グラフィクスとＣＡＤ 37−10
（1989. 2. 23）

対話的スクリプト法によるアニメーションの生成

INTERACTIVE SCRIPTING FOR COMPUTER ANIMATION

栗原 恒弥
Tsuneya KURIHARA
（株）日立製作所
Hitachi Ltd.

渡辺 智
Satoshi WATANABE
（株）日立超ＬＳＩエンジニアリング
Hitachi VLSI Engineering Co.

あらまし　コンピュータアニメーションを効率よく生成することを目的に、対話的スクリプト生成法を提案する。この方法は従来の２つのアニメーション生成方法、すなわち、対話的手法とスクリプト法（言語による記述方法）とを統合しようとするものである。提案方法では、形状及び動きはスクリプト（専用言語）で記述される。スクリプトはテキスト・エディタで編集できるだけでなく、画面上に表示されている物体をマウス等で直接操作することにより対話的に編集することが可能である。またスクリプトのパラメタは入力装置を用いて対話的に修正できる。言語型のシステムに対話性を導入することにより、容易にアニメーションが生成できる。

Abstract　Interactive scripting method is presented for computer animation. This method attempt to span the two common approachs to animation: programmed and interactive. Models and motions are described with script (animation language). Script can be edited not only by a text editor, but also by direct manipulation of graphical objects. Parameters of script can be controlled interactively with logical input devices. Combining language based method and interactive method, computer animation can be produced efficiently.

１．　はじめに

ＣＧ（コンピュータ・グラフィクス）技術の進歩はめざましく、非常にリアルな映像の生成が可能となっている。現在、ＣＧはさまざまな分野で注目され、利用されている。しかし形状や動きのモデリング技術は遅れており、特にアニメーションの生成には膨大なデータ入力が必要となっている。効率よくアニメーションを生成する方法が望まれている。

従来のアニメーション生成方法として、言語ベースのスクリプト法及び対話的な手法が提案されている［１］。

スクリプト法［２，３］では、形状や動きはアニメーション用の専用言語、すなわちスクリプトで記述される。この方法は拡張性に富み、動きの再利用やデータベースの構築が可能である。また、環境に応じてオブジェクトが自発的に反応するような記述も容易である［４，５］。

しかし、スクリプト法ではモデリングの過程を見ることができないためマン・マシン・インターフェースが悪いという問題点を有している。

これに対して、キーフレーム法等の対話的方法［１］では、ユーザは特定の時刻（キーフ

レーム）での物体の位置を対話的に指定する。キーフレームでのデータを補間してアニメーションを生成する。操作は直観的かつ対話的で、操作性がよい。このため現在のアニメーション・システムの多くがこの方法を採用している。

この方法ではユーザが全ての自由度について指定する必要がある。このため、複雑な物体の運動を記述するためには非常に多くのデータを必要とする。また、拡張性に欠け、動きの再利用、データベース化が困難である。

本報告では、これらの問題点を解決するために、２つの方法を統合した対話的スクリプト生成法を提案する。従来、このような方法として、スクリプトのパラメタを対話的に修正する方法が提案されている［６］。本報告ではパラメラだけでなく、スクリプト自体を対話的に編集する方法を提案する。

アニメーションの記述はスクリプトにより行う。スクリプトはテキスト・エディタで編集できるだけでなく、画面に表示されている物体をマウス等で直接操作するによって自動的に生成及び編集が可能である。生成されたスクリプトからアニメーションが即座に表示される。アニメーションの実行中にはスクリプト中のパラメタを対話的に変更することが可能である。変更されたパラメタは自動的にスクリプトに反映される。対話的手法とスクリプト法を統合することにより容易にアニメーションの生成が可能である。

２．　対話的スクリプト生成法の概要

対話的スクリプト生成法は、言語ベースのスクリプト法に対話性を導入するものである。その構成を図２．１に、特徴を以下に示す。

①スクリプトを編集すると即座にアニメーションが表示される。

②表示画像上の物体をマウス等で直接操作することが可能である。
直接操作によって、スクリプトが自動的に生成、編集される。

①は、従来のスクリプト法に表示機能を付加したものである。スクリプト・エディタにはアニメーションのスクリプトが表示されている。このスクリプトは通常のテキスト・エディタと同様に、編集できる。スクリプト・エディタでスクリプトを編集すると、それに対応したアニメーションが生成され、グラフィクス・ウィンドウに表示される。このアニメーションを参照しながらスクリプトの編集を行う。

