP 208 IMAG
  IHDR 200  ... Common header ...
 FORM 12394 IMAG
 ...
 FORM 12304 IMAG
  IHDR 200  ... Local redefinition ...
  ...
```

メモリ境界に整列する

IFF ブロックは 2 バイトの境界に整列します。ただし、ヘッダーで指定されるサイズではパディングが考慮されていません。一般的に、最近のコンピュータのメモリでは 4 バイトまたは 8 バイトの境界にデータが整列されます。flib ライブラリでは、メモリ境界への整列を指定できるように 8 個の専用タグが用意されています。そのうち、4 個のタグ（FOR4、CAT4、LIS4、PRO4）は 4 バイトの境界への整列を指定するもので、残りの 4 個のタグ（FOR8、CAT8、LIS8、PRO8）は 8 バイトの境界への整列を指定するものです。

データ ブロックは、所属するグループの整列条件を継承します（サブグループを含む。したがって、4 バイトの境界に整列するグループ内に、2 バイトの境界に整列するグループを作成することはできません。その逆は可能です）。

拡張機能（特殊なブロック サイズの指定）

IFF の主要な制限事項の 1 つとして、チャンクまたはグループをファイルに書き込む前に、そのチャンクまたはグループのサイズが分かっていなければならないという点が挙げられます。したがって、ブロックの情報を変更した場合は、ファイル構造の正確なサイズをヘッダーに反映させる必要があります。シーク操作が可能なファイル（ディスク ファイルとメモリ）の場合、この制限事項は特に問題になりませんが、それ以外のファイルについては不都合が発生する場合があります。このため、flib ライブラリでは、ブロックのサイズが不明であることをユーザが事前に指定できる仕組みが用意されています。負の値のブロック サイズが適用されないので、2 つの特別な値がかわりに使われます。FL_szFile は、一度グループが完全に書き込まれた後でサイズを書き込むことを指定する値です。FL_szFifo は、ファイルのシーク操作ができない時にサイズを書き込まないようにする値です。また、構造の終わりを示すのに、特別なゼロ サイズのブロック（GEND）が使われます。

関数

ブロックは、FLgetchunk と FLputchunk のコールを使って読み取りと書き込みを行うことができます。FLbgnget 関数と FLbgnput 関数をコールしてブロックを開くと、より直接的にブロックを操作できます。FLput 関数と FLget 関数では、FLread および FLwrite と同等の機能をブロック内部で利用できます。FLput または FLget を何回かコールした後でブロックを閉じるには、FLendput 関数または FLendget 関数を使用します。

グループを操作するには、FLbgnrgroup、FLbgnWgroup、FLendrgroup、FLendwgroup の各関数を使用します。flib ライブラリには汎用ファイル パーサー FLparse も用意されています。FLparse 関数を使用すると、ファイルのスキャン（解析）や整合性をチェックできます。また、ファイル解析の各ステップ（グループの先頭、グループの終わり）でコールされるコールバック関数の指定もできます。

ツール ボックス（Toolbox）

flib ライブラリには、リンク リスト（FLxxxnode と FLxxxlist）、バッファ（FLmalloc と al）、外部フィルタ（FLfilter と FLexec）を操作するためのツールが用意されています。

イメージ ライブラリの概要

IO イメージ ライブラリは flib ライブラリの一部です。IO イメージ ライブラリのルーチンを使用すると、構造化ファイル内のイメージの読み取りと書き込みを行うことができます。

ファイル フォーマット

イメージ ファイルのフォーマットは非常に柔軟性に富んでいます。ブロックの数や相対位置に関する制限はごく少数しかありません。通常、それらの制限は純粋に論理的なものです。たとえば、「ヘッダーはオフセット 124 から始まる」といった物理的な制限ではなく、「ヘッダーはピクセル ブロックよりも前にくる」といった論理的な制限です。

イメージ ファイルは flib ライブラリの主要な操作対象であり、さまざまなブロックをイメージ ファイルに記述できます。最小限のイメージは、CIMG タイプの

FOR4 グループ（4 バイトのワード境界に整列されるグループ）から構成されます。この FOR4 グループの個々の構成要素は下記の順序で記述します。

- BMHD ヘッダー（ビットマップのヘッダー）
- TBMP タイプの FOR4 グループ（タイル状に配置されるビットマップの情報）

ピクセル情報は TBMP タイプの FOR4 グループに保存され、場合によってはすぐにスキップされます。

TBMP タイプの FOR4 グループでは、各ピクセルを表すデータ ブロックを特定の順序で記述します。たとえば、4 個のタイルに分割されるイメージは次のように表現されます。

