ＣＧ・画像処理の実習・実験用ＡＰＩの開発
—トップダウン的な学習方法の必要性と問題点—

中　村　庸　郎*

Development of the Experimental API for Computer Graphics and Image Processing
− Use and Problem of Top-down Learning Method −

Tsuneo NAKAMURA

要　旨

情報工学科で開講している「画像・図形処理」と「情報工学実験」の数テーマでは，ＣＧや画像処理を計算機実習により学習させているが，何れも高学年の科目であるので既存のソフトウエアを”使う”だけではなく”作る”ことができる能力を身に付けさせねばならない。その手段として，ＣＧ・画像処理アプリケーションを簡単に作成するためのＡＰＩを開発し，ソースコードを参照できる形で使用させることにした。この仕組みにより，ＡＰＩの仕様だけ分かれば取り敢えず”使う”ことができ，ＡＰＩを記述しているプログラミング言語が理解できれば教科書に載っている理論がどのようにプログラム化されているかが分かるので”作る”ことができる能力が身に付く（はずである）。

このＡＰＩは階層構造となっているうえ，どの階層の関数（モジュール）もアプリケーションから呼び出せることを利用し，トップダウン的な学習の教材として利用させる。つまり，上位モジュールの機能をまず体験した後にそのソースコードを参照することにより，下位モジュールでベクトルや行列の計算など低学年で習った理論がプログラム化されている様子が分かる。さらに，下位モジュールをアプリケーションから呼び出してＡＰＩに盛込まれていないアルゴリズムを実現することもできる。

高学年のソフトウエア系科目では，トップダウン的な学習方法が現状では有効と思われるが問題点もある。また，ソフトウエア系科目全体を通した学習方法についても，主観的ではあるが検討してみた。

1．はじめに

一般に，計算機を使って問題を解決するとき，必要なハードウエアやソフトウエアを購入する場合と自作する場合がある。問題解決者の立場は，前者では”ユーザ”であり，後者では”技術者”である。高専教育では，”技術者”の立場で問題解決を図れる能力を養う必要があり，その基礎として「理論を理解し，それを実現する」という学習の積み重ねを重視している。

さて，高専情報工学科におけるＣＧ（Computer Graphics）や画像処理（Image Processing）の教育を考えてみると，現行カリキュラムの運用であるシラバス上では「画像・図形処理（５年選択）」と「情報工学実験（３～５年必修）」で，ＣＧや画像処理に関する多くの基礎理論[1)～4)]を理解させ，それを実現するためのプログラムを作らせる必要がある。ここで，ＣＧや画像処理に限らず，高学年の専門科目で教える理論は，低学年で習ってきた理論の上位階層に位置するので，ソフトウエア系科目であればプログラムを機能的に見ると階層構造[5)]となることに注意したい。

従来は，点・線分・円など基本図形の描画アルゴリズムから順に実現させていたが，この方法では，低学年からボトムアップ的に学んできている，

[1]ＯＳやエディタの使い方
[2]プログラミング言語の文法
[3]プログラムのデバッグ方法
[4]基本的なデータ構造やアルゴリズム
[5]ベクトルや行列等の数学的知識

のうちで，ひとつでも習熟度の低いものがあると，実現したい理論をなかなか正しくプログラム化できないものである。さらに，５年生については授

＊　助教授　情報工学科

業時数が少ないこともあり，高度な内容に到達することが困難となってしまう。結局のところ，1年間の授業時数ではボトムアップ的に基本的な部分から学生に作らせている暇はない。

そこで，限られた授業時数内で，学生の習熟度に左右されずに多くの内容を教える方法として，トップダウン的な学習方法を採用してみることにした。まず，教えたい最上位階層の理論から，低学年で習った下位階層の理論まで，担当教官がすべて関数（モジュール）としてプログラム化しておき，学生が”手本”として参照できる状態にしておく。授業では，最上位のモジュールから使わせ，下位モジュールについては必要に応じてソースコードを参照させたり，それを呼出すプログラムを作らせたりもできる。

