第6章 Graf3Dの試用

特典PDF

当時のQuickDrawに、曲がりなりにも3D描画機能が備わっていたことは、あまり知られていないかもしれません。私自身も、その存在自体は耳にしたことがあったものの、実際にプログラミングしてみたことはなく、今回、本書の執筆を始めるまでの何十年間かは、その名前も含めてすっかり忘却の彼方のものとなっていました。それが、この章で取り上げる「Graf3D」です。そのネーミングからしてLisa由来のものと分かりますが、QuickDraw本体とは別物で、由来も異なるもののように思えます。Inside Macintoshにも、その存在も含めて記載されていないこともあり、資料もあまり残っていないのが現状です。また、QuickDrawのソースコードに含まれていながら、Macintosh本体のROMには組み込まれておらず、必要に応じてアプリケーションプログラムとリンクして利用するライブラリとして提供されていました。本書で利用しているTurbo Pascalでも、1つのUNITとして提供されていて、最初からTurbo Pascalのデフォルトの UNITとして組み込まれています。一通りGraf3Dの機能と使い方を確認してから、ここでもTurbo Pascalを使って、実際にGraf3Dを利用したプログラムを動かしてみましょう。

6-1 Graf3Dとは? ・・・・・・・・・・・・・・・・・・・・ 2
6-2 Graf3Dの純正サンプルプログラムとAPI仕様 ・・・・ 4
6-3 FixMathの概要・・・・・・・・・・・・・・・・・・ 13
6-4 Graf3Dのサンプルプログラム試作 ・・・・・・・・ 15

6-1 Graf3Dとは?

公開されたQuickDrawのソースコードの中に、そのインターフェースに加えて実装部分のソースコードも含まれていることからすれば、Graf3DはQuickDrawの一部と考えることも可能です。しかし、すべてPascalで記述されていることや、主要な変数の型が浮動小数点数を表すREALになっていること、三角関数のSINやCOSを使っていることなど、QuickDraw本体とは、かなり異質な存在となっています。ちなみに、そのPascalのソースコードは500行にも満たないもので、読んで理解しようと思えば、それほど難しいものでもないでしょう。本書では、やはり原則に従って、そのソースコードは転載しません。

ネーミングについても、QuickDrawが当初「LisaGraf」と呼ばれていたことと考え合わせると、いかにも関連が強そうな「Graf」を含んでいて、起源が古いことを匂わせています。そのGrafの部分をQuickDrawに置き換えれば、「QuickDraw3D」になりますが、後に登場した「QuickDraw 3D」とは別物で、技術的にはほとんど、あるいはまったく関係ないものと考えたほうが無難でしょう。

Graf3Dについて、Appleの開発者向けのドキュメントに記述が現れるのは、私が確認している範囲では、初代Macintoshの発売から約1ヶ月後の1984年2月14日に発行された「Lisa Development System / Internals Documentation」が最初です。当初のMacintoshには、セルフの開発環境は存在せず、Macintosh用アプリケーションはLisa上で開発することになっていたので、「Lisa Development System」向けのドキュメントに記述されるのは、何の不思議もありません。ただし、すべてPascalで記述されていることを考えても、Lisa用とMacintosh用のGraf3Dは、共通のものだったと考えられます。そこに記されている文章を、抄訳すると、以下のようになります。

> 3次元グラフィック機能も利用可能です。QuickDrawルーチンの上で動作するGraf3Dと呼ばれるUNITによるものです。Graf3Dでは、本物のパースペクティブの中で、実数の変数とワールド座標系を使って、3次元オブジェクトを描くことが可能となります。

また、それからほぼ1年後の、1985年2月15日に発行されたMacintosh用アプリケーションの開発者向けに提供されたソフトウェアのドキュメント「Software Supplement」には、固定小数点数を使ったMacintosh用のGraf3Dについての記述が登場します。遅くともその時点では、固定小数点ライブラリのFixMathを使う

ように書き換えられたGraf3Dがリリースされていたのでしょう。その記述の一部を抄訳しておきましょう。

> Lisa Workshopでのみ利用可能です。このバージョンは固定小数点演算を使っていて、これまでのGraf3Dのバージョンで使われていた浮動小数点演算に比べて、より小さく、より速くなっています。

さらにそのSoftware Supplementの改定版で、1985年7月2日に発行された「Software Supplement Update」には、Macintosh上のさまざまな環境でGraf3Dの利用が可能となったことが記されています。

> Graf3Dは、Lisa WorkshopやMDSに加え、MDSリンカーを利用するMacintosh上の他の言語でも利用できるようになりました。

その改訂版には、Lisa Workshop上のPascalのサンプルプログラムとして、Boxes、BoxSphere、sineGridという3つが付属していることも明記されています。実は、これら3つのソースコードは、Lisa用Pascalだけでなく、例えば本書で利用しているTurbo Pascalにも移植され、Turbo Pascalのプログラムディスクに含まれていたのです。移植と言っても、おそらく、プログラム本体にはほとんど変更の必要はなかったものと思われます。それらのサンプルプログラムがどんなものなのか、次の節で実際に動かして見てみましょう。

