音楽情報科学 25－2
（1998．5．27）

PMML Animator
---PMMLによるVRMLアニメーション---

会津大学コンピュータ理工学部
丸井淳史　西村憲

音楽を可視化したアニメーションの作成を支援するシステムPMML Animatorを開発した。PMML Animatorは音楽記述言語PMMLのマクロセットという形で実現されている。PMML　AnimatorはPMMLで書かれた音楽記述から自動的にVRMLファイルを生成する。MIDIイベントの持つ様々なパラメータに応じて、複数のオブジェクトが移動、回転、拡大縮小、材質変化するアニメーションを簡単に作成することができる。

PMML Animator
---VRML animation using PMML---

Department of Computer Software, The University of Aizu
MARUI Atsushi, Satoshi Nishimura

We have designed and implemented the PMML Animator, which helps to make an animation of music visualization. The PMML Animator is implemented as a macro set of the PMML. PMML Animator automatically generates a VRML file from a PMML musical description. The user can simply make an animation of objects with translation, scaling, rotation and changing material properties, in response to various parameters in MIDI events.

1. はじめに

音楽を可視化するということについて、いくつもの研究がなされている。音の高さや音量にあわせて3次元空間内で物体が移動するものなどはその代表例だろう[SG97]。可視化することにより音楽構造を容易に理解することが可能になり、また、耳の不自由な人にも音楽を（音とは違う形ではあるが）伝えることができる。

しかし、これまで音楽の可視化アニメーションの作成には高度なプログラミング能力が要求され、さらに、プログラムが完成

しても1通りの動きしか作れないことが多かった。また、楽曲ごとに異なる可視化手法によるアニメーションを作成することが必要な場合も多い。そこで、本研究では音楽の可視化アニメーションの作成を支援するツールを設計・開発した。

音楽の可視化アニメーションを作るには2通りの方法がある。リアルタイムに処理を行う方法と、非リアルタイムに行う方法である。

リアルタイムで処理をする場合は、音楽に画像が合うように、システムが受け取ったMIDIイベントの内容を即座に解釈し、それに応じた動画を生成する[GM96]。演奏しながら可視化処理が行えることが最大の利点である。しかし、これには高い計算量が必要になるため、複雑なものを作ろうとすると、わずかにではあるが画像の遅延が生じてしまう。

また、リアルタイム処理では、未来のイベントに依存したアニメーションが作れない。例えば、物体の位置を音量の大小によって変化させる可視化アニメーションにおいて、物体を滑らかに動かすために位置の補間を行いたい場合などがこれに相当する。リアルタイム処理では未来のイベントの内容を前もって知ることができないので、音量の変化を見越して物体をあらかじめ近い位置に移動しておくことができない。

本研究では非リアルタイムで行う方法を選択した。これにより、演奏しながらの処理が行えなくなるが、遅延のない可視化アニメーションを作ることができる。

本研究で作成したシステムはPMML (Practical Music Macro Language) をベースにしている。PMMLはMIDI楽器の自動演奏を目的とした音楽記述言語である。PMMLは音符、休符、和音、並行する複数のパートなどを扱える。また、C言語に似た制御構造、パラメータの自由曲線による連続変化、パート間の通信と同期などの非常に高度な機能を持っている[Nis97]。PMMLをベースにした理由は、算術演算やマクロ定義などの機能をアニメーション記述にも利用できるからである。

本研究のシステムはアニメーションとしてVRML (Virtual Reality Modeling Language) ファイルを出力する。VRMLは3次元空間内でのインタラクティブなシーンを記述し、インターネット上で仮想世界を構築する言語である。VRMLは商業レベルのグラフィックスを作ることができ、3次元空間記述言語の世界標準になりつつある。また、アニメーション機能にも優れ、JavaやJavaScriptを使うことによって、オブジェクト（物体）の動きなどをプログラムすることができる[CBM97]。VRMLファイルは容易に作成ができ、標準MIDIファイルの読み込みなどをサポートしているブラウザがあることなどから、本研究のためにVRMLが選ばれた。

VRMLのアニメーションを記述するためのスクリプト言語としてはVRMLScriptを選択した。VRMLScriptはJavaScriptをベースとした言語であり、VRMLイベントのコントロールのために様々な拡張がなされている[MK96]。