つまり，この学習方法は，

[A]最上位階層の理論を授業時間内にすぐ実現できるので，プログラミングに苦労して投げ出すことを防止できる。

[B]各自の理解度や授業進度に応じて掘り下げながらプログラムを参照することによって，各階層の理論がプログラム化されている様子を理解できる。

[C]各階層のモジュールを自由に呼出して，独自のアルゴリズムを実現できる。

という3つの利点をもち，「理論を理解し，それを実現する」という学習をできることは確かである。

ところで，既存の製品が，

- 必要な機能をすべて備えている。

という条件さえ満たしていれば，それを教材として購入し”ユーザ”の立場で使わせても，[A]の効果は同様に期待できる。しかし，[B]が可能であるためには，

- 内部がブラックボックスではなく，構造が公開されている。
- その構造が，学生にとって理解しやすい形式で記述されている。

という2つの条件を満足しなければならないであろう。さらに，[C]が可能であるためには，

- プログラミング，デバッグ環境が完備されている。

という条件も必要である。

これら全ての条件を満たす製品を見つけることは簡単ではないので，担当教官が自作するのが運用・保守の面から考えても最良と判断した。

一方，”技術者”が問題解決のためのアプリケーションプログラムを作成する際，既に用意されている関数（モジュール）を呼出すプログラムを作成する仕組みが広く使われており，用意されている関数（モジュール）群はAPI（Application Programming Interface）と呼ばれる。この仕組みを使うと”技術者”はAPIの関数をあたかも”ユーザ”の立場で使用して高機能のアプリケーションを簡単に作成できる。

本論文で紹介するＣＧ・画像処理用の関数群もAPIの一種であり，ソースコードをそのまま取り込む形で使用する。現在，APIが利用可能な開発環境は，

(a) UNIX
(b) MS-Windows95／ＮＴ
(c) MS-DOS

であり，何れもＣ言語（ＡＮＳＩ準拠）で記述されている。このプログラミング言語は低学年から教えているものであり，かつAPIのソースコードはできるだけ分かりやすく記述している（つもり）。ただし，(c)については当面の利用予定がないため一部未完成である。

これらの開発環境では，本来ならグラフィックの描画をするためには特有の関数を特有の方法で呼ぶ必要があるが，APIの仕様（関数の名前と呼出し方法）が共通であるため，アプリケーションのソースプログラムをそのまま共通に利用できる。このため，プログラムを移植する際に機種や言語に合わせた改造をする必要がなくなり，かつ本題のアリゴリズムの実現に集中できるようになる。

APIとして用意した関数は，描画（2次元，3次元），ベクトル・行列の操作，ファイル操作(画像，サウンド，CAD）に分類できる。さらに，各種設定のスイッチの役目をする”大域変数”，色やＯＮ／ＯＦＦを表す”マクロ名”，ベクトルや行列などを表す”型名”が利用可能である。

2．実習・実験用ＡＰＩを使ったＣＧ・画像処理アプリケーション

ＣＧ・画像処理アプリケーションを作成するためには，ＡＰＩをソーズコードの状態で取り込んでからコンパイル，リンクを行う（**図1**）ため，最新のＡＰＩのソースコードを所定の場所に置いておけばよく，保守が楽である。この点は，UNIX，MS-Windows95／NT，MS-DOSの各開発環境に共通である。ただし，MS-Windows95／NTの場合，ディスプレイのグラフィックス表示のた

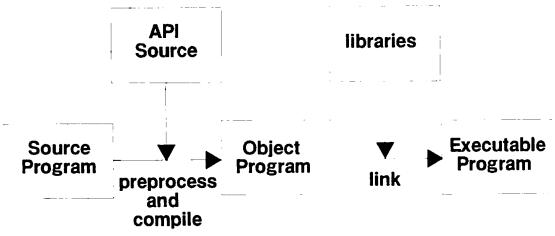


図１　ＣＧ・画像処理アプリケーションの作成

めにライブラリ OpenGL[6)7)]がリンクされる。

ただし，ＣＧアプリケーションを実行したときの描画結果である画像は，UNIX ではハードディスク上に静止画像ファイルとして出力され，MS-Windows95／NT，MS-DOS ではディスプレイ上に直接表示される。（図２）。

したがって，UNIX 上では，”静止画のＣＧ”と”ファイル入出力による画像処理”を，MS-Windows95／NT，MS-DOS 上では”動画のＣＧ”と”リアルタイムの画像処理”を学習することができる。

図２(a)における静止画像ファイルは MS-Windows のビットマップ形式（拡張子 bmp）であり，実習・実験用ＰＣの MS-Windows95／NT 上で即座に表示できる。この UNIX 上における描画は，現行カリキュラムで教えているプログラム開発方法のみで実行できる点，およびファイル出力という基本的な処理で実現されている点から，現時点で最も多用している。