②は、①の逆を行うものである。グラフィクス・ウィンドウには特定の時刻における映像あるいはアニメーションが表示される。グラフィクス・ウィンドウに表示されている物体はメニュー等で操作することが可能である。このとき対応するスクリプトが自動的に生成される。このスクリプトは直ちにスクリプト・エディタに表示される。

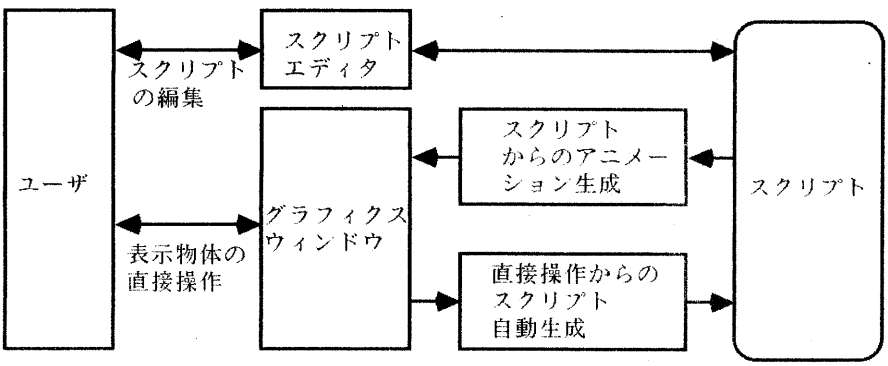


図2.1 対話的スクリプト生成法における対話環境

このように対話的スクリプト生成法では、スクリプトのテキスト編集と、画面上の物体の操作に対応関係がある。このため、スクリプトとして記述しやすい物体間の依存関係等はスクリプト・ウィンドウで記述し、対話的な指定が適している図形の位置や大きさ等はグラフィクス・ウィンドウで指定することができる。また、グラフィクス・ウィンドウでの操作だけで簡単なアニメーションが生成できる。

3. スクリプト（モデリング言語）

対話的スクリプト生成法では3次元アニメーションはすべてスクリプト（専用のモデリング言語）で記述する。すなわち、3次元物体の形状、変形、移動、レンダリング方法、照明やカメラ等をスクリプトで記述する。

以下、プロトタイプにおけるスクリプトの詳細を説明する。

3．1 スクリプトの概要

プロトタイプ・システムでのスクリプトの特徴を以下に示す。

（1）近代的なプログラミング構造

様々な条件分岐、繰返構造、手続き、関数構造、抽象データ構造が扱える。このため、3次元物体及びその運動を容易にモデル化できる。

（2）オブジェクト指向

CGで扱うデータは、光源、カメラ、3次元物体等、全てオブジェクトと考えるのが自然である。オブジェクト指向の考え方を取り入れることで、オブジェクト間の階層構造、属性の継承が容易に記述できる。また、動物や人間等、環境や他の物体との関係から自発的に運動するもののモデル化にも適している。

（3）対話的な環境

対話的な言語を用いることで、アニメーションを対話的に修正できる。すなわち、スクリプトを変更すると、即座に変更されたアニメーションが表示される。

プロトタイプのスクリプトはLisp上のオブジェクト指向言語Flavor上に実現した。基本的な言語の構造はLisp及びFlavorに準じている。スクリプトはFlavorにアニメーション用のクラスを追加したとも考えることができる。

3．2 オブジェクト

CGで扱うカメラ、照明、物体等はすべてオブジェクトとして定義される。オブジェクトは指示（メッセージ）を受け取ると、その指示に対応した動作（メソッド）を行う。

オブジェクトには一定期間だけ動作を継続させるようなメッセージを送ることができる。このようなメッセージを受けたオブジェクトは実行中の動作を記憶しておき、次の単位時刻になった時点でその動作に応じた振舞いをする。

同じ種類のオブジェクトの全体はクラスと呼ばれる。物体の形状（多角形、自由曲面等）を定義するために基本的なクラスが用意されている。これらのクラスに属するオブジェクトには以下のメッセージを送ることが可能である。

