《論　文》

汎用計算機動画作成システムと内部構成*

西原清一** 石垣昭一郎*** 萩原　宏***

Abstract

CINEMA is a general-purpose film animation system, which runs either in the interactive mode or in the program processing mode. The hardware configuration consists of FACOM 270-30, a graphic display FACOM 6231A and the cinecamera driver unit.

A motion picture in the system is composed of seven classes of constituents which are handled by the data handling subroutine package (DSP): cartoons, scenes, objects, motions, paths, rates and dot sets.

The movie maker can input these constituents using a lightpen, function keys and a teletype, and examine the scenes without producing films. And further, the system can process programs written in high level language CINEMA, having the same name as the system.

Using the cinecamera driver unit, the film advance control can be performed by programs.

1. まえがき

動画は視覚にうったえるという大きな特長を持っているが，従来は人手によって作成されてきたため，時間的・経済的・技術的な面で問題点が多かった．動画作成を計算機を用いて行う技術の開発は，実用的な見地から今後最も期待される応用分野の1つであろう．計算機動画の本格的な試みは，Knowlton[1] の BEFLIX システムなどに始まる．その後，動画というものを様々な立場から分析し，いくつかのシステムが作られた[2),3)]．また国内でも，急速に計算機動画への関心が高まりつつある****．従来，個々の問題向きの比較的小規模のシステムが多かったが，Talbot 他[4]は2次元線図形の動画をディスプレイ面での対話操作により定義する汎用システム Animator を作成した．

これらの大部分はインタラクティブ・システムであり，そのため複雑な図形や動きを綿密な計画のもとに組織的に構成するということが困難であった．また動作・変形あるいは運動の定義，すなわち動画の大きな要素である時間的な側面が軽視されがちであった．一方，汎用システムの場合，図形やその動作などを効率よく格納・検索できるデータ構造が要求され，また図形の変換・表示，対話的操作や言語の処理，撮影制御などを含むため，必然的に大きなプログラムとなる．したがってシステム構成という面から，充分な検討・整理が行われている必要があろう．

筆者らは先に実験的なシステム[8]を作成したが，このたび，(i) 汎用性，(ii) 対話操作および言語記述いずれでも定義可能，(iii) 効果的なデータ構造，(iv) 撮影制御などに注目して新しいシステムを作成した．本文では，動画の構造分析を行ったあと，上記の点について特徴を述べる．また1コマ生成の速さ，Animator との比較を簡単な例により行った．

2. 動　画

動画はその構造において多くの興味ある問題を含んでいる．ここでは，本システムで採用した3つの主要な考え方について述べる．

2.1 場面の階層化

動画の構成要素は，表示図形とその変形動作とに大別される．この関係を Fig. 1 のように表わそう．すなわち，表示図形 (O, object) は“点”で，また対応する変形動作 (m, motion) はそれを囲む“閉曲線”で表わす．Fig. 1 のような O と m の最も基本的な

---

* A general purpose computer animation system and its internal structure, by Seiichi NISHIHARA (Data Processing Center, Kyoto Univ.), Shoichiro ISHIGAKI and Hiroshi HAGIWARA (Faculty of Engineering, Kyoto Univ.)

** 京都大学大型計算機センター

*** 京都大学工学部

**** グラフィック・ディスプレイ報告集，情報処理学会，1970．電子計算機による動画作成，情処月例会資料 77，1973．

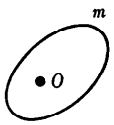

Fig. 1 A scene.

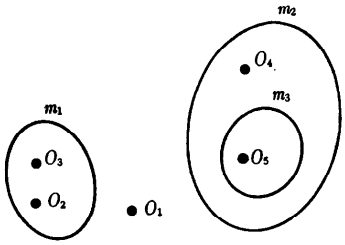

Fig. 2 A compound scene.

組みは，1つの場面 (scene) を形成する．すなわち，場面とは表示図形全体がある変形動作を一斉に行う場合のみ，表現可能である．これに対応するため従来は，動画を場面の集合として定義できるようにしているものが多い．

しかし，より一般的な場合として，Fig. 2 に示すような階層的な定義をも許す方が人間にとって便利であろう．すなわち，変形動作を場面そのものにも適用できるように拡張するのである（場面の階層化）．Fig. 2 の例では，表示図形は O_1〜O_5 の5部分から成り，O_1 は不変，O_2 と O_3 は m_1，O_4 は m_2，O_5 は m_2 と m_3 は変形動作を行うことを表わす．Fig. 2 のような拡張された場面を含む動画に対して，

$$(\phi, O_1)+(m_1, O_2+O_3)+(m_2, O_4+(m_3, O_5)) \tag{2.1}$$

という表現式を与え，これを“動画の式”と呼ぼう．動画の式の与え方は説明なしにほぼ明らかであろう．式中，ϕ はそれを含む閉曲線が存在しないような図形に対して付けられる空の変形動作 (i.e. 不変) を表わす．Fig. 2 において，O_1 はたとえば画面の背景（書割り）のような，終始，静止している図形である．またとくに O_5 は，変形動作 m_2 を行う図形 O_4 に随伴して動作すると同時に，さらに重畳的の変形動作 m_3 を行うことを表わす．たとえば，平行移動 (m_2) している図形 (O_4 および O_5) の一部 (O_5) が回転 (m_3) しているというような場合を表わす．