図２(b)におけるディスプレイ表示の際は，MS-Windows95／NT では OpenGL のダイナミックリンクライブラリ，MS-DOS では常駐されたグラフィックドライバがそれぞれ使用される。特に，MS-Windows95／NT 上での描画は，動きを表現できる点，およびマウスやキーボードの入力に対応できる点で，実習・実験への効果的利用が可能である。MS-Windows95／NT 用の API としては，”クオリティー重視版”と”速度重視版”の２種類を用意している。前者は OpenGL[6)7)]の点の描画機能のみを使っており，低速で動画向きではないが，描画アルゴリズムの学習には向いている。後者は，OpenGL の描画機能を多用し，画質は多少犠牲となるが高速であるため動画の表示に向いている。

上述の通り，最終的な描画手段は開発環境に依存したものではあるが，それは API 内部に記述されており，アプリケーション作成時には全く意識する必要がない。

3．API の仕様

API の仕様として，主な関数の概要のみ述べ，大域変数，マクロ名，型名の詳細には触れないことにする。

3.1 描画用関数

２次元，３次元のＣＧを描画するために呼出す関数である。以下に，一部の関数のみについて機能と書式を概説する。

(1) グラフィック機能の初期化

［書式］ void gInit （int argc, char **argv）

２次元または３次元のＣＧを描画するアプリケーションでは，必ず最初に呼出す。argc, argv はメイン関数への引数である。

(2) グラフィック機能の終了

［書式］ void gEnd （int mode）

２次元または３次元のＣＧを描画するアプリケーションでは，必ず最後に呼出す。mode は UNIX でのみ有効であり，出力される画像ファイルにおけるピクセルあたりのビット数を指定する。

(3) ウインドウの設定

［書式］ void gWindow （float xs, float ys,
float xe, float ye）

２次元ＣＧで，観測したい矩形領域の対角座標(xs,ys)と(xe,ye)を指定する。

(4) ビューポートの設定

［書式］ void gView (int xs, int ys,
int xe, int ye)

(a)ＵＮＩＸ　　(b)ＭＳ－Windows95／ＮＴ，ＭＳ－ＤＯＳ

図２　ＣＧアプリケーションの実行

2次元ＣＧで，ディスプレイ上で表示したい矩形領域の対角座標(xs,ys)と(xe,ye)を指定する。指定する座標はピクセルの位置を示すものであり，ディスプレイの左下隅ピクセルが(0,0)である。UNIXでは，矩形領域のサイズのみが有効で，画像サイズとなる。(3)と(4)の設定の組合せにより，任意の観測領域を任意の縦横比で表示できる。

(5) ビューポートの自動設定

［書式］ void gViewAuto (void)

2次元ＣＧで，観測された矩形領域の縦横比を保存してディスプレイ上に表示する。表示される矩形領域のサイズはデフォルト値（関数 gImageSize で変更可）が使用され，位置はディスプレイの中央（UNIX以外）となる。

(6) 画像サイズの変更

［書式］ void gImageSize (int sx, int sy)

関数 gViewAnto 使用時に表示される矩形領域のサイズを幅 sx・高さ sy に変更する。3次元ＣＧでも gViewAuto が内部で呼ばれるので，2次元，3次元ＣＧに共通である。

(7) ビューポートの回転

［書式］ void gVRotate (float angle)

ビューポートを angle［度］だけ反時計方向に回転する。3次元ＣＧでも gViewAuto が内部で呼ばれるので，2次元，3次元ＣＧに共通である。

(8) 視点と目標点の設定

［書式］ void gEye3d (float ex, float ey, float ez,
float ox, float oy, float oz)

3次元ＣＧで，視点(ex, ey, ez)から目標点(ox, oy, oz)の方向を観察することにする。

(9) スクリーンの設定

［書式］ void gScreen3d (float sx, float sy,
float d)

3次元ＣＧで，スクリーンのサイズを幅 sx・高さ sy とし，視点から距離 d の位置に，視線方向と垂直に置くことにする。

(10) モデル座標系の平行移動

［書式］ void gTranslate3d (float x, float y,
float z)

3次元ＣＧで，モデル座標系を x 方向に x，y 方向に y，z 方向に z だけ平行移動する。

(11) モデル座標系の x 軸まわり回転

［書式］ void gRotateX3d (float angle)

