ADVENTURE_TriPatch

Automatic generation of triangular surface patches from IGES data

Version : 1.X

Developer's manual

January 1 , 2003

ADVENTURE Project

目　次

1　はじめに

ADVENTURE システムは，有限要素法解析システムであり CAE ソフトウエアである。有限要素法に基づく解析（メインプロセス）とその前処理（プリプロセス），後処理（ポストプロセス）をユーザーが行うことを支援する。

プリプロセスの大まかなな流れは，以下の通り。

(1) CAD モデルから表面パッチ作成

(2) 表面パッチから４面体要素生成

(3) ４面体要素に解析条件（境界条件，材料情報）を付与して解析モデル生成

ADVENTURE__TriPatch は，前述(1)の機能を提供する。具体的には，CAD から出力される IGES ファイルを読み込み 3 角形の表面パッチを生成する。本書では，この ADVENTURE__TriPatch の仕様概要を説明する。処理概要の説明においては，説明とともに＜ソースヒント＞欄を設けて，その処理をどのソースソースファイルで行っているのかの補足もある。本書を用いて ADVENTURE__TriPatch の機能拡張等に役立ってもらえれば幸いである。

2　AdvTriPatch の概要

ADVENTURE_TriPatch は，ADVENTURE プロジェクトにおいて CAD とのインターフェイスとなるモジュールである。本モジュールは，以下のサブモジュールから構成される。図 2-1 にモジュール概要図を示す。本書では，maskMelon を中心に説明を行う。