場面の階層化を許すことは，表示図形の中に主従あるいは拘束の関係が導入されることを意味する．あるいは，変形動作を平行移動と回転のみに限れば，場面の階層化は，各図形に随伴する運動座標系の導入を意味する．その場合，各座標系はより外側の変形動作に従って運動する．すなわち，変形動作を表わす閉曲線は，すぐ外側の点（図形）に対する内側の点の相対運動を表わしている．

ここで，2つの変形動作 m_1, m_2 が重畳的に同時に生起する場合を考えよう．その結果も1つの変形動作作となるであろう．これを $m_1 * m_2$ で表わそう．後述するように，一般に＊の操作では交換律は成り立たない．この操作＊を用いると，階層を含んだ動画の式を書き換えて，Fig. 1 のような階層のない場面のみから成る動画の式を与えることができる．たとえば，式 (2.1) に対して，

$$(\phi, O_1)+(m_1, O_2+O_3)+(m_2, O_4)+(m_2 * m_3, O_5) \tag{2.2}$$

が得られる．どちらの場合も全く同一の動画を表わしている．これは，従来の動画の定義方法でも，我々の階層的な動画の定義方法と同じ表現能力を有することを示している．しかし，実際の使用者にとっては，ある場面を他の場面の一部として用いたり，同じ変形動作を行う複数の図形をまとめて扱ったり，あるいは変形動作の重ね合わせなども自由に扱えることが望ましい．このため本システムでは，付録の書換え規則 29. および 37. に示すように，画面の階層化を認めている．

2.2 表示図形

従来は，コマンドや入力機器 (lightpen, joystick など) を用いて線分や円弧を対話的に生成するシステムが多い，しかし，精度や操作性に問題が多く，対話的に複雑な図形を定義するのは，とくに3次元の場合は困難である．本システムでは，図形の最も基本的な構成単位として，点の集合 (dot set) を用いた．図形は，点の集合に対してそれらを線分で結んだり，各点位置に既成の図形を描いたりするための種々の操作（コマンド）を施すことにより形成される（付録の規則 13.〜18. 参照）．点の集合を採用したのは，つぎのような理由による．(i)コマンドを変えることによって1つの点集合から種々の図形を生成できる．(ii)変形動作の1つ，平行移動の軌道 (path) としても使用する．(iii)図形の形状そのものが否剛体的(変形体的)に変わるような変形動作（次節）を定義する場合，点の集合を構成要素とする必要がある．(iv)各点位置に既成の図形を描かせることで，図形の生成効率を上げる．

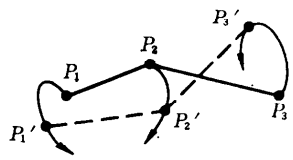

Fig. 3 Shape transformation.

2.3 変形動作

動画の最大の特徴は変形動作にある*. 変形動作は，その種類，総変化量，時間的変化率 (rate) の3つで決まる．とくに rate は，総変化量に対する相対変化量 r の，時間 t に関する値である．すなわち，一価関数 f に対して $r=f(t)$ と表わされる．単一目的の視覚的シミュレーションを行う動画システムでは，各コマ生成時のサンプリング時刻に従って rate は自然に含まれることになる．しかし，汎用システムでは，変形動作とくに rate の要素を取り入れた例は，Baecker[3] にみられるが，大部分のシステムでは $r=k\cdot t$ (k は定数) としているため，動きが単純あるいは不自然である．

変形動作の種類としては，平行移動，回転，拡大・縮小およびこれらの合成が考えられる．これらは，表示図形が常に相似形を保っているという共通性があり，'相似変形動作' と呼ぼう．さらに，図形の形状自体が変化する場合も一般の動画では必要である．これを '形状変化動作' と呼ぶ．Fig. 3 はその例である．最初，3点 P_1, P_2, P_3 を結んで得られる図形（実線）が，各構成点が実線矢印のような径路を動くとすると，ある時間ののち，3点はそれぞれ P_1', P_2', P_3' の位置に移動するであろう．このとき，原図形は，破線のように変化している．このように図形の構成要素として点の集合を採用することによって，任意の形状変形動作の定義が可能となる．