1）幾何学的変換（移動、回転等）
2）表示属性の指定
3）物体のリンク関係の記述

クラスの拡張は容易である。基本的なクラスを組み合わせて複雑なクラスを構成することが可能である。

3．3 アニメーション・スクリプト

アニメーションを生成するメインプログラムをアニメーション・スクリプトと呼ぶ。このスクリプトには、各時刻での物体の生成、物体へのメッセージの伝達等を記述する。一例を下に示す。

```
(defun simple-scene ()
  (animate (time 120)
     (at 10
          (new-obj  a-man 'human-class)
          (send a-man :position
```

```
            (vec 100 0 0))
      (at 30
          (send a-man :start-walk)))
```

この例は１２０フレームのアニメーションであり、時刻１０で物体a-man を生成し、その位置を（100,0,0)とする。さらに時刻３０で物体a-man にメッセージ:start-walk を送り、歩行を始めさせるものである。

４．　対話的なアニメーション生成

４．１　対話方法

図４．１にプロトタイプのウィンドウの表示例を示す。グラフィクス・ウィンドウ、スクリプト・エディタの他、パラメタを設定するためにスライダが存在する。

スクリプト・エディタからの物体を操作する場合には、以下のよう入力する。

（ｓｅｎｄ　オブジェクト名　：メッセージ名　パラメタ列）

オブジェクトにメッセージが送られ、グラフィクス・ウィンドウの映像が更新される。

グラフィクス・ウィンドウでの物体の操作は、以下のように行う。

①操作したい物体をマウス等で選択する。
②その物体に対して送ることが可能なメッセージがメニューに表示されるので、そのなから選択する。
③必要ならばメッセージのパラメタを入力する。パラメタはスライダ等を用いて対話的に入力することもできる。

これらの操作は対応するスクリプトに変換され、スクリプト・ウィンドウに表示されてから実行される。キーボードからの入力とマウス等を用いた対話的な入力とは区別されない。なお、パラメタへは定数だけでなく、変数や任意の式をキーボードから入力したり、スクリプト・エディタからコピーすることができる。このためパラメトリックなアニメーションの記述が容易となる。

この他に、グラフィクス・ウィンドウのメニューを用いてオブジェクトの生成、変数の生成、変数への代入、条件分岐等の指示が可能である。

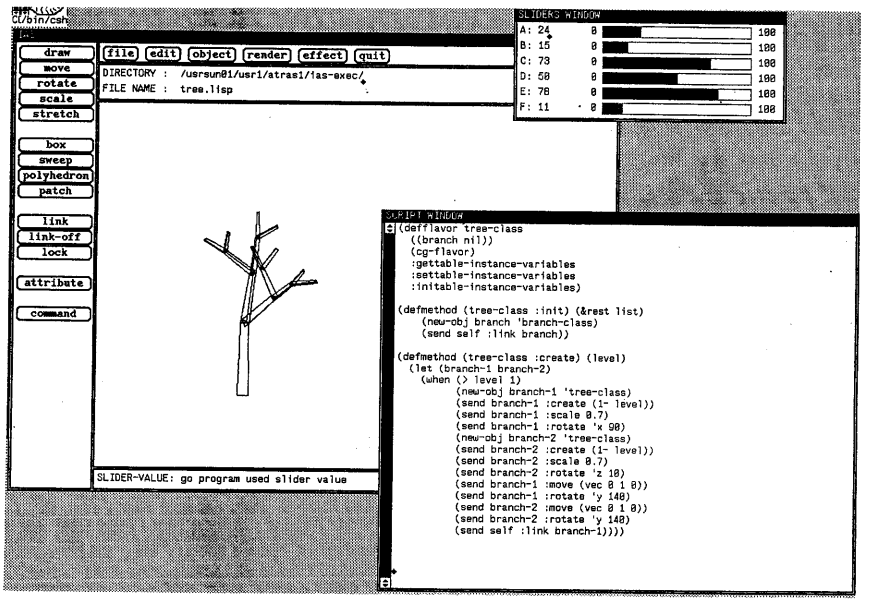


図４．１　プロトタイプシステムの表示例

４．２　アニメーションの生成手順

対話的スクリプト生成法でのアニメーションの生成手順は以下のようになる。

①登場するオブジェクトのクラスを定義する。
②使用するメソッドを定義する。
③アニメーション・スクリプトを定義する。
④アニメーション・スクリプトのパラメタを対話的に修正する。