(1) maskMelon
　IGES ファイルを読み込み，３角形表面パッチを作成する。

(2) chpatch
　表面パッチのチェックを行う。

(3) mcpach
　表面パッチの向きを揃える。（３角形表面パッチのコネクティビティが形状の外からみて右周りになるようにする）

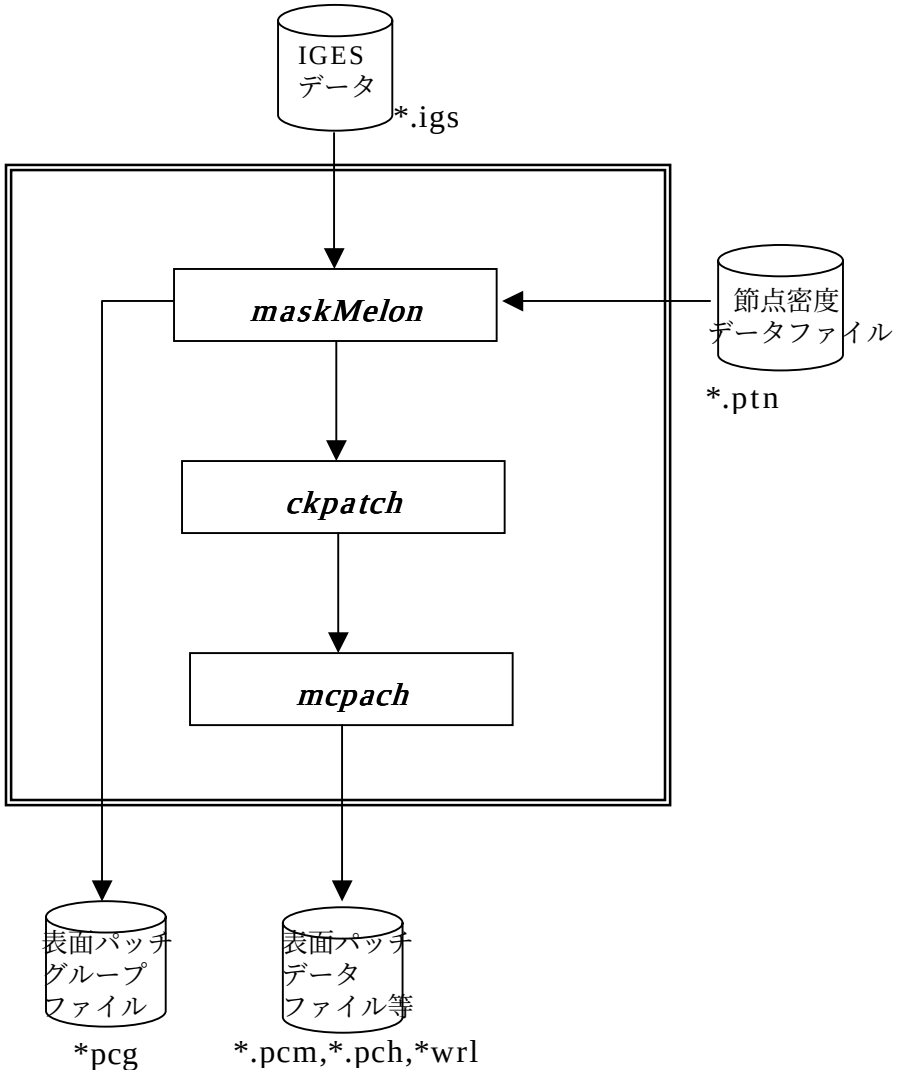


図 2-1　モジュール概要図

3　とりあえず使ってみる

(1)インストールと maskMelon の実行

AdvTriPatch を入手して，とりあえず MaskMelon を使ってみる。以下の手順でインストールすればよい。http://adventure.q.t.u-tokyo.ac.jp/jp/　から AdvTriPatch を入手する。

```
% tar xvfz　AdvTriPatch-x.x.ta.gz　　(x.x はバージョン)
%./configure
%make
%make install
```

$HOME/ADVENTURE/bin の下に maskMelon がインストールされる。

MaskMelon は，簡単なコマンドで動くようになっている。AdvTriPatch に添付の IGES ファイルを利用して，とりあえず動かしてみる。（赤字のコマンドを入力）

```
% maskMelon　--command
%ADV-PRE>
%ADV-PRE>read　adventure_manual_data01.igs
receive　<read>　command
CAD　System is
-----　entity list　of input　-----
  entity　　n　　entity_name(*:skip)
   126　　54　　Rational B-Spline Curve
   128　　8　　Rational B-Spline Surface
～省略～
%ADV-PRE>info　-bb
-- BoundBox info --
x_Min = -7.157656e+01　x_Max = 0.000000e+00
y_Min = -1.614198e+00　y_Max = 4.833711e+01