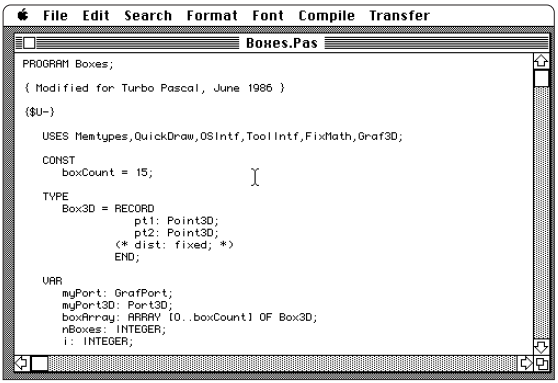
以上のように、初代Macintoshが発売されてから1年半ほどの間のGraf3Dそのものの進化や、動作環境の拡がりについて、ざっと把握することができました。上に挙げたサンプルプログラムもそうですが、実際にGraf3Dが外部の開発者にも利用されるようになったのは、固定小数点化がなされてからのことでしょう。Turbo PascalにUNITとして組み込まれているバージョンも、もちろん固定小数点版の方です。そういうわけで、QuickDrawのソースコードに含まれている浮動小数点版からは少し離れてしまうことになりますが、これ以降は、固定小数点版のGraf3Dを中心にして述べていくことにします。

6-2 Graf3Dの純正サンプルプログラムとAPI仕様

Graf3Dのプログラミングのためのインターフェースを確認する前に、それを使ってどのようなことが可能なのか、どのような3次元グラフィックを描くことができるのか、Apple純正のサンプルプログラムを実際に動かして、感触をつかんでおくことにしましょう。ここでは、3種の純正サンプルプログラムをTurbo Pascalに移植したものを動かして、実際の画面を見ていくことにします。

Boxes

Boxesは、画面に複数の立体的な箱を複数並べて描くものです。動かしてみる前に、ソースコードの先頭部分だけ、初代Macintoshの画面上で確認しておきましょう。



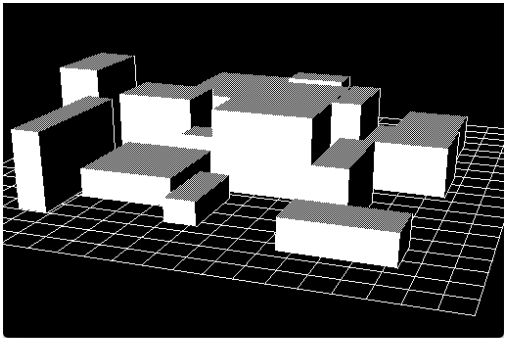
Graf3Dの純正サンプルプログラム、BoxesをTurbo Pascalに移植したもの先頭部分

画面からまず分かることは、このプログラムは、1986年にTurbo Pascal用に移植されたものであることです。これは、Graf3Dの歴史には関係なく、単にTurbo Pascalが1986年に製品として発売されたものであることを反映したものでしょう。

また、Turbo PascalのプログラムからGraf3Dを利用するためには、Macintoshアプリケーションとして一般的なUNIT（Memtypes、QuickDraw、OSIntf、ToolIntf）に加えて、FixMathとGraf3Dという、2つのUNITをUSESで宣言しなければならないことも分かります。このうちFixMathは、Toolboxにはじめから定義されている一般的な32ビットの固定小数点、Fixedや、その演算機能だけでなく、新たに別の形式の固定小数点数の型と、その演算機能を定義するものです。仮にユーザーのプログラム内でFixMathを使っていなくても、Graf3Dを使うためには、必ずその前に使用を宣言しなければなりません。それは、Graf3D自体がFixMathに依存しているからです。つまり、Graf3Dを使うには、「FixMath, Graf3D」を、こ

の順番のままUSESで宣言しなければならない、ということになります。

これらの2つのUNIT、FixMathとGraf3Dについては、後で、Turbo Pascal用のインターフェースを転載しますが、このサンプルプログラムの続きのソースコードを掲載するわけにもいかないので、これを走らせ、動作画面を確認しましょう。上のプログラムの部分からも分かるように、箱は全部で16個（0..boxCount）表示されるはずですが、手前の箱に隠れて、見えない箱もあるでしょう。それらの箱は、グリッド状の床の上に並べられ、箱の前面は白、上面は薄いグレー、側面は黒で塗られているように見えます。



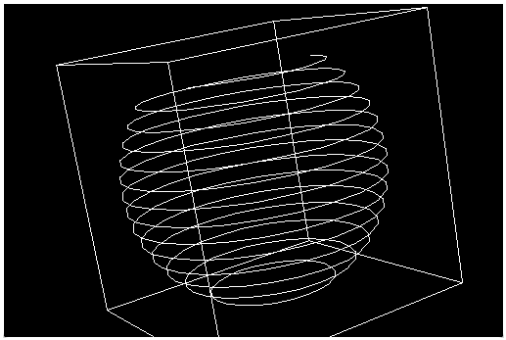
Graf3Dの純正サンプルプログラム、BoxesをTurbo Pascalに移植したものの表示画面

このプログラムは、Graf3Dのサンプルプログラムとして代表的なもので、巧妙にできています。というのも、Graf3Dで扱える3Dオブジェクトは、実は「点」だけで、「線」や「面」はありません。線は、3Dの点（Point3D）の間に直線を引くことで表現しています。このためには、MoveTo3DやLineTo3Dといったプロシージャが用意されていますが、それは引数が3軸の座標点になるからであって、実際には内部でQuickDraw本体の`MoveTo`や`LineTo`を使って描画することになります。またこのプログラムの画面を見ると、3次元空間で面を塗りつぶすことができているように思えますが、実際にはMoveTo3DやLineTo3Dを実行する前に、1つの面ごとにリージョンをオープンしておき、その面のワイヤーフレームの描画が終わったら、その2次元のリージョンを閉じて塗りつぶすことで、3Dの面を塗りつぶしているように見せているのです。また、当然ながら隠線処理や隠面処理もないので、面は、手前から見て正面、上面、右側面の3つしか描いていません。また奥の方の箱から順に描いて、手前の箱で上書きすることで、天然の隠面処理を実現していると考えることができるでしょう。