3次元ＣＧで，モデル座標系を x 軸まわりに angle［度］だけ回転する（x 軸正方向に右ネジを進ませるときの回転方向）。y，z 軸まわりの回転については省略。

(12) モデル座標系の3次元アフィン変換

［書式］ void gAffine3d (float ax, float bx,
float cx, float dx, float ay, float by,
float cy, float dy, float az, float bz,
float cz, float dz)

3次元ＣＧで，指定した係数でモデル座標系にアフィン変換を行う。

(13) モデル座標系の変換行列の待避

［書式］ void gBlock3dStart (void)

3次元ＣＧで，モデル座標系の変換行列をスタックに待避する。

(14) モデル座標系の変換行列の復帰

［書式］ void gBlock3dEnd (void)

3次元ＣＧで，モデル座標系の変換行列をスタックから復帰する。

(15) 描画色の設定

［書式］ void gColor (int r, int g, int b)

2次元ＣＧの描画色と3次元ＣＧの拡散反射係数を設定する。r, g, b にはそれぞれ赤，緑，青の強さを0～255の範囲で与える。色の指定には，マクロ名 gM_BLACK ～ gM_WHITE を使用することもできる。

(16) 背景色の設定

［書式］ void gColorBg (int r, int g, int b)

2次元，3次元ＣＧの背景色を設定する。

(17) ウインドウ枠の描画

［書式］ void gFrame (int r, int g, int b)

2次元，3次元ＣＧで，ウインドウの枠を指定された色で描画する。

(18) 環境色の設定

［書式］ void gColorAmb (int r, int g, int b)

3次元ＣＧの環境光の色を設定する。

(19) 光源色の設定

［書式］ void gColorLightf (float r, float g, float b)

3次元ＣＧで，光源の色を設定する。r, g, b にはそれぞれ赤，緑，青の強さを0～1の範囲で与える。

(20) 光源の位置の設定

［書式］ void gPutLight (float x, float y, float z)

3次元ＣＧで，光源の位置を(x, y, z)とする。大域変数 gV_SwPointLight がＯＮならば点光源の位置となり，ＯＦＦならば平行光源の向きとなる。

(21) テクスチャ画像の指定

［書式］ void gTex2dImage (char *image)

3次元ＣＧで，テクスチャとして貼り付ける画像ファイル名 image を指定する。

(22) ソリッドテクスチャ関数の指定

［書式］ void gFuncTex3d
(gT _COLORf (＊ f) (gT _VEC3d))

3次元ＣＧで，ソリッドテクスチャリングに使用する関数名 f を指定する。関数 f は3次元座標を入力とし色を出力する関数である。

(23) ペンを置く（2次元）

［書式］ void gMove2d (float x, float y)

2次元ＣＧで，ペンを(x, y)の位置に置く。

(24) ペンで線分を描く（2次元）

［書式］ void gDraw2d (float x, float y)

2次元ＣＧで，ペンで(x, y)の位置まで線分を描く。

(25) 点を描く（2次元）

書式］ void gPlot2d (float x, float y)

2次元ＣＧで，(x, y)の位置に点を描く。

(26) ペンを置く（3次元）

［書式］ void gMove3d (float x, float y, float z)

3次元ＣＧで，ペンを(x, y, z)の位置に置く。

(27) ペンで線分を描く（3次元）

［書式］ void gDraw3d (foat x, float y, float z)

3次元ＣＧで，ペンで(x, y, z)の位置まで線分を描く。

(28) 点を描く（3次元）

［書式］ void gPlot3d (float x, float y, float z)

3次元ＣＧで，(x, y, z)の位置に点を描く。

(29) 円を描く（2次元）

［書式］ void gCircle (float x, float y, float r)

2次元ＣＧで，中心(x, y)，半径 r の円を描く。

(30) ベジェ曲線を描く（2次元）

［書式］ void gBezier2d (int n, int np,
gT _VEC2d ＊ v)

2次元ＣＧで，v で示される np 個の制御点リストに基づく n 次ベジェ曲線をえがく。

(31) Ｂスプライン曲線を描く（2次元）

［書式］ void gBsplind2d (int n, int np,
gT _VEC2d ＊ v)

2次元ＣＧで，v で示される np 個の制御点リストに基づく n 次Ｂスプライン曲線を描く。

(32) 三角形を描く（3次元）

［書式］ void Triangle3d (float x1, float y1,
float z1, float x2, float y2, float z2,
float x3, float y3, float z3)