z_Min = 0.000000e+00　z_Max = 1.000000e+01
%ADV-PRE>density -base 10
receive　<density>　command
baseDistance = 1.000000e+01
%ADV-PRE>patch -density
receive　<patch>　command
created boundary vertex = 51
faceID = 0 / 8　generated patch = 3
faceID = 6 / 8　generated patch = 30
～省略～
%ADV-PRE>write –adventure
receive　<write>　command
writing ... filename = adventure_manual_data01.pch
written　vertex = 76
wrttten　triangular patch = 148
% ADV-PRE>end
```

(2)maskMelon のコマンド解説

前述のコマンドの解説を以下に示す。

以下の起動コマンドを実行することで，コマンド実行待ち状態になる。

% maskMelon　--command

%ADV-PRE>

以下のコマンドは，IGES ファイルを読み込むためのコマンド

%ADV-PRE>read　adventure_manual_data01.igs

バウンディングボックスを表示するためのコマンド

%ADV-PRE>info　-bb

作成したい３角形表面パッチの１辺の稜線の長さを maskMelon にセットするためのコマンド

%ADV-PRE>density -base 10

３角形表面パッチを作成するためのコマンド

%ADV-PRE>patch –density

作成した３角形表面パッチをセーブするためのコマンド

%ADV-PRE>write –adventure

maskMelon**を終了するためのコマンド**

% ADV-PRE>end

MaskMelon は，これらのコマンドを受け付けて処理を行う仕組みとなっている。私自身は，開発の際これらのコマンドを利用してデバックを行っている。これらのコマンドのインプリメントを参照すれば，ソースの理解の助けになることと思う。

