3Q-10

コンピュータグラフィックスとプログラミング言語の機能

石原亘
(京都芸術短期大学・ＣＧ研究室)

1　緒言

プログラミング言語は、情報処理システムにとって受理可能な形式であるというだけではなく、プログラマに対しては思考のガイドラインとして機能する。したがって、ＣＧ（＝コンピュータグラフィックス）システムの構築においても、使用するプログラミング言語の表現能力が作業効果に大きく影響するであろうことが予想される。

しかし、既存のプログラミング言語の多くは、汎用性を強調して設計されており、ＣＧに適しているとはいい難い。すなわち

○　本来の作業のほかに、意図した処理を言語の枠組みに合わせて翻訳する作業が必要になる。しかも、この２者を分離することは極めて困難である。

○　プログラムの文面から処理の内容を汲み取ることが難しい。

○　発想が、使用している言語の枠組みに支配される傾向がある。

などの弊害を生じやすい。

見方を変えれば、これらの問題は現時点での情報処理技術の制約として本質的であると言えなくもない。しかし、ＣＧにおいてはこれらが必要以上に作業上の制約になっている。

より効果的にＣＧシステムが構築できるようにするためには、必要な処理を、記法上の問題にとらわれずに自然な形式で表現でき、しかも実行可能な表記法が必要である。本稿では、ＣＧ用プログラミング言語に望まれる表現能力とは何かを考察する。

なお、本研究では、モデルやその変換など主として描図以前の処理について検討する。すでに各方面で研究がなされているように、図形の読み／描きの記述もＣＧ言語にとっては重要な要素であるが、これらについては全く別のアプローチからの検討が必要になろう。

2　要求分析

図形処理のイメージを明確にするため、例として譜１を掲げる。譜１は、Ｚ軸方向に３個つながった立方体を描く処理を、パスカル語プログラム風に表現したものである。図形は頂点表（座標の表）と稜線表（端点の番号の表）とのレコードとして表現されている。

```
program 例;

type
    頂点=record
        x,y,z: real end;
    稜線=record
        start,goal,color: integer end;
    図形=record
        頂点表: list of 頂点;
        稜線表: list of 稜線 end;

var 深度,倍率: real; 背景色,K: integer 原型,F: 図形;

    …;

function 平行移動(ＯＢＪ: 図形; ＤＸ,ＤＹ,ＤＺ: real): 図形;
var F: 図形; R: 頂点;
begin
Ｆ.頂点表:=空;
(ＯＢＪ.頂点表)を巻き戻す;
while (ＯＢＪ.頂点表)がまだ続いている do
    begin
    (ＯＢＪ.頂点表)を次に進める;
    R:=(ＯＢＪ.頂点表)の注視場所の内容;
    Ｆ.頂点表に[x=R.x+ＤＸ,y=R.y+ＤＹ,z=R.z+ＤＺ]を挿入する;
    (Ｆ.頂点表)を次に進める
    end;
Ｆ.稜線表:=ＯＢＪ.稜線表
end;

procedure 表示(ＯＢＪ: 図形; ＤＰＴ,ＳＣＬ: real; ＢＧＣ: integer);
var F: 図形; R: 稜線; S,G: 頂点;
begin
ＢＧＣでスクリーンを塗り潰す;
(ＯＢＪ.稜線表)を巻き戻す;
while (ＯＢＪ.稜線表)がまだ続いている do
    begin
    (ＯＢＪ.稜線表)を次に進める;
    R:=(ＯＢＪ.稜線表)の注視場所の内容;
    S:=(ＯＢＪ.頂点表)の(R.ｓｔａｒｔ)番目の内容;
    G:=(ＯＢＪ.頂点表)の(R.ｇｏａｌ )番目の内容;
    ＳとＧとを結ぶ線分のＤＰＴとＳＣＬとによる透視像をスクリーンに引く
    end
end;

begin
深度:=-8.0;
倍率:=100.0;
背景色:=0;
原型:=立方体;
Ｆ:=空;
for K:=-1 to 1 do
    Ｆ:=併合(Ｆ,平行移動(原型,float(Ｋ),0.0,0.0));
表示(Ｆ,深度,倍率,背景色)
end.
```