2. PMML Animator

2.1 概要

本研究ではPMMLの音楽記述から自動的にVRMLファイルを生成するマクロセット、PMML Animatorの設計および開発を行った。PMML Animatorを使用することにより、ユーザはPMMLによる音楽記述に可視化アニメーション作成のための指示や、アニメーションの制御のための命令を埋めこむことができる。PMML Animatorは柔軟かつ記述が容易になるように設計されており、音楽の可視化アニメーションを簡単かつ自由に作成できる。

PMML Animatorは楽曲ファイルに書かれ

たアニメーション生成用のコマンドと音楽記述をもとにVRMLファイルを生成する。出力されたVRMLファイルにはオブジェクトの形や色などや、音楽と画像の同期のための情報が含まれる。

出力ファイルが標準MIDIファイルとVRMLファイルなので、ファイルサイズがMPEGやQuickTimeなどの動画ファイルに比べて非常に小さく、インターネットなどを通じての配付が簡単かつ短時間で行える。

2.2 アニメーション作成の流れ

PMML Animatorによるアニメーション作成の流れは図1のようになっている。

まず、PMML Animatorを使用可能にするために、PMML楽曲ファイルの先頭部分にPMML Animatorをインクルードする。

PMMLインタプリタは楽曲部分を標準MIDIファイルとして出力し、またアニメーションはVRMLとして標準出力に出力される。ユーザはリダイレクトなどを使ってVRMLソースをファイルに保存する。

出力されたVRMLファイルには標準MIDIファイルのファイル名が記録されており、VRMLブラウザ (Silicon Graphics社のCosmoPlayerなど) で読み込めば、楽曲の演奏とともに可視化アニメーションが再生される。

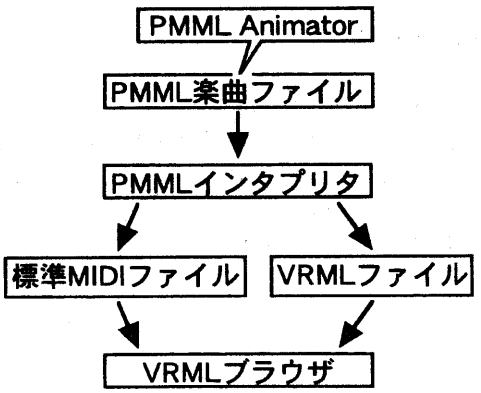

図1: アニメーション作成の流れ

2.3 使用法

PMML Animatorを使うためには、ユーザはPMML楽曲ファイルの先頭部分にいくつかの命令を書く必要がある。まずincludeを使用してPMML Animatorを読み込む。

PaObject命令はPMML Animatorの核をなす命令で、オブジェクトの形状と動作の方法を指定する。この命令はMIDIイベントの記述が始まる前に書いておかなければならない。

各PaObjectが1つのオブジェクトに対応しているため、いくつかのPaObjectを書けば複数のオブジェクトを同時に制御することができる。

以下がPaObjectの文法である。

```
PaObject("obj", "id", cmd[, cmd ...])
  obj     :=  VRMLオブジェクト名
  id      :=  識別子
  cmd     :=  varcmdまたはconcmd
  varcmd  :=  #(VRMLフィールド名,
                MIDIイベント種別,
                MIDIチャンネル, from, to)
  concmd  :=  #(VRMLフィールド名, 値)
```

PaObjectにはVRMLオブジェクト、識別子、そして少なくとも1つの、オブジェクトの属性を制御する命令 (*cmd*) が指定されていなければならない。*obj*には表1にあるVRMLオブジェクト名を指定できる。*id*はユーザが各オブジェクトに与える一意の名前である。

Box	直方体
Cone	円錐
Cylinder	円柱
Sphere	球

表1: 使用できるVRMLオブジェクト名

オブジェクトの属性を制御する命令はMIDIイベントの持つパラメータに依存したもの (*varcmd*) か、独立したもの (*concmd*)

を書くことができる。また、命令を複数書くことで、オブジェクトに「色を変化させながら左右に動かす」といった、複雑な動きを指定することができる。オブジェクト属性の種類としては表2のVRMLフィールド名を指定できる。

材質指定	ambientIntensity	環境光係数
	diffuseColor	拡散光係数
	emissiveColor	放射光成分
	shininess	鏡面光の指数
	specularColor	鏡面光係数
	transparency	透明度
座標変換指定	center	回転・スケーリングの中心
	rotation	回転移動量
	scale	スケール係数
	scaleOrientation	スケール方向
	translation	平行移動量
形状指定	size	x,y,z軸方向の長さ (Boxに使用可)
	bottomRadius	底面の半径 (Coneに使用可)
	height	高さ(Cone, Cylinderに使用可)
	radius	半径 (Sphere, Cylinderに使用可)