またバッチ処理したければ，上記の入力したコマンドをテキストファイル（例えばファイル名 test.dat）に書いて,以下のコマンド実行すればよい。

% maskMelon　--command　< test.dat

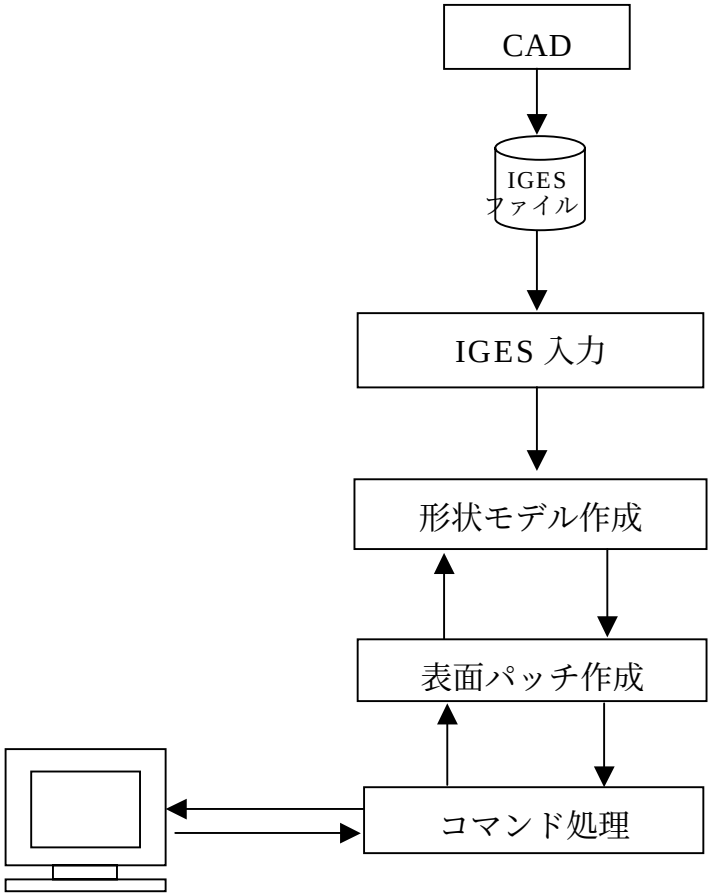
＜ソースヒント＞

・コマンド受付処理

AdvTriPatch-x.x/IGES5.3/patch の下の agk_preCommand.h, agk_preCommand.cpp を参照。

4　内部システム構成

本プログラムのシステム構成図を図 4.1 に示す。



(1) 形状モデル作成
　内部的な形状モデルを作成する。
(2) 表面パッチ作成
　形状モデルにアクセスして，表面パッチを作成する。
(3) コマンド処理
　コマンドを受け付け，コマンドに応じた処理を行う。
(4) IGES 入力
　IGES ファイルから形状モデル変換する。

5　データ構造

5.1　データ表現

maskMelon では、IGES ファイルを読み込み後、立体の境界表現[1][2]として，図 5-1 に示すデータ構造を用いている。詳しくは，末章の参考文献等を参照。