いずれにしてもこのプログラムは、3Dグラフィックライブラリとしては、かなり機能の限られたGraf3Dの欠点を覆い隠し、限定的なレイアウトながら、効果的なデモプログラムとなっています。

BoxSphere

このプログラムの名前は、前半が「箱」、後半は「球面」という意味で、いったいどんなものが描かれるのか、名前からは想像しにくいかもしれません。実際には、箱の中に、球面を表現したような、ワイヤーフレームの3D画像を描くものです。



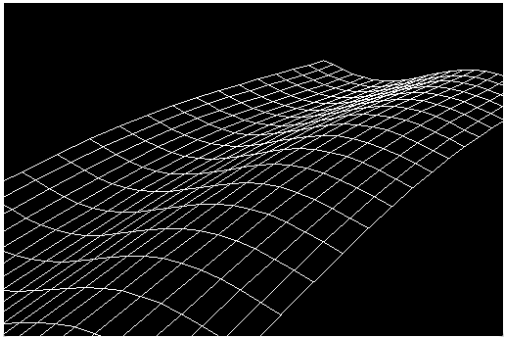
Graf3Dの純正サンプルプログラム、BoxSphereをTurbo Pascalに移植したものの表示画面

箱の方はともかく、球面と言っても、円の直径を少しずつ変化させ、輪切りの縁のような線で構成された擬似的な球面を描くものです。描く過程もはっきり見えるので、螺旋状に描いているように勘違いしてしまいそうですが、それぞれ独立した円を15個ほど、3D空間内でずらしながら重ねるように描きます。1回描き終わると、箱と中の球を、少しずつ傾けながら描き続けるので、デモ効果はあるでしょう。

このプログラムでは、円の輪郭の座標を計算するために、三角関数のSINを使っています。これはFixMathの中に定義されている`FracSin`というファンクションを利用しています。これに限らず、FixMathで使えるファンクションについては、後で簡単に説明します。

SineGrid

このプログラムは、起伏のある床のような、歪んだグリッドだけを描くものです。1回だけ描くと動作が停止するので、マウスボタンをクリックして戻ります。



Graf3Dの純正サンプルプログラム、SineGridをTurbo Pascalに移植したものの表示画面

このプログラムでは、やはりFixMathに定義された`FracSin`や`FracCos`といった三角関数を利用してグリッドの交点の3次元座標を計算し、それらの間をMoveTo3DやLineTo3Dでつなぐことで描いています。

これらのサンプルプログラムが、実際にどのように3Dグラフィックを描いているのかについては、ソースコードを読んでいただくのが最も確実です。勝手にソースコードを転載するわけにもいかないので、気になる方は、Turbo Pascalを入手して、付属するソースコードを確認し、実際に動かしてみることをお勧めします。

Graf3DのAPI仕様

Graf3Dのプログラミングを試す前に、そのAPIを確認しましょう。公開されたQuickDrawのソースコードに含まれている「Graf3D.p」は、「Graf3D」のUNITの定義という形式になっていて、その先頭部分には、INTERFACEが置かれ、このUNITで利用できる型、プロシージャ、ファンクション、定数、変数の名前や形式が一通り記述されています。そのオリジナルのUNITでは、実数としてPascalの浮動小数点数REAL型を使っています。しかし、Turbo Pascalに付属するUNITは、それがすべてFixed型に変更されたものとなっています。このFixedは、32ビットの固定小数点数で、上位16ビットが整数部、下位16ビットが小数部というものでした。

このオリジナルのREALタイプのGraf3Dと、Fixedタイプの Graf3Dのインターフェースを比べると、REALがFixedになった以外、ほとんど違いはないのですが、後者には1つだけ追加されたプロシージャがあります。それはInitGrf3D（「InitGraf3D」ではないことに注意）という、いかにも重要そうなものです。これは、Boxesなど、Apple純正のサンプルプログラムでも使っているので、Turbo Pascalが独自に追加したものではなく、Graf3Dを固定小数点化するタイミングあたりで、Apple自身が追加したものと思われます。実は、必ずしもこの初期化プロシージャを使わなくても、Graf3Dを利用したプログラムは動作するのですが、Graf3Dの初期化に関わるプロシージャの実行と、2DのQuickDrawの初期化に関わるプロシージャの実行の順番、その他の手順によっては、システムエラーが発生するなど、動作に不安定な部分が見受けられます。そういった初期化の不確実さを緩和するために追加されたものなのかもしれません。