表2: 使用できるVRMLフィールド名

*MIDIイベント種別*はどの種のイベントのどのパラメータによってオブジェクト属性を制御するかを指定し、表3にあげたものを使用することができる。この部分にPMMLの仮想コントロールチェンジを使用すればMIDIイベントのパラメータには依存せずに、ユーザがその動きを全て明示的に指定するようなオブジェクトを定義することができる。

*MIDIチャンネル*は1から16であるが、0を指定することによって全てのMIDIチャンネルを対象とすることができる。

*from*と*to*はMIDIイベント中のパラメータ値からVRMLフィールド値への写像を、上限値と下限値によって指定する。これはVRMLフィールドの種類によって値の型を変えなければならない。VRMLフィールドがベクトル型を要求してきた場合には、*from*と*to*には配列を使用する。また、*from*と*to*に同じ値をセットした場合は、MIDIイベントの値がそのままVRMLに引き渡される。

n,note,note_on,note_off	ノート番号
ctrl[(*1~255の番号*)]	拡張コントロールチェンジ
bend	ピッチベンド量
cpr	チャンネルプレッシャー
prog	プログラム番号
tempo	テンポ
meta[(*1~127の番号*)]	テンポを除くメタイベントのうち、数値を指定するもの
t	時刻
tk	トラック番号
ch	MIDIチャンネル番号
v	ヴェロシティ
nv	ノートオフヴェロシティ
do	ノートオンとノートオフの時間差

表3: 使用できるMIDIイベント種別

基本的な例

音楽の可視化において最も一般的なものは、3次元空間内でオブジェクトを音高にあわせて動かすものである。これはPMML Animatorを使用することで以下のように容易に作成することができる。

```
 1: // 楽曲ファイルの先頭
 2: include("pa.pml")
 3: PaObject ("Sphere", "example",
 4:          #("translation", 'n', 1,
 5:            #(-1,0,0), #(1,0,0)),
 6:          #("diffuseColor", #(1,1,0)))
 7: PaBackground (#(#(.0, .0, .0),
 8:                  #(.1, .4, .1),
 9:                  #(.2, .8, .2)),
10:                #(#(.0, .0, .0),
11:                  #(.0, .1, .3),
12:                  #(.0, .2, .6)))
13: // 以下に音楽記述が続く
14: ch=1 C D E F G A B ^C
```

PaObjectの1つ目の引き数("Sphere")は動作させるオブジェクトの形状を表している。ここでは球を指定している。次の引き数("example")はこのオブジェクトを区別するための名前であり、自由に決めてしまってよい。

その次に出てくる2つの引き数はPMMLの配列になっている。最初の配列の、1つ目の要素("translation")はコントロールされるVRMLフィールドを、2つ目の要素('n')はVRMLフィールドをコントロールするMIDIイベントを表し、ここではノート番号を指定している。配列内の3つ目の要素(1)はチャンネル1のMIDIイベントを対象とすることを示している。その次に続く、2つの要素 (#(-1.0.0)と#(1,0,0))はオブジェクトが動く3次元座標の範囲である。この場合、音の高さにあわせて (-1,0,0) から(1,0,0)の範囲内で動かすことになる。

PaObjectの次の配列("diffuseColor")は球の色を指定している。この配列内の#(1,1,0)は直接VRMLの初期設定部分に書き込まれる。このように定数を指定する場合は要素数が2つになる。

次のPaBackgroundは背景色を指定する。1つ目の配列が地面の色に、2つ目の配列が空の色を表している。

この例に対しPMML Animatorを適用すると、「example」という名前の黄色い球が、x軸上を音の高さにあわせて左右に動くアニメーションができあがる。また、地面は緑色、空は紺色である。図2にSilicon Graphics社のCosmoPlayerとNetscape社のCommunicator上で実行した画面を紹介する。