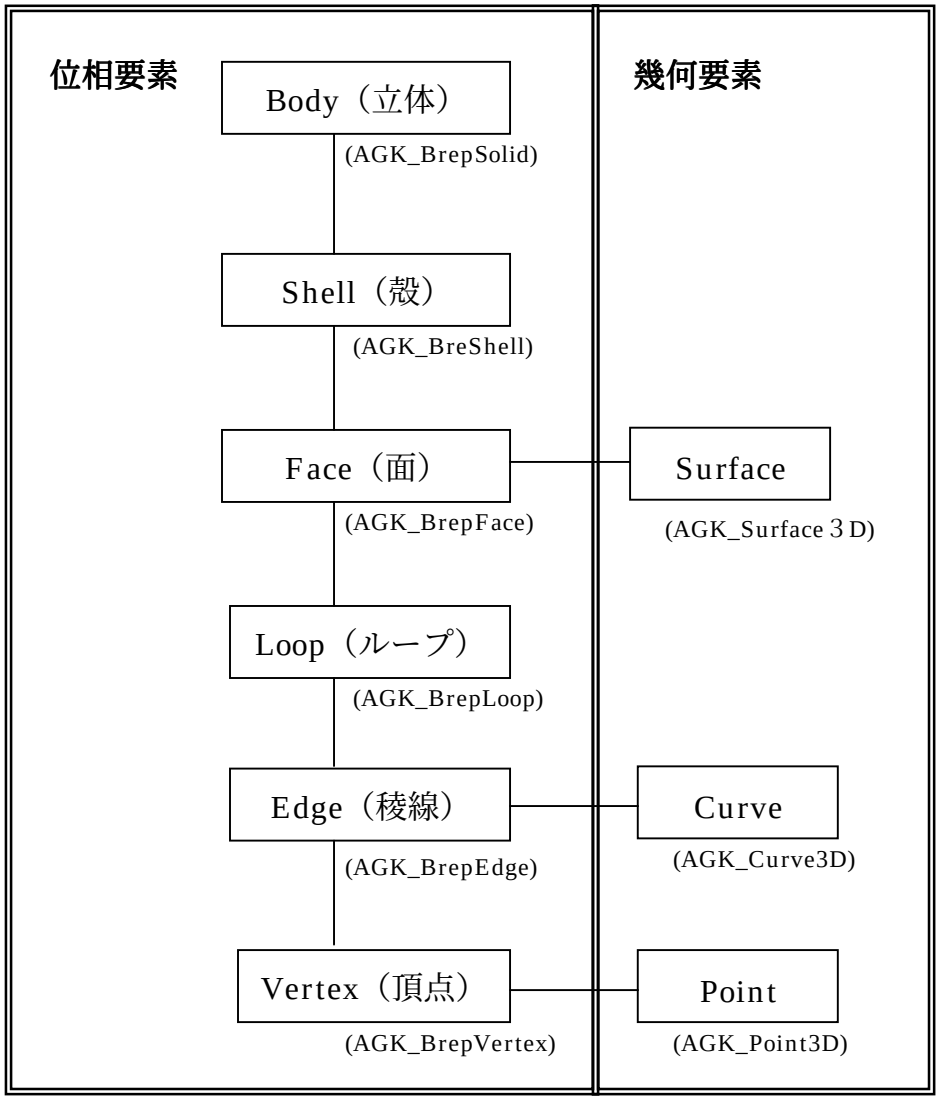


図 5-1 データ構造

( )，maskMelon で実装されているクラス名

＜ソースヒント＞

・データ構造

位相要素は，AdvTriPatch-x.x/IGES5.3/object3D の下の

・agk_BrepObject3D.h(cpp)

幾何要素は，AdvTriPatch-x.x/IGES5.3/geometry/の下の

・ agk_surface3D.h(cpp), agk_curve3D.h(cpp), agk_vector3D.h(cpp),

を参照.

境界表現では，立体を内側と外側に分ける境界面を定義して表現する方法であり，立体をとりまとめる要素として Body があり，Body は，位相要素として Shell,Face,Loop,Edge,Vertex の保持する。図 5-2,図 5-3 にこれらを示す。