3次元ＣＧで，3点(x1, y1, z1)，(x2, y2, z2)，(x3, y3, z3)で構成される三角形を描く。大域変数 gV _SwLine がＯＮならば線画となり，ＯＦＦならば面で描かれる。また，大域変数 gV _SwZbuf （デフォルトはＯＦＦ）がＯＮの時，線画ならば隠線消去され，そうでなければ隠面消去される。

(33) 球を描く（3次元）

［書式］ void gSphere (float r)

3次元ＣＧで，原点を中心とした半径 r の球を描く。

(34) トーラスを描く（3次元）

［書式］ void gTorus (float ri, float ro)

3次元ＣＧで，小径 ri，大径 ro のトーラスを描く。

(35) 直方体を描く（3次元）

［書式］ void gBox (float w, float d, float h)

3次元ＣＧで，幅 w，奥行 d，高さ h の直方体を描く。

(36) 双 n 次ベジェ曲面を描く（3次元）

［書式］ void gBezier3d (int n,
int nu, int nv, gT _VEC3d ＊ v)

3次元ＣＧで，v で示される nu × nv 個の制御点に基づく双 n 次ベジェ曲面を描く。

(37) 双 n 次Ｂスプライン曲面を描く（3次元）

［書式］ void gBSpline3d (int n,
int nu, int nv, gT _VEC3d ＊ v)

3次元ＣＧで，v で示される nu × nv 個の制御点に基づく双 n 次Ｂスプライン曲面を描く。

3.2 ベクトル・行列操作用関数

2次元，3次元のベクトルや3×3，4×4の行列を対象とした計算を行うために呼出す関数であり，描画を行うわけではないので関数 gInit, gEnd の呼出しは不要である。以下に，4×4行列，3次元ベクトルに関する一部の関数のみについて機能と書式を概説する。

(1) 4×4行列の成分の表示

［書式］ void gM44Disp (gT _M44 m)

4×4行列 m の成分を標準出力に出力する。

(2) 4×4行列どうしの積

［書式］ gT _M44 gM44xM44
(gT _M44 m1, gT _M44 m2)

4×4行列 m1の右から m2を掛けた積を返す。

(3) 4×4行列の逆行列

［書式］ gT _M44 gM44Inv (gT _M44 m)

4×4行列 m の逆行列を返す。

(4) 3次元ベクトルの座標変換

［書式］ gT _Vec3d gM44xVec3d
(gT _M44 m, gT _Vec3d v)

4×4行列mの右から3次元ベクトル（列）vを同次座標として掛けた結果を，3次元ベクトルとして返す。

(5) 3次元ベクトルの単位ベクトル化

[書式] gT_Vec3d gVec3dUnit (gT_Vec3d v)

3次元ベクトルvを単位ベクトル化して返す。

(6) 3次元ベクトルどうしの内積

[書式] float gVec3dSProd
(gT_Vec3d v1, gT_Vec3d v2)

3次元ベクトルv1とv2の内積を返す。

(7) 3次元ベクトルどうしの外積

[書式] gT_Vec3d gVec3dVProd
(gT_Vec3d v1, gT_Vec3d v2)

3次元ベクトルv1とv2の外積を返す。

(8) 3次元ベクトルの成分の表示

[書式] void gVec3dDisp (gT_Vec3d v)

3次元ベクトルvの成分を標準出力に出力する。

3.3 ファイル操作用関数

画像・サウンド・CADデータのファイル操作を行うために呼出す関数であり，描画を行うわけではないので関数gInit, gEndの呼出しは不要である。以下に，一部の関数のみについて機能と書式を概説する。

(1) 画像データの読込み

[書式] gT_Color **gReadImage(char *file,
int *x, int *y)

画像ファイルfile（拡張子bmp）から全てのピクセルデータを読込み，それらを格納したバッファへのポインタを返す。x，yには画像サイズが格納される。

(2) 画像ファイルの書込用オープン

[書式] FILE *gWOpenImage (char *file,
int x, int y, int mode)

画像ファイルfile（拡張子bmp）を書込みモードでオープンし，ストリームへのポインタを返す。x，yには画像サイズ，modeにはピクセルあたりのビット数を指定する。

(3) サウンドファイルの読込用オープン

[書式] FILE *gROpenSound (char *file,
long *n, short *bpp, short *ch, long *rate)