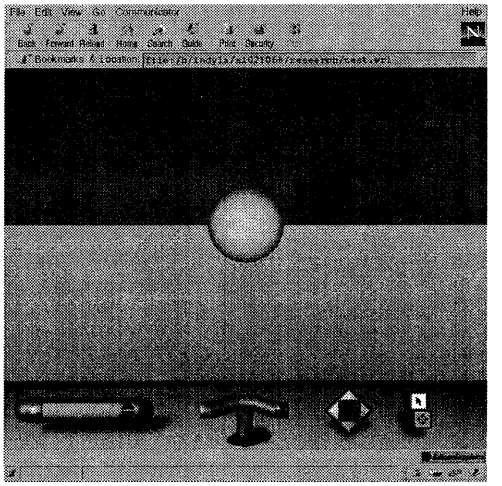

図2: 基本的な例の出力

イベントに依存しないオブジェクトの記述例

ときには演奏されるMIDIイベントに関係なく、自分で指定した動作をするオブジェクトを作りたいことがある。以下の例のように、新しく仮想コントローラを定義すれば、簡単に実現することができる。

```
 1: // ファイルの先頭
 2: include("pa.pml")
 3: defPaCtrl("Extra")
 4: PaObject("Cone", "object1",
 5:          #("translation",
 6:            'ctrl(PaExtra)', 1,
 7:            #(-1,0,0), #(1,0,0)))
 8: PaObject("Cylinder", "object2",
 9:          #("translation", 'n', 1,
10:            #(0,-1,0), #(0,1,0)))
11: // 音楽記述
12: ch=1 Extra(0)
13:       C D E
14:       ctrl_pt(PaExtra,127)
15:       F
16:       ctrl_to(PaExtra, 0, 15u, 1)
```

3行目で新しくExtraという名前のPMML Animator専用の仮想コントロールチェンジを定義している。

1つ目に定義されたオブジェクトは円錐で、Extraのコントロール値によって左右に動く。2つ目のオブジェクトはノート番号にあわせて上下に動く円柱である。

12行目から14行目にかけて、連続コントロールチェンジによってExtraの値を変化させている。円柱はC、D、E音にあわせて右に、F音にあわせて左に滑らかに移動する。PMML Animator専用の仮想コントロールチェンジのコントローラ番号を参照するときにはコントローラ名の先頭に「Pa」をつける。

自由曲線による位置の補間例

PMMLが折れ線、自由曲線による連続コントロールチェンジをサポートしているので、PMML Animatorからそれを位置の補間のために利用することもできる。これにより、未来のMIDIイベントに依存して動くアニメーションの作成が可能になる。

```
 1: include("pa.pml")
 2: defPaCtrl("ContNote_ctrl")
 3: defeff(ContNote_eff) {
 4:   init {
 5:     $flag = 0
 6:   }
 7:   case(note) {
 8:     if($flag == 0) {
 9:       ctrl(PaContNote_ctrl, n)
10:       $flag = 1
11:     } else {
12:       ctrl_pt(PaContNote_ctrl, n)
13:     }
14:   }
15:   wrap {
16:     ctrl_cto(PaContNote_ctrl,
17:              n, 15u, 0)
18:  }
19: }
20: PaObject("Sphere", "object1",
21:        #("translation",
22:          'ctrl(PaContNote_ctrl)',
23:          1, #(-3,0,0), #(3,0,0)))
24: ContNote_eff()
25: ch=1 C D C E C F C G C
```

2行目において、ノート番号を連続変化させるために使用する仮想コントローラContNote_ctrlを定義している。3行目から19行目にかけてエフェクタを定義している。このエフェクタの働きは、ノートイベントがあった場合は自由曲線の制御点を追加するというものである。16～17行目で折れ線(ctrl_to)か自由曲線(ctrl_cto)かの選択をする。20行目から23行目はContNote_ctrlの値をx座標としたオブジェクトを作成する。24行目でエフェクタをアタッチし、25行目は音楽記述となっている。

結果として、球が音の高さにあわせ左右に、自由曲線に沿って滑らかに動くことになる。このように、エフェクタを使用することによって、ノート番号のような離散的な情報を連続的な情報に変換することができる。

3. 内部の処理

PMML AnimatorはPMMLのマクロとエフェクタによって実現されている。PMML Animatorの開発に他のプログラミング言語を用いなかった理由は、少ない努力で、高い柔軟性を持ったシステムを開発するためである。PMMLは標準MIDIファイルの入出力、様々なデータ型の演算、文字列処理などの機能を備えている。それらを使用することでユーザは自分で容易に機能を追加することができる。