クラス及びメソッドが既に定義されている場合には、③④でアニメーションが生成できる。

４．３　アニメーション・スクリプトの編集

アニメーション・スクリプトに各時刻でのオブジェクトの生成、オブジェクトへのメッセージの伝達を記述して、アニメーションを生成する。これは以下のように行う。

（１）オブジェクトの生成

オブジェクトを生成したい時刻をメニュー等で対話的に設定する。次にクラス名を指定して、オブジェクトを生成する。これは、スクリプト・エディタからもグラフィクス・ウィンドウからも可能である。スクリプト・エディタからは

（ｎｅｗ－ｏｂｊ　オブジェクト名　クラス名）

とする。

メニューを用いる場合には、メニューから「オブジェクト生成」を選択する。現在定義されているクラス名が表示されるのでその中から選択する。生成されたオブジェクトはグラフィック・ウィンドウに表示される。

（２）オブジェクトへの指示

オブジェクトにメッセージを送りたい時刻をメニュー等で対話的に設定した後、対話的にオブジェクトにメッセージを送る。

４．４　パラメタの対話的な修正

メニューから「実行」コマンドを入力すれば、アニメーション・スクリプトが定義されたアニメーションが再生される。このとき、対話的にパラメタを設定することが可能である。すなわち、スクリプト・ウィンドウから設定したいパラメタを選択し、これをスライダ等で対話的に修正することが可能である。プレイバックを見ながらパラメタを最適なものに設定できる。このため、物体の位置や、方向等を対話的に修正することが可能である。

４．５　メソッドの編集

オブジェクトがメッセージを受けとった場合に起動されるメソッドを定義することが可能である。その手順を以下に示す。

①メソッドを指定したいオブジェクトをマウス等で選択する。
②メニューから「メソッド編集」を選択する。
③メソッドの名称及びパラメタを入力する。パラメタにはその実例も入力する。もしそのメソッドが既に定義されている場合には定義されているプログラムがスクリプト・エディタに表示される。
④画面上のオブジェクトに種々のメッセージを送り、所望の動きを得る。このときこれはすべてプログラムとしてスクリプト・エディタに表示される。

④において、メニューからも簡単な条件分岐、繰返を定義できる。これはメニューから「if」、「repeat」を選択することで実現される。なお、スクリプトウィンドウで直接プログラムを修正することも可能である。

４．６　クラスの定義

新しくクラスを定義することが可能である。クラスの定義は以下のように行う。

①メニューから「クラス定義」を選択し、クラス名、インスタンス変数名を入力する。
②そのクラスのオブジェクトを構成するオブジェクトを対話的に生成する。

②で生成されたオブジェクトはすべてそのクラ

スのインスタンス変数として登録される。また、②でのオブジェクトの生成手順はクラスの生成メソッドとして登録される。

5. アニメーションの生成例

現在、対話的スクリプト法を用いたアニメーション・システムのプロトタイプはＳＵＮ３上にCommon Lisp で記述されるいる。ウィンドウ環境はSun Viewを用いている。以下、簡単なアニメーションの生成例について述べる。

5．1 木のモデルの生成

図５．１に再帰的な木のモデルを対話的に生成する例を示す。これ例では次のよう木のモデルを定義している。

①一本の枝からなるクラス名「tree」を定義する。
②実際に木を生成するメソッド「:create 」を定義する。これは、新たに木を２本生成し、これを縮小して、自分自身につなげることにより実現する。

メソッド「:create 」はパラメタとして分岐の回数level を取る。このlevel が０の場合には、新たな木の生成は行わない。このような簡単な条件分岐はメニューを用いて指定できる。これは、メニューから「when」を選択し、次に真か偽を返す関数を呼び出すことで実現する。

木の生成、回転、移動等はオブジェクトを見ながら対話的に指定できる。また、枝の角度等も対話的に修正できる。

5．2 ロボットのモデルの生成

図５．２にロボットモデルの生成例を示す。ここでは、ロボットはリンクされた直方体で定義されている。クラス「human-class 」の定義において、必要な部品を生成し、対話的に移動し、リンクする。以上でロボットのモデルが生成される。ロボットのモデルに対しては間接角の指定が可能である。