ここで，2.1 で触れた変形動作の重ね合せ操作＊を詳述しよう．変形動作 m は，原図形に対して上述の種類の変換を施すのであるが，rate で述べたように，それ自身時間によって変化してゆく．そこで変形動作を $m(t)$ と表わす．相似変形動作のみの場合，$m(t)$ は変換マトリクスで与えられる[8]．$(m(t), O)$ は，図形 O の時間 t の後における状態を表わす．とくに任意の m に対して $m(0)\equiv\phi$，したがって，$(m(0), O)$ は図形 O と同じものである。$(m_1(t)*m_2(t), O)$ は，図形 O の各構成点に対して，まず $m_2(t)$ を，続いて $m_1(t)$ の順に変換を施すことを意味する．交換律は成り立たない．また，$m(t)$ 自身は，ある時間間隔 $0\leq t\leq T$ の範囲において変化するが，実際に1つの階層的場面を生成する場合に，ある図形 O に，$m_1(t)$ と $m_2(t)$ が並行的あるいは継起的に施されるような指定が可能となっている．すなわち，たとえば Fig. 4 の場合，$(m_1(t)*m_2(t), O)$ は，実際にはつぎのような変形動作が行われる．

$$\begin{cases} (\phi, O) & 0\leq t\leq t_1 \\ (m_1(t-t_1), O) & t_1\leq t<t_2 \\ (m_1(t-t_1)*m_2(t-t_2), O) & t_2\leq t<t_3 \\ (m_1(T_1)*m_2(t-t_2), O) & t_3\leq t<t_4 \\ (m_1(T_1)*m_2(T_2), O) & t\geq t_4 \end{cases}$$

ある変形動作の完了後は，その最終状態が維持される．なお，動画は，このような場面とその開始時刻とのペアの集合で定義される (Fig. 11 参照)．

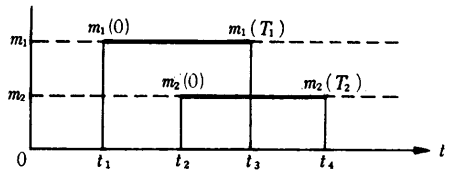

Fig. 4 Composition of two motions.

3. システム構成

3.1 CINEMA の構成モード

ここでは，前節で述べた方法による動画作成システム CINEMA の構成を説明する．必要とされる主な機能として，つぎのようなものがある．

(i) 動画の各構成要素を定義するための対話的処理を行う．

(ii) 動画記述用言語（後述）で書かれたプログラムを処理する．

(iii) (i), (ii)の結果，入力された各構成要素を内部データとして構造化する．

(iv) 動画作成のための図形の変換処理，各コマ生成および撮影装置の制御．

これらの機能を整理した形で実現するため，つぎのような構成方法を採った．すなわち，(i), (ii)で入力される動画構成要素は共通のデータ構造のもとに区別なく混在させ，これを一括してデータ処理用の汎用サブルーチン集合体 DSP (後述) に任せる．この DSP は汎用のデータ構造処理機能を有し，それ自身

* "Animation is not the art of DRAWINGS—that—move but the art of MOVEMENTS—that—are—drawn." Norman McLaren Canada).

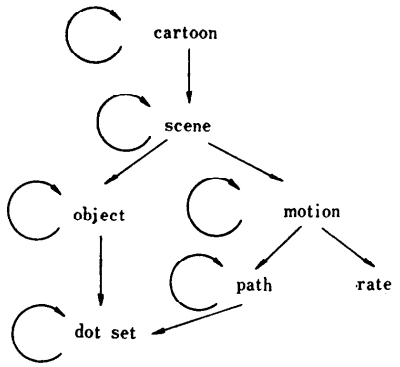

Fig. 5 Data hierarchy.

は CINEMA システムとは，完全に独立したものである．したがって，対話的ないしは記述言語によって行われる動画の定義作業と，実際に動画を生成・表示面上でテスト・撮影する作業とは，互いに切り離されており，両者間のデータの受け渡しは， DSP が行っている．また，使用計算機 (FACOM 270-30) の主記憶容量が小さいため (約 20 キロ語，16 ビット/語)，全体を 7 つの独立なプログラム (モード) に分け，主記憶常駐の mode 0 および DSP 以外は磁気ドラム (256 キロ語) に置き，実行時に呼び出すようにした．さらに，DSP の扱うデータもすべて磁気ドラムに格納される．