ここではまず、実際に動作環境が用意できている、Turbo Pascalに付属するGraf3Dのインターフェースを、Turbo Pascalのリファレンス・マニュアルから転載しておくことにします。

```
unit Graf3D(-13);

interface

uses MemTypes,QuickDraw,FixMath;

const radConst = 3754936; {radConst 57.29578}

type Point3D=record
        x: fixed;
        y: fixed;
        z: fixed;
    end;

    Point2D=record
        x: fixed;
        y: fixed;
    end;

XfMatrix  = array [0..3,0..3] of fixed;
Port3DPtr = ^Port3D;
Port3D    = record
                GrPort:                     GrafPtr;
                viewRect:                   Rect;
                xLeft,yTop,xRight,yBottom:  fixed;
                pen,penPrime,eye:           Point3D;
                hSize,vSize:                fixed;
                hCenter,vCenter:            fixed;
                xCotan,yCotan:              fixed;
                ident:                      Boolean;
                xForm:                      XfMatrix;
            end;

var thePort3D: Port3DPtr;

procedure InitGrf3D     (globalPtr: Ptr); external;
procedure Open3DPort    (port: Port3DPtr); external;
procedure SetPort3D     (port: Port3DPtr); external;
procedure GetPort3D     (var port: Port3DPtr); external;

procedure MoveTo2D      (x,y: fixed); external;
procedure MoveTo3D      (x,y,z: fixed); external;
procedure LineTo2D      (x,y: fixed); external;
procedure LineTo3D      (x,y,z: fixed); external;
procedure Move2D        (dx,dy: fixed); external;
procedure Move3D        (dx,dy,dz: fixed); external;
procedure Line2D        (dx,dy: fixed); external;
procedure Line3D        (dx,dy,dz: fixed); external;

procedure ViewPort      (r: Rect); external;
procedure LookAt        (left,top,right,bottom: fixed); external;
procedure ViewAngle     (angle: fixed); external;
procedure Identity;     external;
procedure Scale         (xFactor,yFactor,zFactor: fixed); external;
procedure Translate     (dx,dy,dz: fixed); external;
procedure Pitch         (xAngle: fixed); external;
procedure Yaw           (yAngle: fixed); external;
procedure Roll          (zAngle: fixed); external;
procedure Skew          (zAngle: fixed); external;
procedure TransForm     (src: Point3D; var dst: Point3D); external;
```

```
function Clip3D          (src1,src2: Point3D; var dst1,dst2: Point): Boolean;
                         external;
procedure SetPt3D        (var pt3D: Point3D; x,y,z: fixed); external;
procedure SetPt2D        (var pt2D: Point2D; x,y: fixed); external;
```

すでに他の環境で3Dグラフィックのプログラミング経験がある人なら、このインターフェースを見ただけで、Graf3Dのシンプルな機能が把握でき、使い方もだいたい想像できるのではないでしょうか。一方、3Dグラフィックに慣れていない人には、ここに登場するプロシージャやファンクションの概要の説明を読んだだけでは、自在にプログラミングできるようにはならないかもしれません。以下、大きな機能別に、ざっと説明していきますが、ここでは3Dグラフィックの基礎については省くので、必要に応じて他の文献を当たっていただければと思います。

APIとして用意されたプロシージャやファンクションの説明に入る前に、1つだけ補足しておきます。上に示したインターフェースの先頭部分で定義されている定数、radConstについてです。名前からして、ラジアンについての定数であることは分かりますが、その値が3754936と言われても、ピンと来ない人も多いでしょう。しかしヒントが、コメントにあります。それは、57.29578という実数値です。これは、180をπで割った値です。ラジアンでは、デグリーの360度がラジアンでは2πであることから、デグリー値をこの57.29578で割れば、ラジアン値が求まります。そして、定数値の3754936というのは、実数の57.29578をFixed型の固定小数点で表現したものを、LONGINTとして見た場合の値なのです。単純に57.29578に$10000（65536）を掛ければ求まります。

この定数が用意されているということは、Graf3Dでの角度は、QuickDrawのようなデグリーではなく、ラジアンで指定するのだろうと、勘の良い人は考えるでしょうが、そこに罠が存在します。元の浮動小数点版のGraf3Dでは、確かに角度はラジアンで指定していましたが、固定小数点版のGraf3Dでは、角度はデグリーで指定するようになっています。ではなぜ、ここにradConstが定義されているのかと言えば、それは元のGraf3Dとの互換性を維持するためとしか考えられません。ただし、後で取り上げるFixMathに含まれている三角関数のファンクション、`FracCOS`と`FracSIN`では、紛らわしいことに角度をラジアンで指定することになっています。それらを利用する場合、デグリーをラジアンに変換する必要があれば、このradConstが定数として使えます。その際の割り算には、やはりFixMathに含まれているFixed同士の割り算、`FixDiv`が使えるでしょう。その結果はFixedになりますが、`FracCOS`と`FracSIN`の引数の型はFixedなのです。

それでは、Graf3Dに定義されているプロシージャ、ファンクションについて、個別にざっと説明していきます。

初期化と Port3D 関連

PROCEDURE InitGraf3D(globalPtr : Ptr);

本家の QuickDraw の `InitGraf` に相当する、Graf3D 自体の初期化ルーチンです。実装部分のソースコードのある REAL 版の Graf3D には用意されていなかったプロシージャなので、実際に何を実行しているのかは不明ですが、確かなのは、引数として指定した globalPtr を、現在の Port3DPtr、つまり thePort3D として設定することです。そのため、このプロシージャを呼び出す際には、必ず @thePort3D を指定することになっています。このプロシージャは、QuickDraw を `InitGraf` によって初期化した直後に呼び出すのがよいでしょう。

PROCEDURE OpenPort3D(port:Port3DPtr);

InitGraf3D で指定した thePort3D が指す Port3D レコードの中身を、すべてデフォルト値に初期化します。通常は、InitGraf3D を呼び出した直後に、この OpenPort3D を呼び出すことになるでしょう。