3.1 処理の流れ

PMML Animatorは「初期化」「イベント取り込み」「出力」という3つのステージによってアニメーションを作成する。

初期化ステージ　すべての変数、定数の初期化が行われる。

イベント取り込みステージ　このステージではすべてのMIDIイベントを読み、アニメーション作成に必要なものだけを選び

だしてバッファに一時保存しておく。MIDIイベントがいつ発生したかを知るために、曲の先頭からの経過秒数もイベントとともに保存される。

出力ステージ　イベント取り込みステージにおいて保存されたMIDIイベントをもとにVRMLを出力する。イベント発生時刻は複数のオブジェクトが同期するように変換され、アニメーションの制御情報はVRMLScriptノードとして出力される。オブジェクトの形状、色、材質なども適切なVRMLノードを使って出力される。

3.2 時間の管理

イベント取り込みステージにおいて、拍数とテンポ情報をもとに、曲の先頭からの経過秒数が計算され、MIDIイベントとともにバッファに保存される。そのMIDIイベントと経過秒数は、出力ステージでVRMLでの時間に変換される。なぜなら、VRMLのSoundノードは曲中での場所を0.0から1.0の間の実数として表すので、経過秒数を全体の演奏時間で割ってやらなければならないからである。

実行時の時間の管理はVRMLブラウザによって、TimeSensorノードとVRMLScriptを使って行われる。同期の正確さとパフォーマンスはVRMLブラウザの実行スピードに依存するが、最近のコンピュータの性能であれば問題ない。

3.3 オブジェクトの動きの制御

VRMLのTimeSensorとインターポレータ・ノードのみでは複雑な動きを表現するのに十分ではない。そこで、オブジェクトの動きを表現するためにPMML AnimatorではVRMLScriptを使用している。

現在のPMML AnimatorでのVRMLScriptの使用は、いたって単純である。VRMLScriptノードはTimeSensorから経過時間を受け取り、それをもとに各オブジェクトの属性を決定する。各オブジェクトの属性は数値としてオブジェクトに渡され、それによって画像が更新される。

PMML AnimatorはPaObject命令において数値の線形変換を指定できるが、現在は非線形変換には対応していない。しかし、PMMLが自由曲線による連続コントロールチェンジをサポートしており、PMML Animatorからそれを利用することもできるので、さほど大きな問題ではないように思われる。

3.4 複数のオブジェクト属性の制御

PMML Animatorは、1つのオブジェクトを数種類のMIDIイベントのパラメータによって同時に制御する場合、パラメータごとにVRMLScriptノードを出力する。例えば、ノート番号をオブジェクトの位置に、音量をその色にといった場合である。

実現法は以下のとおりである。まず、イベント取り込みステージにおいて、アニメーションに関係のあるMIDIイベントをすべてをバッファに保存する。次に、出力ステージにおいてMIDIイベントは時間順に並べかえられ、VRMLScriptの形に変換される。その際、異なる属性は別々のVRMLScriptノードに振り分けられる。各々のVRMLScriptノードは独立して機能し、単一のTimeSensorノードによって同期がとられることになる。

4. まとめ

PMML Animatorは、VRMLをPMMLの音楽記述から制御するという目標を十分に達成できた。当初計画された機能は、複数のオブジェクトの簡便な制御、複数のMIDIイベントによる1つのオブジェクトの制御、という2点だった。すべての機能の設計および開発は成功し、また正常に動作する。

PMML Animatorを使うことによって、ユ

ーザは非常に短時間で音楽の可視化アニメーションを作ることが可能になった。

今後の課題としては、オブジェクトの動きに物理法則を導入することである。オブジェクトどうしの衝突や重力の適用などをシミュレートしたアニメーションが作成できるようなシステムに発展させたいと考えている。

参考文献

[CBM97] Rikk Carey, Gavin Bell, and Chris Marrin. *ISO/IEC 14772-1:1997 Virtual Reality Modeling Language (VRML97).* The VRML Consortium Inc., 1997. http://www.vrml.org/Specifications/VRML97.

[GM96] Masataka Goto and Yoichi Muraoka. A Virtual Dancer "Cindy" ---Interactive Performance of a Music-controlled CG Dancer---. In Proceedings of *Lifelike Computer Characters '96*, page 65, October 1996.

[MK96] Chris Marrin and Jim Kent. VRMLScript Reference. Silicon Graphics, Inc., October 1996.

[Nis97] 西村憲. 音楽記述言語PMMLの概要. 情報処理学会研究報告 97-MUS-12, pp. 59-66, 1997

[SG97] Sean M. Smith and Glen N. Williams. A Visualization of Music. Visual proceedings on *SIGGRAPH '97 visual proceedings: the art and interdisciplinary programs,* page 196, also in Proceedings of *Visualization '97*, 1997.