```
FOR4 <size> CIMG
 BMHD 24 ... definition of size, maps, etc...
 FOR4 <size2> TBMP
  RGBA <ttile1> ... tile 1 pixels ...
  RGBA <ttile2> ... tile 2 pixels ...
  RGBA <ttile3> ... tile 3 pixels ...
  RGBA <ttile4> ... tile 4 pixels ...
```

ヘッダーは ILheader 構造によって定義されます。RGBA ブロックの構造は次のとおりです。

[x1, y1, x2, y2]: タイル座標（それぞれ 2 バイト）

[pixels]: 圧縮モードに従ってエンコードされたピクセル情報

イメージに Z バッファが含まれている場合は、RGBA ブロックと同じ構造を持つ ZBUF ブロックを使用して Z バッファ情報（エンコード方式は RLE）を記述します。

- End フィールド: このフィールドはイメージ データの終わりを示し、fcheck ユーティリティによって表示されます。イメージに関する情報を表示してチェックするには、イメージを読み込んだ後で flib ライブラリ関数をコールします（ILIB ライブラリでは End フィールドが無視されます）。イメージに関する情報は次の 4 個のフィールドによって表されます。

HIST: イメージを作成した Maya コマンドラインを表す文字列データ。

VERS: Maya バージョン情報を表す文字列データ。

CLPZ: 使用されたクリッピング プレーンを表すデプス マップ固有のフィールド。2 つの浮動小数点値として格納される。

ESXY: 視点の x-y 比。デプス マップに固有のフィールドで、2 つの浮動小数点値として格納された正規化ピクセル座標から XY 視点座標を計算するのに使われる。

注: ILIB ライブラリでは End フィールドがサポートされません。

関数

関数によっては、タイルの順序を気にせずに個別ライン モードでイメージの読み取りと書き込みができます。イメージの読み取り中に自動的にズームや補正を行う関数もあります（圧縮イメージに対する補正処理の性能が大幅に改善されました）。また、フォーマット変換関数を使用すると、イメージの読み取り時にフォーマットを自動的に変換できます。

イメージ ライブラリ内の関数の詳細については、各関数のマニュアル ページ、ライブラリに付属の「include」ファイルとサンプル プログラムなどを参照してください。

IFF フォーマットの拡張子

PATH は include の検索パスを定義します。

INCL は include ブロックです。

EOVC は可変長チャンクの終わりを示すマーカーです。

GEND はグループの終わりを示すマーカーです。

構文:

```
PATH # <directory names>
INCL # <file names>
EOVC szUnknown
GEND 0
```

PATH と INCL はデータ ブロック（チャンク）であるため、メモリ境界の整合情報をグループから継承します。

EOVC と GEND は fifo ファイル（パイプなど）で使用されます。fifo でサイズを指定せずにグループやチャンクを作成すると、次のようになります。

```
FORM sz_Fifo TYPE ; start of form
 BLCK sz_Fifo data ; block 1
 EOVC sz_Unknown  ; end of block 1
 ....   ;
 BLCK sz_Fifo data ; block 2
 EOVC sz_Unknown  ; end of block 2
 GEND 0  ; no more block in this FORM
```

EOVC は単に可変長チャンクの終わりを表すマーカーですが、GEND は現在のグループを閉じるリクエストと解釈されるため、1つ上のレベルに移動します。

シーク可能なファイルを書き込むときにブロックのサイズを指定しない場合は、fifo と同様のファイル構造が作成されますが、EOVC は書き込まれません。こうすると、グループのヘッダー情報にランダム アクセスしなくてもグループの終わりを検出できるため、作成途中のファイルを解析するときなどに非常に便利です。ほかの標準的な IFF パーサーからエラーが報告されないように、GEND のサイズ フィールドはゼロに設定されます。EOVC のサイズ フィールドの値を標準的な IFF パーサーで読み取ると、fifo 拡張機能が実装されていない場合は、エラーが発生します。

警告: 旧バージョンの IFF パーサーとの互換性を保つために、GEND の後には sz_Unknown が付きます。また、fifo ファイルの場合は、EOVC の後にも sz_Unknown が付きます。

サイズが指定されていないブロックをスキップする確実な方法はないため、fifo ファイルの書き込み側プロセスと読み取り側プロセスの間でファイルの内容を十分に確認するようなロジックを組み込むことを強くお勧めします。

最新の IFF パーサーにはファイル内の EVOC を検出する機能が備わっています。ただし、ブロックのスキップ操作は大量の読み取り処理と比較処理を伴うため、性能が著しく低下することに注意してください。また、スキップ操作は 100% 信頼できるものではないことにも注意してください。

チャンネルmoveファイル（MOV）

4

チャンネル move ファイル（.mov）

move ファイル（.mov）は、モーション データに関連するチャンネル データ（x translate、y translate、z translate など）を保持する ASCII ファイルです。

フォーマット（Format）

.mov ファイルには行列形式の数値（チャンネル データ）が格納されます。行はフレームを表し、列はチャンネルを表します。.mov ファイルでは、個々のフレームごとに 1 つの行があり、個々のチャンネルごとに 1 つの列があります。

各行は改行文字の後（新しい行の先頭）から開始し、各列はスペースで区切る必要があります。各チャンネル データは倍精度浮動小数点数（double）で表されるため、有効桁数は約 16 桁となります。

次の図に、.mov ファイルのフォーマットの例を示します。このファイルは 4 行 6 列になっています。すなわち、4 フレーム × 6 チャンネルのチャンネル データがあります。

```
1.000 0.000 0.900 36.000 0.000 0.000 4.000 0.000 2.000 36.000 0.000
 8.000 0.000 9.500 16.000 8.000 0.000 0.000 9.450 0.000 0.000 50.000
 3.500 8.000
```

.mov ファイルの読み取りと書き込みを行って、モーション データをインポートしたりエクスポートしたりできます。ただし、.mov ファイルは単に数値の行列であり、ヘッダー情報は存在しないことに注意してください。同じオブジェクトとチャンネルの間でデータのエクスポートとインポートを行うためには、ファイル

のインポート/エクスポート オプション ウィンドウまたは movIn/movOut コマンドで同じチャンネル リストを使用する必要があります。

索引

記号

A

C

E

F

G

I

L

M

P

S

あ

い

う

か

く

こ

さ

し

せ

た

つ

て

と

の

ひ

ふ

へ

み

ら

り