PROCEDURE SetPort3D(port: Port3DPtr);

引数の port で指定した Port3DPtr を、現在の Port3D として設定します。この後には、さらに QuickDraw の `SetPort` を呼んで、その Port3D に付随する GrafPtr を、現在の GrafPort として設定します。これは、複数の Port3D を使う場合に、描画対象の Port3D を切り替える際に使うものと考えられます。

PROCEDURE GetPort3D(var port: Port3DPtr);

現在の Port3D へのポインターを取得します。これは Port3D を切り替える際に、それ以前の描画対象だった Port3D を保存しておく際や、Port3D のレコードのフィールドに直接アクセスしたい場合に使えます。

ペンの移動と直線の描画

PROCEDURE MoveTo2D(x,y:fixed);

ペンを、z 軸上の座標値は維持したまま、引数で指定した座標値、x、y の点に移動します。

PROCEDURE MoveTo3D(x,y,z:fixed);

ペンを、引数で指定した座標値、x、y、z の点に移動します。

PROCEDURE Move2D(dx,dy:fixed);

z 軸上の座標値は維持したまま、引数で指定した変位 dx、dy を、それぞれ現在の座標値、x、y に加えて、ペンを移動します。

PROCEDURE Move3D(dx,dy,dz:fixed);

引数で指定した変位 dx、dy、dz を、それぞれ現在の座標値、x、y、z に加えて、ペンを移動します。

PROCEDURE LineTo2D(x,y:fixed);

現在の点から、z 軸上の座標値は維持したまま、引数で指定した座標値、x、y の点まで直線を描きます。

PROCEDURE LineTo3D(x,y,z:fixed);

現在の点から、引数で指定した座標値、x、y、zの点まで直線を描きます。

PROCEDURE Line2D(dx,dy:fixed);

現在の点から、z軸上の座標値は維持したまま、引数で指定した変位dx、dyを、それぞれ現在の座標値、x、yに加えた点まで直線を描きます。

PROCEDURE Line3D(dx,dy,dz:fixed);

現在の点から、引数で指定した変位dx、dy、dzを、それぞれ現在の座標値、x、y、zに加えた点まで直線を描きます。

視点（カメラ）の設定

PROCEDURE ViewPort(r:rect);

QuickDrawのgrafPortの中の、どの位置に3Dグラフィックを表示するかを、引数の長方形で指定します。通常はウィンドウ全体（portRect）を指定すればよいでしょう。

PROCEDURE LookAt(left,top,right,bottom:fixed);

Graf3D空間の中の、どの部分を2D画面に表示するのかを、left、top、right、bottomという4つの座標値で構成する長方形で指定します。それらの型はFixedなので、整数値で座標を指定したければ、通常は`FixRatio`を使って、Fixedに変換してから設定します。ここで指定した範囲が、上の`ViewPort`で指定した2Dの長方形にマッピングされることになります。このプロシージャを呼んだ後は、すぐに`ViewAngle`を呼んで、視線の角度を指定しなければなりません。

PROCEDURE ViewAngle(angle:fixed);

視点から見るパースペクティブを、視野ピラミッドの水平方向の範囲の角度、angle（デグリー）で指定します。指定する値によって、カメラのレンズを交換するような効果が得られるでしょう。角度として0を指定すれば、パースのまったく付かない望遠レンズのような描画となり、25程度が肉眼で見るのに近い値とされています。それより大きい値は、広角レンズのような効果をもたらします。

座標変換

PROCEDURE Identity;

座標変換の行列を、単位行列にリセットします。また、Port3Dレコードのidentフィールドを真にセットします。この状態では、座標変換を施しても、元と同じ座標になります。以下の座標変換に関するプロシージャの説明は、この単位行列に対して単独に適用した場合の話ですが、実際には`Identity`を実行しない限り、座標変換は行列に累積されていきます。

PROCEDURE Scale(xFactor,yFactor,zFactor:fixed);

座標変換の行列を、x、y、z成分それぞれに、xFactor、yFactor、zFactorを掛けた値にスケーリングするようなものに設定します。

PROCEDURE Translate(dx,dy,dz:Fixed);

座標変換の行列を、x、y、z成分それぞれに、dx、dy、dzを加えて平行移動するようなものに設定します。

PROCEDURE Pitch(xAngle:fixed);

座標変換の行列を、x軸についてxAngleだけ回転させるようなものに設定します。x軸の正方向から原点を見たとき、正の値の回転は時計回りとなります。

PROCEDURE Yaw(yAngle:fixed);

座標変換の行列を、y軸についてyAngleだけ回転させるようなものに設定します。y軸の正方向から原点を見たとき、正の値の回転は時計回りとなります。

PROCEDURE Roll(zAngle:fixed);

座標変換の行列を、z軸についてzAngleだけ回転させるようなものに設定します。z軸の正方向から原点を見たとき、正の値の回転は時計回りとなります。

PROCEDURE Skew(zAngle:fixed);

座標変換の行列を、z軸についてzAngleだけスキュー（せん断変形）させるようなものに設定します。その座標変換で変更されるのはx座標の値だけで、y、z軸の値は維持されます。z軸の正方向から原点を見たとき、正の値の回転は時計回りとなります。

PROCEDURE Transform(src:Point3D; VAR dst:Point3D);

座標変換の行列をsrcで与えられたPoint3Dの点に適用し、結果をdstで示されるPoint3Dの点として返します。

3D クリッピング

FUNCTION Clip3D(src1,src2:Point3D; VAR dst1,dst2:Point):Boolean;