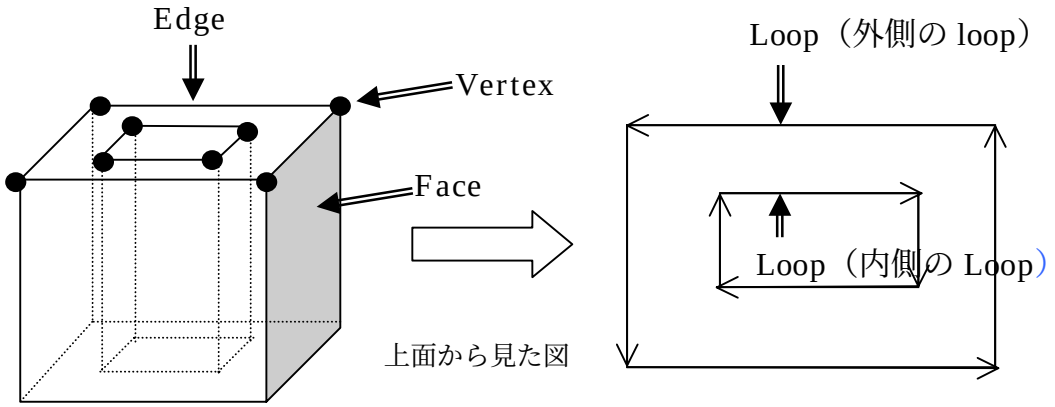


図 5-2　Face,Loop,Edge,Vertex の図

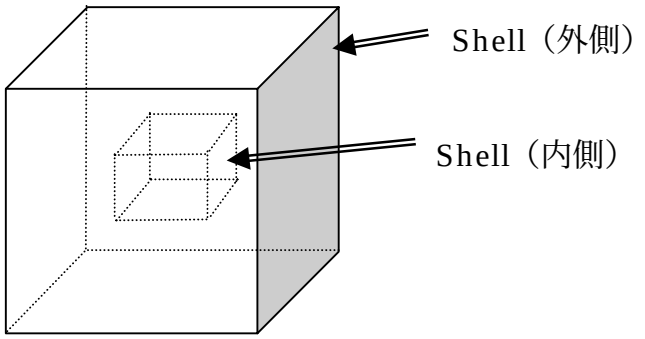


図 5-3　Shell の図（立体内部に空洞がある場合）

幾何要素は, Surface,Curve,Point がある。幾何要素は立体の形を決める要素であり、位相要素の Face,Edge,Vertex にそれぞれ付属する。

Loop の方向は外側 Loop は，形状の外から見て左回り，内側 Loop は，右回りである。面の法線は，形状の内側から外側のループを見た時に，イメージとして右ねじの進む方向である。Surface（曲面）の法線は，曲面のパラメータ空間の u とｖのそれぞれの一階偏導関数の外積の方向である。Face(面)の法線と Surface(曲面)の法線は，一致している必要がある。

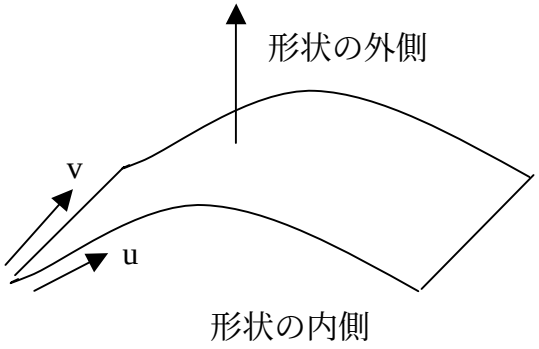


図 5-4　曲面の法線方向

6 外部データ入力

6.1 IGES 入力

MaskMelon は，IGES 仕様（Ver5.3: ASCII フォーマット）のソリッドデータの読み込みが可能である。主な仕様を，以下に示す。

(1) 対応エンティティ

maskMelon で対応しているエンティティを表 6-1 に示す。

表 6-1 対応エンティティ

No	エンティティ番号	エンティティ名
1	100	円弧
2	110	線
3	124	変換マトリックス
4	126	有理化 B スプラインカーブ
5	128	有理化 B スプラインサーフェース
6	186	多様体ソリッド B-Rep オブジェクト
7	502	頂点
8	504	辺
9	508	ループ
10	510	面
11	514	シェル