サウンドファイルfile（拡張子wav）を読込みモードでオープンし，ストリームへのポインタを返す。nにはポイント数，bppにはポイントあたりのビット数，chにはチャンネル数，rateにはサンプリングレートが格納される。

(4) CADデータ（DXF）の読込用オープン

[書式] FILE *gROpenDXF (char *file)

AUTO CADのCADデータファイル（拡張子dxf）を読込みモードでオープンし，ストリームへのポインタを返す。

4．実習・実験における利用例

「情報工学実験（3年）」の”関数と曲面”は，陽関数形式$z = f(x, y)$で定義される曲面(x，yの範囲は［$-\pi$，π］に固定)の形状をCGで表示させる実験であり，

- 数学で習ってきた知識の復習
- 数式をC言語で記述する練習
- CGの基礎知識の学習

を目的としている。**図3**は，関数

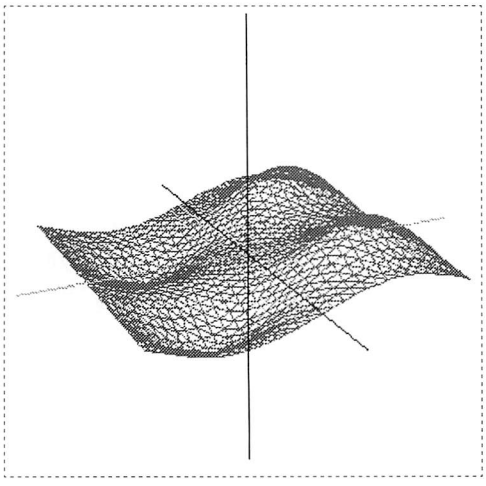
$$z = \frac{\sin y}{e^{\cos^2 x}} \qquad (1)$$

の場合に，隠線／隠面消去の設定を変更した例である。学生が作成する部分は，(1)式のような数式をC言語で記述する関数（**リスト1**）のみであり，上位階層の理論は既に実現されているので”使う”だけである。曲面を定義する関数としては，実験指導書で指定したものと，自由に設計させたものを使う。また，これだけで曲面の形状と各座標軸は表示されるが，さらに隠線／隠面消去，座標変換，視点・視線方向の設定，照明・色の設定を自由に変更させて，形状を把握しやすい設定を見つけさせる。

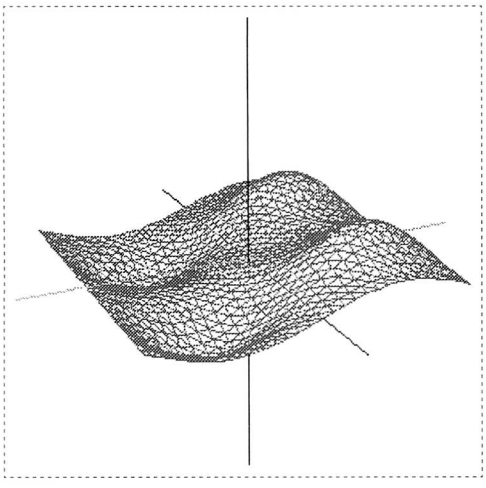
「情報工学実験（4年）」の”画像処理”では，ファイルから読込んだ画像データに基本的な変換やフィルタリングを施し，画像ファイルやディスプレイに出力する。

「画像・図形処理（5年選択）」では，テクスチャマッピング（**図4**，ソリッドテクスチャリング（**図5**），3次元形状のモデリング（**図6**），サウンドファイル（拡張子wav）から読込んだ音の波形表示（**図7**），サウンドファイルの作成・加工，フラクタルやカオスの描画（**図8**）等の実習が可能となっている。

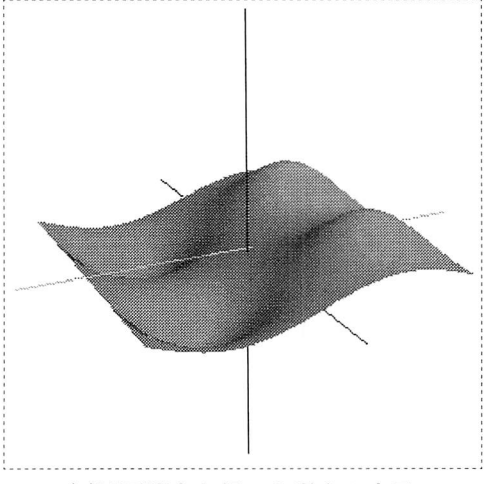
現在のAPIには実習・実験を行う上で最低限必要と考えられる機能のみを盛り込んだが，さらにタートルグラフィックスやボリュームレンダリングのサポートを予定している。また，現在はプログラム上での指定によって形状や動きを記述しているが，3次元計測やCAD，モーションキャプチャといった手段の導入も検討している。