引数のsrc1とsrc2で与えられたPoint3Dの2点を結ぶ直線が、視野ピラミッドの範囲に収まるかどうかを真偽値として返します。もしその直線のどの部分も視野ピラミッドに入らない場合には偽を、それ以外なら真を返します。また、参照渡しの引数、dst1とdst2には、クリッピング後のsrc1とsrc2の座標を、画面の2D座標に変換したものが入ります。このClip3Dの動作には、座標変換行列の設定は影響しません。

点の設定

PROCEDURE SetPt2D(VAR pt2D:Point2D; x,y:fixed);

参照渡しの引数で指定したPoint2Dレコード、pt2Dのフィールドを、それぞれx、yで指定した値に設定します。

PROCEDURE SetPt3D(VAR pt3D:Point3D;x,y,z:fixed);

参照渡しの引数で指定したPoint3Dレコード、pt3Dのフィールドを、それぞれx、y、zで指定した値に設定します。

6-3 FixMathの概要

ここでは、念のために、Graf3Dも依拠しているFixMathのインターフェースを、Turbo Pascalのリファレンス・マニュアルから抜粋して掲載しておきます。インターフェースは、すべてファンクションの形式となっています。それらの名前や引数、戻り値の仕様を見れば、機能や使い方は容易に想像でき、誤解の余地はないでしょう。ファンクションの定義の後ろに、inline～と書いてあるものは、それらの実装部分が初代MacintoshのROMに含まれていて、プログラムの実行時にはトラップコードとして実行されるものであることを示しています。そして、externalと書いてあるものは、実装部分は別途ライブラリとして用意されているものを、リンクして使うものと考えればよいでしょう。

これらのファンクションについて、個別の説明は省きますが、やはりここでも1点だけ、この中に登場する変数の型について補足しておきます。それは、ここに挙げる多くのファンクションの引数や戻り値としても指定されている「Fract」です。その定義も、以下のインターフェースに記述されていますが、それはtype Fract = LongInt;というものです。この定義だけでは、形式的には32ビットのLONGINTとして扱うこともできるということで、32ビットの固定小数点型、Fixedと同じです。このFractも、やはり固定小数点型の1つなのですが、FixedがLONGINTの上位16ビットを整数部、下位16ビットを小数部としていたのに対し、Fractは、符号ビットも含めて上位2ビットが整数部、下位30ビットが小数部という、小数部に特化した固定小数点数となっています。これは、SINやCOSなどの三角関数と相性がよいものと想像できるでしょう。LONGINTの値を見ても、それがFractとして実際にどんな値を表すのか、Fixed以上に分かりにくいのですが、やはりFixMathが提供しているFix2FracやFrac2Fixといったファンクションによって、Fixedと相互変換することが可能です。ただし、FractをFixedに変換すると、小数部の精度が30ビットから16ビットに落ちてしまうことは、頭に入れておく必要があります。このFixMathについては、以下にPascalのインターフェース部分を示すに留めます。

```
unit FixMath(-12);

interface

uses MemTypes;

type Fract = LongInt;

{ These routines are only available on a system with a 128K ROM: }

function Long2Fix(x:longint):Fixed;     inline $A83F;
function Fix2Long(x:fixed):LongInt;     inline $A840;
function Fix2Frac(x:fixed):Fract;       inline $A841;
function Frac2Fix(x:fract):Fixed;       inline $A842;

{ Functions to convert between fixed-point types }

function Fix2X(x:fixed):Extended;       inline $A843;
function X2Fix(x:extended):Fixed;       inline $A844;
function Frac2X(x:fract):Extended;      inline $A845;
function X2Frac(x:extended):Fract;      inline $A846;

{ Functions to convert between fixed, fract, and the extended }
{ floating-point type }

function FixAtan2(x,y:LongInt):Fixed;   inline $A818;

{ The following routines are supplied as glue code:

function FracMul(x, y: Fract): Fract;   external;
function FixDiv(x, y: Fixed): Fixed;    external;
function FracDiv(x, y: Fract): Fract;   external;
function FracSqrt(x: Fract): Fract;     external;
function FracCos(x: Fixed): Fract;      external;
function FracSin(x: Fixed): Fract;      external;
```

6-4 Graf3Dのサンプルプログラム試作

MacTechサンプルのリソースを復元して動作

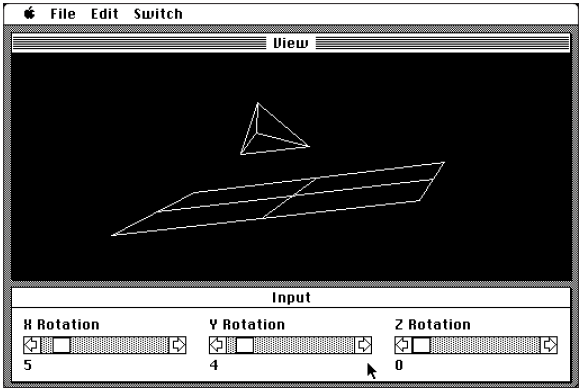
Graf3Dが固定小数点化されて、サードパーティ製の開発環境でも利用できるようになった当時でも、その解説やサンプルプログラムが、それほど潤沢に出回っていたことはなかったでしょう。それから何十年も経過して、本書の執筆時点（2025～2026年初頭）まで残っていて、比較的容易にアクセスできるものとしては、元はApple純正のサンプルプログラム３種と、MacTech誌の記事のウェブ上のアーカイブくらいのものにとどまっています。MacTech誌は、初代Macintoshの発売から少し遅れて誕生した、主にMacintosh用アプリケーションのプログラミング技術を扱った専門誌です。今でもその多くの記事をウェブ上で読むことができます。