(2)想定する CAD

CAD から出力される IGES ファイルは､CAD が内臓しているモデリングカーネル（モデルの表現方法）やトランスレータによってさまざまな出力がある。,動作確認は，以下の２つの CAD で行った。

a. I-DEAS MasterSerise 8

b. MicroCADAM V4R2

＜ソースヒント＞

・ **IGES 読み込み**

AdvTriPatch-x.x/IGES5.3/transrator 以下の agk_IGESBrepObject.h(cpp)を参照。

6.2 その他のデータ入力

表面パッチ作成処理では，内部的なデータ表現にアクセスして表面パッチを作成する。内部的なデータを適切に作成すれば，データインポート等の拡張も可能となる。ここでは，内部的なデータ表現をもう少し詳しく見てみる。とりあえず最も簡単な４面体を境界表現で表してみる。図 6.1 のようになる。頂点数は４，稜線数は６，面数は４である。

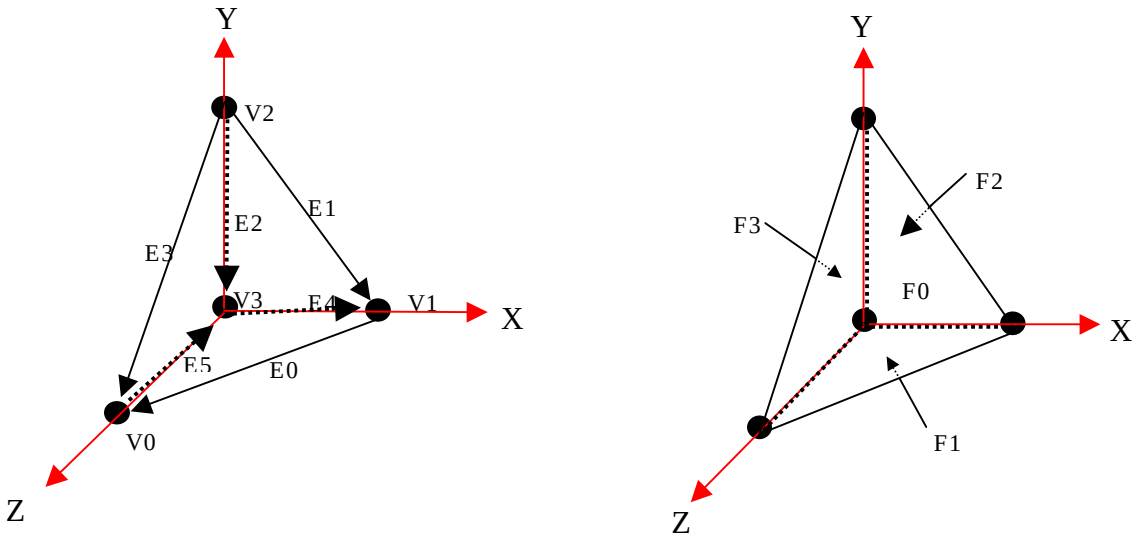


図 6.1 ４面体形状

これを境界表現で表すと図 6.2 のようになる。

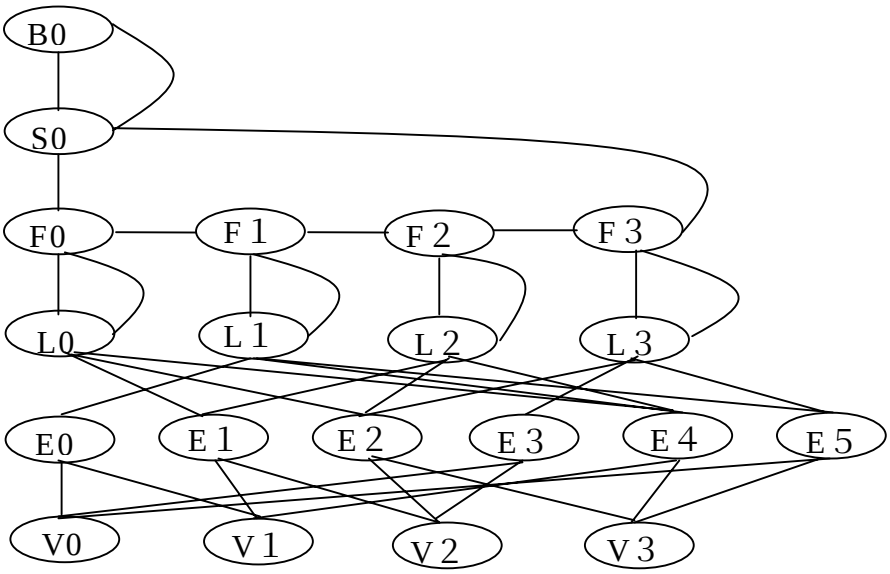


図 6.2 ４面体形状の位相情報

　４面体をmaskMelonの内部データ表現で作成した場合のサンプルソースが以下にある。
AdvTriPatch-x.x/IGESV5.3/patch/exsample1.cpp

AdvTriPatch-x.x/IGESV5.3/patch/ の下で

%make exsamp1
%./exsampl1

とすると”exsample1.pch”のファイルができる。
　AdvTriPatch-x.x/bin/の下のユーティリティshell2vrmlを利用するとVRML形式(Ver1.0)の出力ができる。