5．3 ロボットの歩行

ロボットの歩行の例を図５．３に示す。この例では１歩だけの歩行を定義している。動かしたい部分を選択してその部分に動作の指示を行う。例えば、右足を選択しその角度を１５こま後に１５度にし、さらに、１５こま後に－１５度にする等の指定を行う。このような指定を各部位に行うことで歩行を行うメソッドを定義できる。

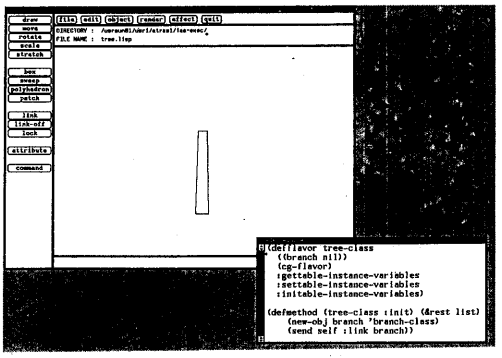
実際に歩行させるスクリプトを図５．４に示す。この例は３つのロボットが生成され、歩き続けるものである。

6. おわりに

アニメーションの生成方法として、スクリプトによる記述に対話性を導入した対話的スクリプト生成法を提案した。画面上の物体の直接操作からスクリプトを自動的に生成することで効率よくアニメーションが生成できる。

今後、運動の知識や法則の取り込み、拘束条件の利用等を検討していきたい。

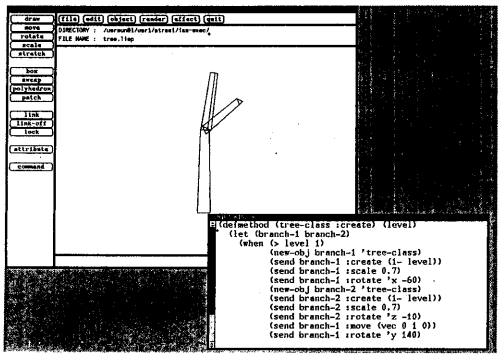
１）Magnenat-Thalmann, et al.: Computer Animation, Springer-Verlag (1985).
２）Reynolds,C.W.:Computer Animation with Scripts and Actors, Computer Graphics, Vol.16,No.3,pp.289-296(1982).
３）村上、他：３次元アニメーション用モデリング言語、日経ＣＧ, no.1,pp.146-158 (1986).
４）内木、他：行動シミュレーションに基づいたアニメーションシステムＰaradise 、コンピュータソフトウェア、vol.4,no.2, pp.24-38(1987).
５）Reynolds,C.W.:flocks,herds,and schools: a distributed behavioral model, Computer Graphics,Vol.21, No.4,pp.25-34 (1987).
６）Hanrahan,P.,Sturman,D. : Interactive Animation of Parametric Models, Visual Computer,Vol.1,No.4,pp.260-266(1985).



```
(defflavor tree-class
   ((branch nil))
   (cg-flavor)
   :gettable-instance-variables
   :settable-instance-variables
   :initable-instance-variables)

(defmethod (tree-class :init) (&rest list)
   (new-obj branch 'branch-class)
   (send self :link branch))
```

(1) クラス「ｔｒｅｅ」の定義

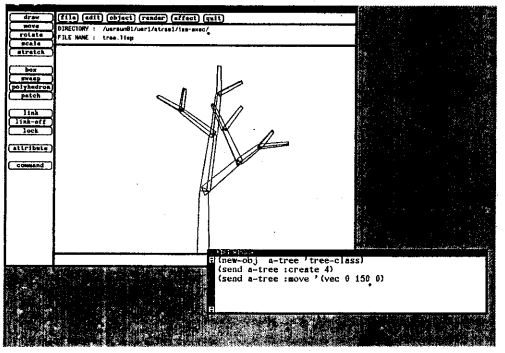
ｔｒｅｅの部品となる枝（ｂｒａｎｃｈ）を定義する。



```
(defmethod (tree-class :create) (level)
  (let (branch-1 branch-2)
    (when (> level 1)
          (new-obj branch-1 'tree-class)
          (send branch-1 :create (1- level))
          (send branch-1 :scale 0.7)
          (send branch-1 :rotate 'x -60)
          (new-obj branch-2 'tree-class)
          (send branch-2 :create (1- level))
          (send branch-2 :scale 0.7)
          (send branch-2 :rotate 'z -10)
          (send branch-1 :move (vec 0 1 0))
          (send branch-1 :rotate 'y 140)
```