「MacTech Graf3D」といったキーワードで検索すると「Understanding Graf3D」（Volume 3 / Issue 3）という記事を読むことができるでしょう。これが未来永劫存続するか分かりませんが、URLを示しておくと、「http://preserve.mactech.com/articles/mactech/Vol.03/03.03/UnderstandingGraf3D/index.html」となっています。

この記事には、当時のMacintosh用のPascal開発環境として、Turbo Pascalと人気を二分していたLightspeed Pascal用のプログラムが掲載されています。ただし、Turbo Pascal用のUNITのインポート用の記述も、コメントとして記されているので、少ない変更でTurbo Pascalでも動かせるようになっています。とはいえ、困ったことに、このプログラムで使用するリソースの中身の定義については、どこにも記述がありません。おそらく当時は、リンク可能なリソースファイルが、付録のフロッピーディスクの中に含まれていたのでしょう。また、本来の記事には配置されていたはずの画像ファイルは、現時点ではリンク切れとなっていて表示されないので、どのような画面を表示するプログラムであったかさえ、視覚的に確認することができなくなっています。

そこで、掲載されているプログラムのソースコードから逆算して、必要なリソースを復元して定義し、アプリケーションとして動かしてみることにしました。２つのウィンドウそのものや、１つのウィンドウの中のコントロール（スクロールバーが３つ）の正確な配置は分かりませんが、かなり近いものにはなっていると考えています。

まずは、プログラムの動作画面を見てみましょう。



MacTech誌Volume 3/Issue 3に掲載されたGraf3Dのサンプルプログラムを復元した画面

このプログラムで描画するのは、平面を表すグリッドと、その上に浮かぶ正四面体のワイヤーフレーム表現だけです。視点の設定によって、プログラム起動時の平面は、1本の横線にしか見えないでしょう。そのような3Dグラフィックを表示するのが「View」ウィンドウですが、その下にもう1つ、「Input」ウィンドウがあります。その中にはX、Y、Zの各軸について、オブジェクト、つまり正四面体の回転量を指定する「X Rotation」、「Y Rotation」、「Z Rotation」という3本のスクロールバーが配置されています。スクロールバーのツマミをドラッグして動かすと、それに応じてViewウィンドウの中の表示が変化し、正四面体が回転します。これは、起動直後の、オブジェクトを回転させるモードの動作です。

ここで「Switch」メニューから「Rotate Space」を選ぶと、今度は空間を回転させるモードとなります。このとき、下のInputウィンドウの3本のスクロールバーのラベルは「Pitch」、「Yaw」、「Roll」という3つに変わり、スクロールバーを動かすと、正四面体の下の平面を含めて、空間全体が回転するようになります。

比較的シンプルなプログラムですが、これによって3Dグラフィックの基礎と、Graf3Dの機能が把握できる優れたサンプルプログラムだと思います。そこで、同じようなものを自分で新たに作るより、これを紹介する方が適切だと考え、リソースを復元して動かしてみたというわけです。自らの環境で、このプログラムを動かしてみたいという人は、以下のリソースの定義ファイルを、RMakerを使ってコンパイルして、MacTechのウェブサイトにあるプログラムにリンクしてください。本書の付録のソースコードの中にも、リソースの定義ファイル「MacTech3D.R」に加え、それをコンパイルした「MacTech3D.Rsrc」ファイルを含めて、ダウン

ロードできるようにしてあります。それをそのままリンクして使っていただければ、RMakerによる処理は不要です。このリソースファイルを、Pascalのソースコードと同じフォルダーに入れ、プログラムの先頭部分に{$R MacTech3D.Rsrc}のような記述を入れておけば、自動的にリンクして、プログラムを動かせるはずです。

```
MacTech3D.Rsrc

Type MENU
,128
\14
  About Me...
  (-

  ,129
File
  Quit/Q

  ,130
Edit
  Undo/Z
  (-
  Cut/X
  Copy/C
  Paste/V
  Clear

  ,131
Switch
  Rotate Space

Type WIND
  ,32000
View
45 8 245 504
Visible NoGoAway
0
0

  ,32001
Input
271 8 336 504
Visible NoGoAway
0
0

Type CNTL
,128
horizontal scroll bar
25 10 41 155
Visible
16
0
0 50 0

,129
horizontal scroll bar
```

```
25 175 41 320
Visible
16
0
0 50 0

,130
horizontal scroll bar
25 340 41 485
Visible
16
0
0 50 0

Type ALRT
  ,128
100 100 190 400
129
CCCC

Type DITL
  ,129
2

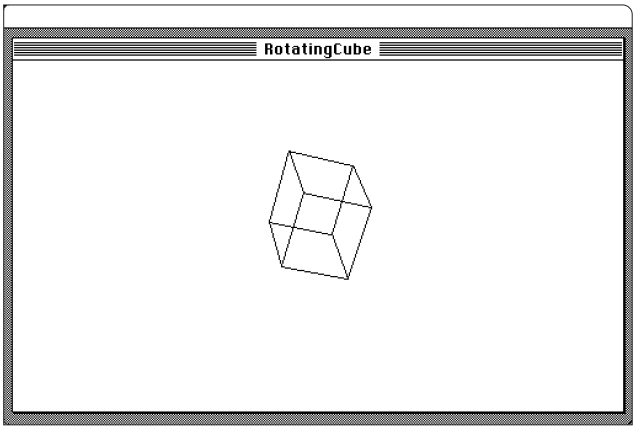
StatText
25 20 36 300
This is a Graf3D Test Program.

BtnItem
60 230 80 290
OK
```