譜１

譜１を見ると、ＣＧに独特なものとして以下のような要求を見出すことができるであろう。

D1　要素型が異なる複合型の操作

要素型だけが異なる複合型（少なくともベクトル・行列・リスト）に対しては、相似の演算を共通の記法で表現することが可能でなければならない。

譜１には現れていないが、ＣＧで扱うベクトル／行列には、要素型が実数のもの（おもにモデルに関連する）と整数／基数のもの（おもに描図に関連する）とがある。しかも、通常の処理ではこの双方を同時に扱わなければならない。

要素型の別にかかわらず、ベクトル／行列に対する操作としては、和・差・内積などが考えられる。既存の多くの言語では、それぞれの要素型に対して関数あるいは手続きを別々に準備し使い分ける必要がある。しかもそれらの関数や手続きに対しては、それぞれ異なった名称を与えなければならない。このことは、相似の演算であっても各要素型ごとに表記上の区別をいちいち理解しなければならなくなることを意味する。

ＣＧではリストも処理の対象になる。ＣＧで用いるリストには、要素型が頂点、稜線、あるいはそれらの複合体である図形など、複数のものが考えられるため、これに対しても同様の問題が発生する。たとえば、譜１の平行移動の定義と表示手続きの定義には相似の処理が用いられている。

Programming Language Descriptability on Computer Graphics

Wataru ISIHARA
Kyoto College of Art

[D2] 複合型を結果型とする関数
図形や頂点・稜線のような複合型データを式として表現できるように、これらを結果として返す関数が定義できなければならない。
譜1には、図形型（頂点表と稜線表とをレコード化によって結合したもの）のデータを結果として返す関数として立方体や平行移動が現れる。しかし、多くのプログラミング言語では関数の結果型を一部の単純な型だけに制限しているため、同様の処理を表現しても譜1より可読性において劣る。

[D3] 複合型データの要素列挙による表現
たとえば、譜1の平行移動関数の定義にある「各座標がそれぞれR.x+DX, R.y+DY, R.z+DZである頂点」といったものが、自然な形式で記述できなければいけない。

[D4] 要素へのアクセス
図形など複合型のデータに対し、その要素が容易にアクセスできなければならない。

[D5] 不定長データ列
図形は頂点・稜線などの要素から構成されているが、これらの個数は特定することができない。したがって、不定長データ列（要素の型は列内で一定としてもいい）が扱える必要がある。

3 既存の言語とCGプログラミング—パスカル語の場合

以上に掲げた要求が既存の言語にはおいてどう扱われているか、パスカル語を例に検討する。

[D1] 要素型が異なる複合型の操作
パスカル語においては、異なる基底型ごとに手続き（関数として実現するのには困難がともなう）を作り分け、それらの名称が衝突しないように「R…」とか「…I」といった接頭／接尾辞を名称に付けて使い分ける必要がある。

[D2] 複合型を結果型とする関数
パスカル語では、関数の結果の型はスカラーかポインタに制限されている。このため、譜1の立方体や平行移動の関数をそのまま実現することはできない。
ひとつの解決は、関数としてでなく手続きとしてこれらを実現することである。しかしこの場合、関数の評価の積み重ね（譜1の主譜）によって処理を表現することができなくなる。
もうひとつの方法は、結果そのものの代わりに結果の格納場所を示すポインタを返す関数を作ることである。この場合、図形など関数の結果になりうる型は、複合型そのものでなく目的の型のデータを指すポインタ型として定義する必要がある。しかし、この方法にもふたつの欠点がある。
欠点のひとつは、ごみが溜まることである。特にCGにおいてはアニメーションのように、何度も繰り返して関数を
呼び出す処理が多く、ヒープエリア不足のために途中で実行を続けられなくなることがある。参照したあとでdispose を用いて解放することもできるが、そのためには評価の結果をいったん変数に記録しておく必要がある。この場合も手続きを用いる場合と同様に、関数式によって処理を表現できるメリットが失われる。ただし、ごみ集めを自動的に行う処理系ではこのことは問題ではない。
もうひとつの問題は多重参照である。すなわち、ポインタ経由のデータが複数の変数に代入されている場合、ある変数の内容だけを更新したつもりで、ほかの変数の内容まで更新してしまうことが起こりうる。これを避けるには、内容の複写による代入を行う手続きをほかに作り、これを「:=」の代わりに使うようにするなどの対策をとる必要がある。しかもこの場合、代入手続きは各型ごとに別々のものを準備しなければならない。