まず，システムで取り扱うデータは，前節の動画の構成要素にしたがって，Fig. 5 に示すような 7 種類とした．図中，A→Bは，AがBを直接の構成要素として含むことを表わす．また，セルフ・ループ (rate にはない) は，同じ種類の他のデータをその一部として含みうることを示す．ただし，同じデータを再帰的に含むことは許されない．1 つの動画を表わす内部データ項目は cartoon である．したがって，動画の定義完了後，撮影などに際しては，cartoon (につけられた識別名) に対して指示を与えることになる．また，cartoon は scene の集合である．前節で述べたように，scene は表示図形 (object) と変形動作 (motion) とのペアを基本とする階層を持った場面である．図に示す 7 種類の構成要素は，システムが DSP に依頼して構造化し格納するときの単位となるデータ項目であり，以後これらを '単位データ' と呼ぶ．したがって，cartoon も 1 つの単位データであり，後述するように，動画は cartoon を source* とし，各節に単位データが対応し，cycle** を持たない有向グラフ構造を持っている．すなわち，動画を定義する作業は，単位データを source とし cycle を持たない (i.e. 再帰的な包含関係はない) という条件を常に満たしつつ，最終的に 1 つの動画 (cartoon) を表わす有向グラフへ向けて，その部分グラフを形成してゆく作業であるといえる．

システムの構成を Fig. 6 に示す．システムの開始時にはまず mode 0 (nucleus mode) となり，表示面

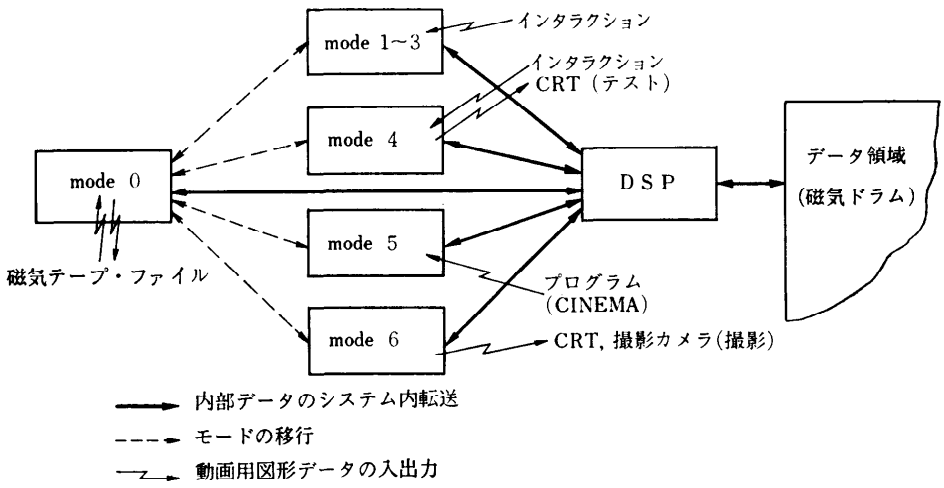

Fig. 6 Structure of CINEMA system.

* 有向グラフにおいて，他のすべての節へ到達可能な節のこと．ただし有向グラフとは，節 (node) の集合 V と節の順序対 (u,v) の集合とで与えられる $(u,v\in V)$．なお，順序対を弧 (arc) という．

** 節と弧よりなる交代列 $u_0x_1v_1\cdots x_nv_n$ で，1) $x_i=(v_{i-1},v_i)$, $1\leq i\leq n$, かつ 2) $v_0=v_n$ となるもの．

には各 mode への移行待ちを示す文字列 'SELECT A MODE' が出る．モード間の移行は，一度 mode 0に戻ってからファンクション・キーを用いて行われる．対話的操作（インタラクション）が行われるモードは，mode 1～4 である．対話的操作[9]は，表示面上のメニュー（ライト・ボタン）の選択，タイプライタからのコマンド入力などが大部分で，システム側から積極的に指示を要求してくるよう設計されている．対話操作の機能は，3次元動画を除けば，プログラムで表現できる全機能（付録）を含んでいる．また既成の図形を任意位置に再表示するのは随意である．各 mode において操作の中断・続行が可能である．Fig. 5 に示した単位データのうち，dot set は mode 1, object は mode 2, motion, path, rate は mode 3, scene は mode 4 で定義する．また，mode 4 では scene をテスト表示できる．動画記述言語で書かれたプログラムの処理は，mode 5 で行われる．したがって，mode 1～4 で定義された単位データ（の識別名）を，プログラムの中で用いたりすることは自由である．また cartoon は mode 6 で定義され，適当なコマ飛ばしを行ってテスト表示され，さらに撮影が行われる．

各モードは，FORTRAN を基に GSP (Graphic Subroutine Package) および DSP（後述）を用いて記述した．システム全体の大きさは約 11,000 ステートメント，約 1.5 人・年の作業量を要した．

3.2 動画記述言語の処理（mode 5）

表示図形の定義は対話操作が向いているが，変形動作の定義やシナリオを書く作業に当たる scene, cartoon の構成などはむしろ言葉で書く方が適している．そこで，動画記述用の言語（CINEMA と呼ぶ）を定義し，これによっても mode 1～4 と同様のことが定義できるようにした．CINEMA は単位データ定義のステートメントから成っている．そのシンタクスは付録に掲げてあるが，2. で述べた動画の構成方式，とくに動画の式をそのまま踏襲している．Fig. 5 に示す7種類の単位データは，O.〈id〉, M.〈id〉 などの形式で識別される．〈id〉は6文字以下の英数字列．処理プログラム（mode 5）は，この識別名でもって，hash 技術を用いた DSP に格納・検索を依頼するのである．