(a)隠線消去を行わない場合の表示



(b)隠線消去を行った場合の表示



(c)隠面消去を行った場合の表示

図３　$z = \dfrac{\sin y}{e^{\cos^2 x}}$　の形状（$-\pi \leq x,\ y \leq \pi$）

リスト１　式(1)をＣ言語で記述した例

```
/* 高さを返す関数 z=f(x,y) */
float f(float x, float y)
{
   return sin(y)/exp(cos(x)*cos(x));
}
```

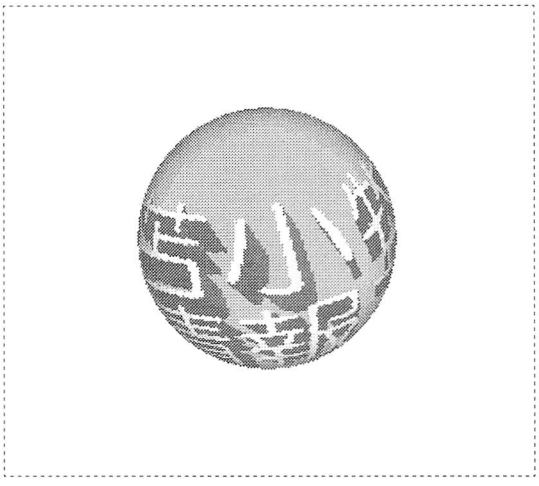


図４　テクスチャマッピングの例

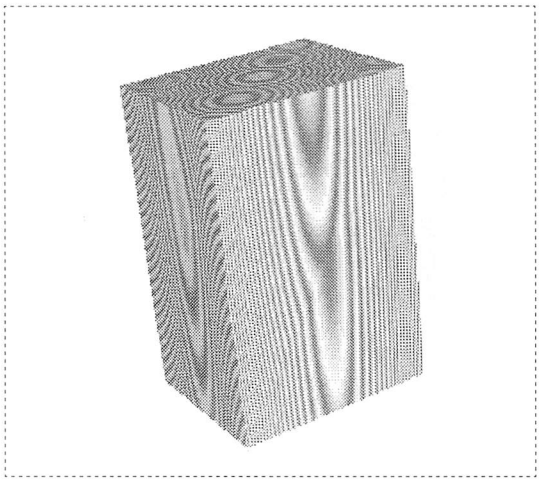


図５　ソリッドテクスチャリングの例

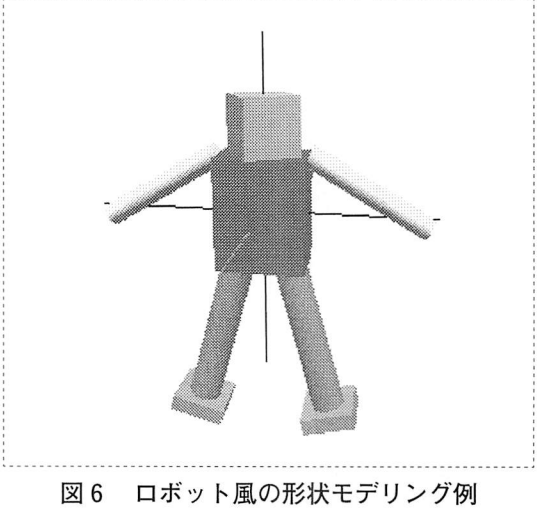


図６　ロボット風の形状モデリング例

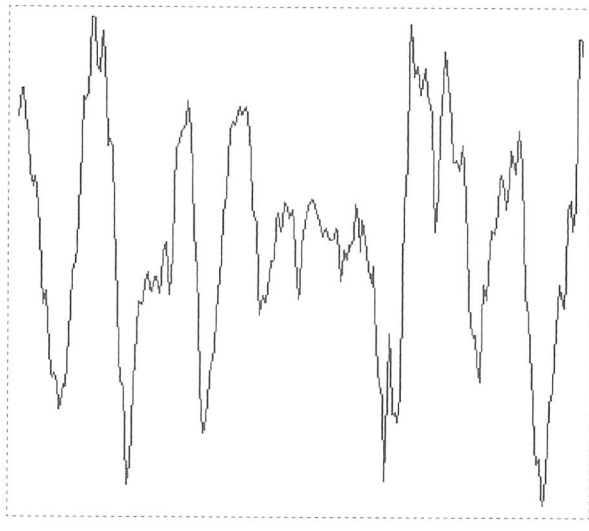


図7　サウンドデータの表示例

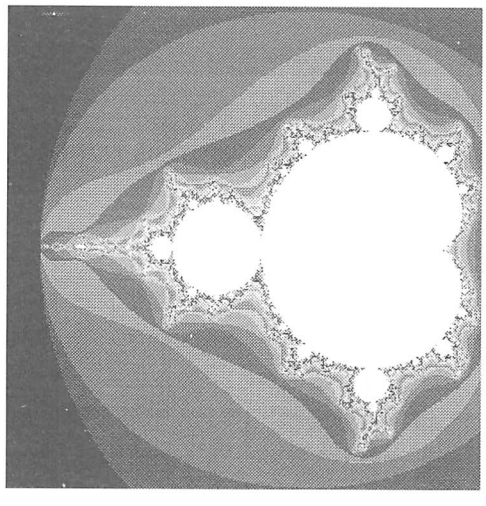


図8　マンデルブロ集合の表示例

5．ソフトウェア系科目の学習方法について

現状では，高学年のソフトウェア系科目でトップダウン的に学習させるメリットが大きいことは前述の通りであるが，次のような問題点もある。

- ●担当教官がいくら努力してわかりやすくＡＰＩのソースコードを記述しても，所詮は他人のプログラムであるから読みにくいであろう。
- ●ＡＰＩを各開発環境に合わせて作成しなければならず，開発・保守のコストが大きい。
- ●ＡＰＩの設計は，科目間（低～高学年の縦方向，高学年どうしの横方向）の連携を念頭において一貫性を持たねばならず，担当教官間の思想の一致が必要である。
- ●担当教官が独自に作り上げたＡＰＩを教材にするため，理論の掘り下げ方によっては試験問題が作りにくい場合がある。