(2) メソッド「：ｃｒｅａｔｅ」の定義

新たに木を２本生成し、これを対話的に縮小、回転、移動して、元の木に接続する。

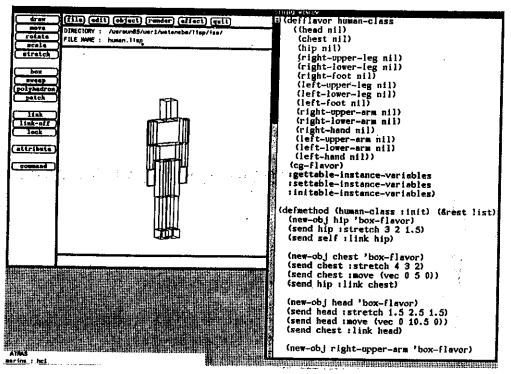


```
(new-obj  a-tree 'tree-class)
(send a-tree :create 4)
(send a-tree :move '(vec 0 150 0)
```

(3) 木の生成

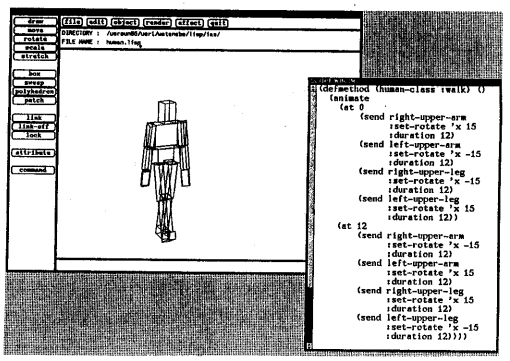
分岐数４の木を生成し、移動する。

図５．１　木の生成例



```
(defflavor human-class
  ((head nil)
   (chest nil)
   (hip nil)
   (right-upper-leg nil)
   (right-lower-leg nil)
   (right-foot nil)
   (left-upper-leg nil)
   (left-lower-leg nil)
   (left-foot nil)
   (right-upper-arm nil)
   (right-lower-arm nil)
   (right-hand nil)
   (left-upper-arm nil)
   (left-lower-arm nil)
   (left-hand nil))
  (cg-flavor)
  :gettable-instance-variables
  :settable-instance-variables
  :initable-instance-variables)

(defmethod (human-class :init) (&rest list)
  (new-obj hip 'box-flavor)
  (send hip :stretch 3 2 1.5)
  (send self :link hip)

  (new-obj chest 'box-flavor)
  (send chest :stretch 4 3 2)
  (send chest :move (vec 0 5 0))
  (send hip :link chest)

  (new-obj head 'box-flavor)
  (send head :stretch 1.5 2.5 1.5)
  (send head :move (vec 0 10.5 0))
  (send chest :link head)

  (new-obj right-upper-arm 'box-flavor)
```

図5．2　クラス「human-class」の定義



```
(defmethod (human-class :walk) ()
  (animate
    (at 0
        (send right-upper-arm
              :set-rotate 'x 15
              :duration 12)
        (send left-upper-arm
              :set-rotate 'x -15
              :duration 12)
        (send right-upper-leg
              :set-rotate 'x -15
              :duration 12)
        (send left-upper-leg
              :set-rotate 'x 15
              :duration 12))
    (at 12
        (send right-upper-arm
              :set-rotate 'x -15
              :duration 12)
        (send left-upper-arm
              :set-rotate 'x 15
              :duration 12)
        (send right-upper-leg
              :set-rotate 'x 15
              :duration 12)
        (send left-upper-leg
              :set-rotate 'x -15
              :duration 12))))
```

図5．3　メソッド「:walk」の定義

```
(defun walk-test nil
  (animate
    (at 0
        (new-obj man-1 'human-class)
        (send man-1 :motion-forever :walk))
    (at 10
        (new-obj man-2 'human-class)
        (send man-2 :move (vec -10 0 0))
        (send man-2 :motion-forever :walk))
    (at 20
        (new-obj man-3 'human-class)
        (send man-3 :move (vec 10 0 0))
        (send man-3 :motion-forever :walk))))
```

図5．4　歩行のアニメーションのスクリプト