プログラムのエラーは，動画 (cartoon) を構成するグラフのなかで，エラーの発生したステートメントが表わす単位データに対応する節を source とする部分グラフの全体が定義されなかったことを意味する．また，エラーの発生した単位データの節を含むステートメントは警告が与えられる．逆に，エラーの発生しなかったステートメントは，正しく登録されているわけである．ゆえに，エラー発生の場合はシステムを初期化しないで，エラーの生じたステートメントのみを修正して mode 5 へ入力すればよい．エラー・メッセージは 79 種類．現在，3次元動画へ拡張され，15 個の書き換え規則が追加されている[9]．Fig. 7 は，3次元物体の透視図の動画を，シャッタを開放のまま数コマ撮影したものである．

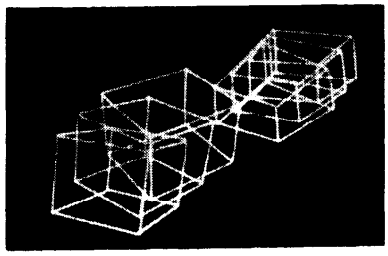

Fig. 7 A cartoon for 3-dimensional objects.

4. 内部構成

4.1 データ構造処理用サブルーチン・パッケージ (DSP)

動画作成システムの内部構成で，最も重要な問題はデータの記憶装置である．Fig. 5 に示す7種類の単位データは，磁気ドラムの同一領域内に混在して記憶される．システムは，これらの単位データを統一的に扱えるよう充分に汎用性があり，かつ登録・検索などが迅速に行えるような内部データ構造を要求する．とくに，何千コマもの動画を作成する場合など，データへのアクセスが頻繁に行われるので，後者の迅速性は最優先されるべき要求である．

これに対処するため，我々はすべてのデータを無 cycle 有向グラフとして把握し，これらを操作するプログラムの集合体 DSP (Data handling Subroutine Package) を，CINEMA とは独立に作成した．内部のデータ構造の実現に際し，上記の要求に応じるため，plex 構造[5]，hash 法[6] およびページングの技法を取り入れた．つぎに無 cycle 有向グラフに限った理由を述べる．Fig. 5 に示すように，ある単位データが複数個のより上位の単位データの要素になることは普通にありうるので，トリー構造では不十分である．また既述したように Fig. 5 のセルフ・ループにおいて，同一の単位データが再帰的な包含関係にあることは許

されないので，cycle が生じることはありえない．再帰的包含が存在する場合，単位データの展開が有限時間内に終わらなくなる．

DSP は，単位データの登録・検索・削除を円滑に行うためのサブルーチンの集合である．その機能としては，単位データの名前をハッシング[6]する，ハッシングされた値に対して新しい空領域を探す（登録），ハッシングされた値のデータを探す（検索），plex 構造へ組み込む，collision の発生の有無を調べるなどを備えている．

4.2 データ構造

DSP の扱うデータの内部構成（internal storage structure[5]）について述べる．まず磁気ドラム内の領域は，連想領域（association field, A 領域と略記，3×1024 語）とデータ領域（data field, D 領域と略記，1024×31 語）とに分けられる．主記憶上には，現在開いているページを格納する領域（current page area C領域と略記，1024 語）および実際にデータの合成・分解を行うバッファ領域（B領域と略記，100 語）がとられている．

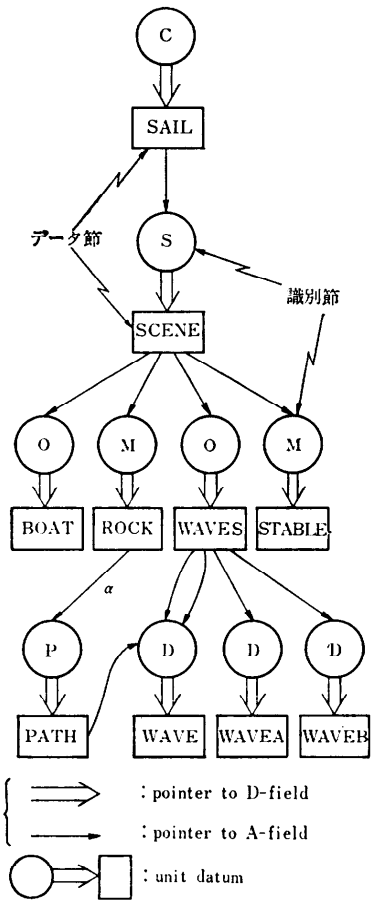

Fig. 8 Example of abstract data structure[5] handled by DSP.