[D3] 複合型データの要素列挙による表現
パスカル語では配列を要素の列挙によって表現することはできない。withを用いれば、レコードはかなり簡潔に表現できる。

[D4] 要素へのアクセス
パスカルでは、複合型データの要素に対するアクセスを「.」（レコード）・「[]」（配列）等によって記述できる。しかし、これらは変数名など限られたものにしか適用できない。したがってデータが式によって表現されている場合には、そのデータを一時的な変数にいったん代入し、その変数に対して「.」や「[]」を適用するか、あるいはこれらを適用するだけの関数を作るかしなければならない。

[D5] 不定長データ列
パスカルでも、十分長い配列またはリストを代用すれば不定長データ列を扱うことができる。しかし、配列を用いる場合は、プログラム側で配列の有効長さを管理しなければならない。また、扱う図形が変わるたび、メモリの有効利用のためプログラムの全面的な書き直しが必要になる。一方リストは、プログラマがポインタを用いて各要素型ごとに定義することになっている。このため、[D1]で述べた要素型に係わる問題が生じる。
以上のように、パスカル語では、機械的な読み替えによってある程度自然な形式でプログラムを記述することができるが、さまざまな制約が発生し十分とはいえない。
なお特にパスカル語の場合、多くの処理系では、CGプリミティブとして関数・手続き群を準備しても、これらを自／手動によってインサート（インクルード）ファイルからソース内に展開しなければならない。このことも（言語の記述能力とは異なった側面に属するが）パスカル語を用いてCGシステムを開発する際の障害になっている。

4 解決の戦略

2に掲げた要求を満足しうる新しいプログラミング言語としては、要求を包含するより一般的な記述能力をもつ言語と、要求を満足するミニマルな機能をもつ言語とのふたつのタイプが考えられる。以下では特に、[D1]に対する解決について検討する。

[1] マキシマルな解決＝一般的な機能の埋め込み
一般性を尊重した場合、少なくともふた通りの解決が考えられる。そのひとつは、エーダのようにジェネリックな関数／手続きを定義できるようにする方法であり、もうひとつは、データ型間に階層を設け、上位階層から下位に属性を継承できるようにする（スモールトークなど）方法である。

[2] ミニマルな解決＝特殊な機能の埋め込み
現時点では、CGに必要な最少限の概念だけを組み込んだ言語を構成してミニマルな解決を図ることも意義がある。
そのような言語のうち、最も有効と考えられるのは、たとえばパスカル語にリストを組み入れ、配列やレコードと同様に新しい型を導出する手段として利用できるようにすることである。このリストは、通常のリストとしての特性に加えて
○アイテム型は制約されない。少なくとも、スカラー型およびそれらをレコード／配列／リスト化して導出した型程度のものはアイテム型として使用できる。
○リストに対する基本操作を標準関数／手続きとして装備する。これらは、アイテム型に関係なく共通の名称のもとに使用できる。
○代入は他の型と同様に「:=」によって記述できる。通常のプログラミングでポインタを用いてリストを構成する場合と異なり、共有ではなく複写によって代入が行われる。
○動標（カーソル）はデータ側にもち、que・scroll などの標準手続きで移動させることができる。
○高速な順処理のほかに乱処理も可能（時間効率が犠牲になるのはやむを得ない）である。
これらの機能があれば、CGで扱うたいていの処理は極めて自然に表現することができる。図形型もふたつのリストからなるレコードとして定義できる（譜1）。