キューブ連続回転プログラムの試作

最後に、Graf3Dを使ったシンプルなオリジナルのサンプルプログラムを示すことにします。Graf3Dの使い方は、その初期化の手順も含めて、やや分かりにくい部分がありますし、これまでに見たApple純正から移植したものや、MacTechに掲載されたものも、凝った部分に覆い隠されて、エッセンスの部分が見えにくくなっているというきらいがあります。そこでここでは、Graf3Dならではの機能を利用しつつ、とにかくできるだけ短いプログラムにする、ということを心がけました。また、Turbo Pascalの実行環境のデフォルトのウィンドウをそのまま使うようにしたので、初期のMacintoshの標準的なアプリケーションの初期化手順は踏んでいませんが、それによって、かえってGraf3D特有の初期化部分が見えやすくなっていると考えています。

プログラムを起動すると、ウィンドウの中で小さいキューブ（正六面体）が回転し続け、マウスボタンを押すと終了するというだけのものです。まずは、その動作画面を示します。



❶できるだけ短いプログラムでGraf3Dならではの機能を実現することを目指したRotatingCubeの動作画面

この画面からも分かるように、キューブを構成する点の数は８個ですが、それを描くための直線は、全部で12本必要となります。一般的な手法のように、「辺」を定義して１本ずつ直線を描くようにすると、始点と終点で合計24の点の3D→2D変換が必要となり、初代のMacintoshにとっては、それだけでも大きな負荷となります。そこで、このプログラムでは、12本の辺を、できるだけ一筆書きで描くようにしてみました。最初に８本を一筆書きで描き、それだけでは描ききれない４本を後から追加しています。

それも含めて、あまり美しいプログラムとは言えませんが、動作速度については、ここから大幅に速くするのは難しいのではないかと考えています。以下にソースコード全体を示します。複雑なことは何もしていないので、説明は省きます。

```
 1| PROGRAM RotatingCube;
 2|
 3| USES
 4|   MemTypes,QuickDraw,OSIntf,ToolIntf,FixMath,Graf3D;
 5|
 6| VAR
 7|   verts:    ARRAY[0..7] OF Point3D;
 8|   fTicks:   LONGINT;
 9|   my3DPort: Port3D;
10|   myPort:   GrafPtr;
11|   i:        INTEGER;
12|
13| PROCEDURE InitCube;
14| BEGIN
15|   SetPt3D(verts[0],FixRatio(-10,1),FixRatio(10,1),FixRatio(-10,1));
16|   SetPt3D(verts[1],FixRatio(-10,1),FixRatio(-10,1),FixRatio(-10,1));
17|   SetPt3D(verts[2],FixRatio(-10,1),FixRatio(-10,1),FixRatio(10,1));
18|   SetPt3D(verts[3],FixRatio(-10,1),FixRatio(10,1),FixRatio(10,1));
```

```
    SetPt3D(verts[4],FixRatio(10,1),FixRatio(10,1),FixRatio(10,1));
    SetPt3D(verts[5],FixRatio(10,1),FixRatio(-10,1),FixRatio(10,1));
    SetPt3D(verts[6],FixRatio(10,1),FixRatio(-10,1),FixRatio(-10,1));
    SetPt3D(verts[7],FixRatio(10,1),FixRatio(10,1),FixRatio(-10,1));
  END;

  PROCEDURE DrawCube;
  BEGIN
    Moveto3D(verts[0].x,verts[0].y,verts[0].z);
    FOR i := 1 TO 8 DO
      Lineto3D(verts[i MOD 8].x,verts[i MOD 8].y,verts[i MOD 8].z);
    Moveto3D(verts[0].x,verts[0].y,verts[0].z);
    Lineto3D(verts[3].x,verts[3].y,verts[3].z);
    Moveto3D(verts[1].x,verts[1].y,verts[1].z);
    Lineto3D(verts[6].x,verts[6].y,verts[6].z);
    Moveto3D(verts[2].x,verts[2].y,verts[2].z);
    Lineto3D(verts[5].x,verts[5].y,verts[5].z);
    Moveto3D(verts[4].x,verts[4].y,verts[4].z);
    Lineto3D(verts[7].x,verts[7].y,verts[7].z);
  END;

  BEGIN
    InitGrf3D(@thePort3D);
    Open3DPort(@my3DPort);
    SetPort3D(@my3DPort);
    GetPort(myPort);
    ViewPort(myPort^.portBits.bounds);

    LookAt(FixRatio(-90,1),FixRatio(50,1),
           FixRatio(90,1),FixRatio(-50,1));
    ViewAngle(FixRatio(40, 1));

    InitCube;

    Pitch(FixRatio(1,1));
    Yaw(FixRatio(3,1));
    Roll(FixRatio(5,1));

    HideCursor;

    REPEAT
      FOR i := 0 TO 7 DO
        Transform(verts[i],verts[i]);
      EraseRect(myPort^.portRect);
      DrawCube;
      Delay(20,fTicks);
    UNTIL Button;
  END.
```