DSP で取り扱われるデータの論理的な構造（abstract data structure[5]）は，Fig. 8 にあるような無 cycle 有向グラフである．図において，識別節（円形の節）は単位データの名前を，データ節（矩形の節）はその内容を表わす．識別節とデータ節のペアが，1つの単位データに対応する．識別節はA領域へ，データ節はD領域へ記憶される．したがって，Fig. 8 の各節を結ぶ矢印（弧）のうち，2重矢印はA領域からD領域へのポインタ，1重矢印はD領域からA領域へのポインタで実現される．単位データの操作は，識別名にハッシングを行い，まず，A領域に対して開始される．

単位データの識別名をKで表わそう．ハッシング関数には，互いに独立な h_A, h_N の2種類があり，Kをそれぞれ 10, 12 ビットのコードへ変換する．Kに関する情報としては，$h_A(K)$, $h_N(K)$ の計 22 ビットだけが用いられ，Kそのものは記憶領域には残らない．$h_A(K)$ は，A領域の 1024 個のエントリの1つを決定する．$h_N(K)$ は，K に代わって最終的な識別子となる．$h_A(K)$ をハッシュ番地，$h_N(K)$ をハッシュ名と呼ぼう．

システムの動作中，あるKという名前の単位データが登録されている状態で，新しく K′ を登録しようとした場合を考えよう．Collision の処理には直接連鎖法[6]を用いた．問題となるのは，$h_A(K')=h_A(K)$ かつ $h_N(K')=h_N(K)$ となる場合である．これは，A領域のすでにKが登録されているエントリを $h_A(K')$ が指定したために collision となり，さらに 12 ビットのハッシュ名も一致してしまった場合である．このとき，名前 K と K′ の識別ができなくなるので，K′ の登録は拒否される．これを“決定的 collision”が生じたという．このような場合システムは，K′ に代わる別の名前を要求してくる．この名前の付けかえ操作は，キーの同定時間の短縮および記憶領域の有効利用のために，インタラクティブなシステムであることを考慮して，あえて採用したものである．

Fig. 9 は，A, D 各領域間の関係を表わす．Fig. 8 の識別節は，A領域のエントリに対応する．データ節は，D領域内の bead 列[5]で，A領域を経由しないで

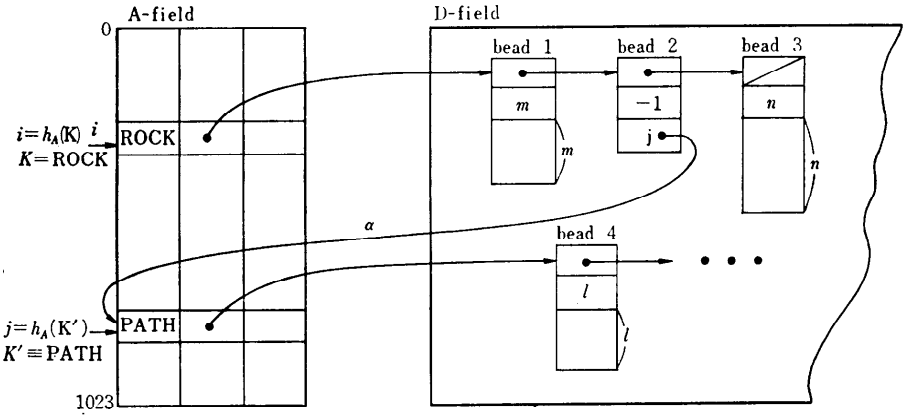

Fig. 9 Internal storage structure[5] handled by DSP.

結合されているもののひとかたまり（図では bead 1～3 がその例）が対応する．データの検索要求は単位データに関して生起する．このため単位データのエントリはすべてA領域につくられている．したがって，ある単位データの構成要素として他の単位データが含まれる場合は，特殊な bead（固定長）によって結びつけられる．Fig. 9 の例では，単位データ ROCK と PATH がその関係にある．このとき，まずA領域で PATH が検索され，その位置 j を用いて固定長の bead 2 が作られる．Fig. 8 と Fig. 9 の各 α が対応している．

1エントリは3語（16ビット/語）であり，Fig. 10 のように割り付けられている．

class（3ビット）は，単位データの種類を表わす．すなわち，001～111 で7種類の単位データを，また 000 で空エントリであることを表わす．

hash name（12ビット）は，h_N によって求められたハッシュ名．結局，データ検索時には第1語を調べればよい．

page（5ビット）および bead address（10ビット）は，それぞれD領域のページおよびページ内番地を表わす．page と bead address とが，D領域内のデータ本体へのポインタとなる．

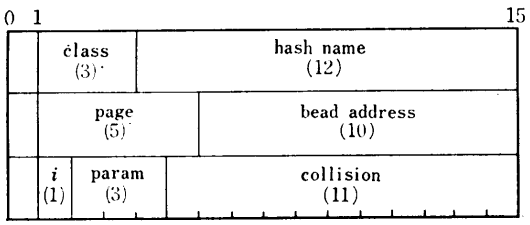

Fig. 10 Word format of entry of A-field.