ところで，もともと教官側がプログラムを作って与えねばならない原因は，低学年で習ってきた下位階層の理論が使える形のプログラムとして残っていないことである。したがって，各学年が低学年からボトムアップ的に，再利用可能なモジュール[5]を作成し部品として蓄積していく方法が考えられる。この方法では，高学年科目で必要な基礎理論をすべて低学年でプログラム化させるような授業計画を立てなければならない。

一方，低学年からトップダウン的な学習をさせたとしたらどうであろうか。つまり，”ユーザ”の立場で上位階層の理論をいきなり実現させ，徐々に細かい部分まで掘り下げていく方法である。この方法だと，初めに”情報工学”らしい体験により興味を持たせることができ，”情報工学”らしさがよく分からないうちにリタイヤする学生を少しでも減らせる気はする。この場合にも，やはり先程述べたような問題点が浮上してくるが，低学年では情報リテラシー教育が必要との立場から市販製品を利用する手段も考え得る。

6．おわりに

高学年のソフトウエア系科目におけるトップダウン的な学習方法の教材として，ＣＧ・画像処理の実習・実験用ＡＰＩを開発したのでその概要を報告した。また，トップダウン的な学習方法の問題点を挙げ，さらにソフトウエア系科目の学習方法について主観的な検討を行った。

参考文献

1) L. Ammeraal：「Ｃによるグラフィックス技法」，オーム社（1995）
2) 技術系ＣＧ標準テキストブック編集委員会：「コンピュータグラフィックス」，画像情報教育振興協会（1996）
3) 画像処理標準テキストブック編集委員会：「イメージプロセッシング」，画像情報教育振興協会（1997）
3) 安居院猛，中嶋正之：「コンピュータグラフィックス」，昭晃堂（1992）
4) 河村一樹：「ソフトウェア工学入門」，近代科学社（1996）
6) OpenGL Architecture Board：「OpenGL

Reference Manual, The Official Reference Document for OpenGL, Release 1」, Addison Wesley (1992)

7) OpenGL Architecture Board : 「OpenGL Programming Guide, The Official Guide to Learning OpenGL, Release 1」, Addison Wesley (1993)

(平成10年11月30日受理)