i（1ビット）は，消去禁止フラグ，

param（3ビット）は，各種のパラメータ用，

collision（11ビット）は，collision が発生したときのA領域内の他のエントリを指すポインタである．

単位データを検索する場合は，A領域を経てD領域の該当ページを求め，主記憶のC領域にそのページが存在するかどうかを調べて，もし必要ならば新しいページをC領域へ転送する．実際のデータは，B領域へ bead 単位に数回に分けて取り出される．以上の操作は，すべて DSP が自動的に行う．検索時の磁気ドラムへの平均アクセス回数は，A領域での単位データの格納されている割合（表占有率）を a とすれば，約 $2+a/2$ 回である．

5. 例

Fig. 11 および Fig. 12 は Animator[4] で作られた2次元動画を CINEMA で作成した例である．図中，①では点 (x_1, y_1) を始点として角度 0 の方向へ間隔 d で並ぶ4点列 (i.e. P_1, P_2, P_3, P_4) を定義する．②は，①で定義された dot set WAVE を水平・垂直方向へ各々 $-x_2, -y_2$ だけ平行移動した点列 (i.e. P_5, P_6, P_7, P_8) を生成する．③の INT 2 (l, m, n) は，2つの dot set 間を，各々の l 番目の点から始めて，m 番目ごとに n 連続点同志 $(n \leq m)$ を線分で結ぶことを意味する．④において，P.〈id〉は図形〈id〉上の平行移動を意味する．ROT は回転を表わす．$(f, 0)$ は変形動作に要する時間が f コマ，時間的な相対変化率が線形であることを表わしている．⑤では，scene が変形動作と対象図形のペアの集合として定義

```
D.WAVE = VEC(x1,y1:0)*4(d)                                    ①
D.WAVEA = PAR(-x2,-y2)**D.WAVE                                ②
D.WAVEB = PAR(x2,-y2)**D.WAVE
O.WAVES = INT2(1,1,1)**(D.WAVE,D.WAVEA)                       ③
C         + INT2(1,1,1)**(D.WAVE,D.WAVEB);(x0,y0)
P.PATH = SD.WAVE
M.ROCK = ((P.PATH,ABS),(f,0))*(3(ROT.-25-ROT.25               ④
C          -ROT.25-ROT.-25),(f,0))
M.STABLE = (STABLE,f)
S.SCENE = (M.ROCK,O.BOAT) + (M.STABLE,O.WAVES)                ⑤
C.SAIL = (S.SCENE,0)
END
```

Fig. 11 CINEMA soutce program.

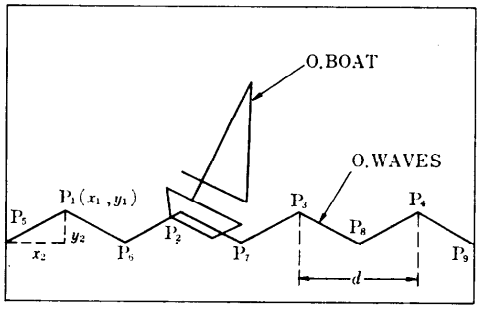

Fig. 12 Example similar to the one made by Animator[4].

されている.

また，書換え規則（付録）全体にわたって，$A**B$ の形は B に操作 A を施すことを表わす．さらに，＋，＊，－はそれぞれ，合併，並行的操作，継続的操作を意味するように統一した．なお，Fig. 8 のグラフは本例に含まれる単位データの関係を表わしたものである.

6. むすび

対話的操作およびプログラムの混用を許して動画を定義し，表示面上でテストし，最終的に 1 コマ連動撮影装置を制御して 16 mm フィルムを生成する汎用システムを作成した．動画の持つ階層的な構造の実現を中心として，システム設計の思想，変形動作，データ構造について述べたが，対話操作[9]，3 次元動画，撮影時間の短縮などの詳細は割愛した.

CINEMA 言語では単位データが O.〈id〉などの形で識別されるので，そのまま既存の言語 (FORTRAN など) に組みこんでも矛盾は生じない.

数本のフィルムを作成した結果，1 コマ生成の所要時間として 1.0～3.5 秒の値が得られた．本システムは，ブロック図などを用いた説明映画など教育的な目的に適していると思われる.

なお，本システムの作成には，京都大学大型計算機センター・システムⅢを使用した．また，1 コマ撮影装置の開発など種々援助をいただいた方々に感謝する.

参考文献

1) K. C. Knowlton: A computer technique for producing animated movies, Proc. SJCC, pp. 67-87 (1964).
2) J. Nolan & L. Yarbrough: An on-line computer drawing and animation system, Proc. IFIP Congress 68, pp. 605-610 (1968).
3) R. M. Baecker: Picture-driven animation, Proc. SJCC, pp. 273-288 (1969).
4) P. A. Talbot et al.: Animator: An On-line Two-dimensional Film Animation System, C. ACM, Vol. 14, No. 4, pp, 251-259 (1971).
5) F. R. A. Hopgood: Compiling Techniques, American Elsevier, New York (1969).
6) R. Morris: Scatter Storage Techniques, C. ACM, Vol. 11, No. 1, pp. 38-44 (1968).
7) P. Lucas et al.: Method and notation for the formal definition of programming languages, TR 25. 087, IBM Lab. Vienna, (1968, revised 1970).
8) 西原，萩原：計算機による動画の作成について，昭和 45 情報処理学会大会予稿，pp. 13-14 (1970).
9) 石垣：データ構造の形式的記述と相互表現性について，京都大学修士論文 (1973).

＜付録＞

プログラムのシンタクス

```
 1. <program> ::= <statement>●●●<end statement>
 2. <statement> ::= <cartoon st> | <scene st> | <motion st>
      | <rate st> | <path st> | <object st> | <dot array st>
 3. <end statement> ::= END
 4. <dot array st> ::=
      D.<id>={<basic dot array> | {+●<dot expression>●●●}}
 5. <basic dot array> ::= <direct data>
      |VEC(<data element>:<phi>)*<integer>({-●<interval>●●●})
      |ARC(<data element>:<phi>,<radius>)
       *<integer>({-●<theta>●●●})
 6. <dot expression> ::= [*●<transform operator>●●●]**D.<id>
 7. <transform operator> ::=
      ROT(<theta>,<integer>) | <parallel>
 8. <object st> ::= O.<id>=<object expression>;(<data element>).
                    [;A<rectangle>]
 9. <object expression> ::= {+●<object unit>●●●}
10. <object unit> ::=
      [{*●<basic transform>●●●}**]{O. | D. | CD.}<id>
      | {<1 dot connect> | ({+●<1 dot connect>●●●})}**D.<id>
      | {<2 dots connect> | ({+●<2 dots connect>●●●})}
        **(D.<id>,D.<id>)
      | SET(<subst object>-<direct data>) | <direct data>
11. <basic transform> ::=
      Z(<data element>) | ROT.<theta> | <parallel>
12. <parallel> ::= PAR(<data element>)
13. <1 dot connect> ::= <1 dot op>[*<parallel>]
14. <1 dot op> ::= INT(<intermittent points>)
                 | RAD(<integer>:<intermittent points>)
                 | REP(<subst object>,<theta>:
                   <intermittent points>,<points number>)
                 | SUB(<subst object>:{,●<integer>●●●})
15. <2 dots connect> ::= <2 dots op>[*<parallel>]
16. <2 dots op> ::= INT2(<intermittent points>)
                  | RAD2(<integer>:<intermittent points>)
17. <intermittent points> ::=
      <beginning pt>,<every pt>,<connected pts>
18. <subst object> ::= O.<id> | CIR.<radius> | P
19. <path statement> ::= P.<id>=<path>
20. <path> ::= [<path expression>-]<standard path>
               [-<path expression>]
21. <path expression> ::= [-●<path element>●●●]
22. <standard path> ::= S{<path element> | (<direct data>)}
23. <path element> ::= D.<id> | P.<id>
24. <direct data> ::=
      <data element>{[:<data element>]
      | [:<direction><integer>[<direction><integer>]]}●●●
25. <direction> ::= L | R | U | D
26. <rate st> ::= R.<id>={,●<rate expression>●●●}
27. <rate expression> ::= (<data element>)[,<integer>]●●●
28. <motion st> ::= M.<id>={--●<motion expression>●●●}
29. <motion expression> ::= {*●{<motion element> | M.<id>}●●●}
30. <motion element> ::=
      [<integer>](<motion component>,(<frame>,<rate>))
      | (ST[ABLE],<frame>) | (INV[ISIBLE],<frame>-<frame>)
31. <motion component> ::= {+●{[<integer>](<basic motion>)}●●●}
32. <rate> ::= <integer> | R.<id>
33. <basic motion> ::= P.<id>,{A[BS] | R[EL]} | SHAPE
      | {-●{R.<theta>}●●●} | {-●{Z(<data element>)}●●●}
34. <data element> ::= <integer>,<integer>
35. <scene st> ::= S.<id>={+●<scene>●●●}
36. <scene> ::= {-●<scene element>●●●}
37. <scene element> ::=
      S.<id> | (M.<id>,{+●{O.<id> | <scene element>}●●●})
38. <cartoon st> ::= C.<id>={+●<cartoon element>●●●}
39. <cartoon element> ::=
      C.<id> | (S.<id>[(<priority>)],<time>)
```

(昭和49年1月12日受付)