%shell2vrml –normal exsample1.pch > exsample1.wrl

　本サンプルソースには，４面体を境界表現で表し，表面パッチを作成するまでの一連のコードが書かれているので参考になると思う。極力コメントも入れておいた。

```
//全てのヘッダー情報が入っているファイル
#include "agk_all_header.h"


//NURBS曲線を作成するためのサンプル関数
//ここでは４面体形状の稜線の部分をＮＵＲＢＳに変換

AGK_Curve3D*    sample_nurbs_line(AGK_Point3D    start_position,
                                  AGK_Point3D    end_position)
{

   //次数
   int    degree = 1;

   //制御点
   AGK_Point3D       control_points[2];
   control_points[0] = start_position;
   control_points[1] = end_position;

   //ウエイト(制御点と同じ数だけ)
   AGK_DoubleArray    weights;
   weights.push_back(1.0);
   weights.push_back(1.0);

   //ノット[ノット数は制御点 + 次数 + 1 ]
   AGK_DoubleArray    knots;
   knots.push_back(0.0);
   knots.push_back(0.0);
   knots.push_back(1.0);
   knots.push_back(1.0);

   //曲線の有効範囲
   double      param[2];
   param[0] = 0.0; //開始パラメータ値
   param[1] = 1.0; //終了パラメータ値

   AGK_NurbsCurve3D*    nurbsc = new AGK_NurbsCurve3D(degree,
                                                      control_points,
                                                      weights,
                                                      knots,
                                                      param);


      return nurbsc;
}
```

```
//空間上の３点を与えて,ＮＵＲＢＳ曲面（平面）を作成する時に必要な４つの制御点を算出する
void     sample_get_plane_control_point(AGK_Point3D     pos0,
                                        AGK_Point3D     pos1,
                                        AGK_Point3D     pos2,
                                        AGK_Point3D*    return_points)

{


   AGK_PlaneEquation3D    plane(pos0,pos1,pos2);

   AGK_Point2D    p0 = plane.parameterAt (pos0);
   AGK_Point2D    p1 = plane.parameterAt (pos1);
   AGK_Point2D    p2 = plane.parameterAt (pos2);
   double x_min    =    min3(p0(0),p1(0),p2(0));
   double x_max    =    max3(p0(0),p1(0),p2(0));
   double y_min    =    min3(p0(1),p1(1),p2(1));
   double y_max    =    max3(p0(1),p1(1),p2(1));
   return_points[0] = plane.positionAt(AGK_Point2D(x_min,y_min)); //left_low
   return_points[1] = plane.positionAt(AGK_Point2D(x_min,y_max)); //left_upper
   return_points[2] = plane.positionAt(AGK_Point2D(x_max,y_min)); //right_low
   return_points[3] = plane.positionAt(AGK_Point2D(x_max,y_max)); //right_upper
}


//３角形をNURBS曲面に変換
AGK_Surface3D*    sample_nurbs_plane(AGK_Point3D     pos0,
                                     AGK_Point3D    pos1,
                                     AGK_Point3D    pos2)


{

   //曲面情報
   int            info[2];
   //[0]:u方向が閉じている場合には1,閉じていない場合には0
   //[1]:v方向が閉じている場合には1,閉じていない場合には0

   info[0] = 0;
   info[1] = 0;

   //次数
   int            degree[2];
   degree[0] = 1; // u方向の次数
   degree[1] = 1; // v方向の次数

   //制御点
    AGK_Point3D            control_points[4]; //(0,0),(0,1),(1,0),(1,1)
   sample_get_plane_control_point(pos0,pos1,pos2,control_points);


   //ウエイト
   AGK_DoubleArray       weights;
   weights.push_back(1.0);
   weights.push_back(1.0);
```

```
    weights.push_back(1.0);
    weights.push_back(1.0);


    //ノット[制御点 + 次数 + 1 ]
    AGK_DoubleArray      uknots;
    AGK_DoubleArray      vknots;
    uknots.push_back(0.0);
    uknots.push_back(0.0);
    uknots.push_back(1.0);
    uknots.push_back(1.0);

    vknots.push_back(0.0);
    vknots.push_back(0.0);
    vknots.push_back(1.0);
    vknots.push_back(1.0);


    //曲面の有効範囲
    double     param[4];
    param[0] = 0.0; //u方向 開始パラメータ値
    param[1] = 1.0; //v方向 終了パラメータ値
    param[2] = 0.0; //v方向 開始パラメータ値
    param[3] = 1.0; //v方向 終了パラメータ値


    AGK_NurbsSurface3D*    nurbss = new AGK_NurbsSurface3D(info,
                                                           degree,
                                                           control_points,
                                                           weights,
                                                           uknots,
                                                           vknots,
                                                           param);

    return nurbss;
}
```

```
int    main()
{
   //ソリッドデータ
   AGK_BrepObject    object;
   //節点密度データ
   AGK_NodeDensity3D            density;
   //節点間隔を1.0とする
   density.setBaseDistance(1.0);


   AGK_TrianglePatchModel3D      surface_patches;
   AGK_SurfacePatchGenerator    meshGenerator;

   //４面体の４節点を定義
   int    vertex_size = 4;

   AGK_Point3D    p0(0.0,0.0,10.0);
   AGK_Point3D    p1(10.0,0.0,0.0);
   AGK_Point3D    p2(0.0,10.0,0.0);
   AGK_Point3D    p3(0.0,0.0,0.0);


   //4点の頂点を作成する.
   //create vertex
   AGK_BrepVertex* v0    =    object.createVertex();
   v0->setPosition3D(p0);
   AGK_BrepVertex* v1    =    object.createVertex();
   v1->setPosition3D(p1);
   AGK_BrepVertex* v2    =    object.createVertex();
   v2->setPosition3D(p2);
   AGK_BrepVertex* v3    =    object.createVertex();
   v3->setPosition3D(p3);

   ////////////////////////////////////////
   //６つの稜線を作成する  //
   ////////////////////////////////////////
   AGK_BrepEdge* e0    =      object.createEdge();
   e0->setStartVertex(v1); e0->setEndVertex(v0);
   //稜線の向きと曲線の進む向きは、一致させること
   AGK_Curve3D*    line0 = sample_nurbs_line(v1->position3DAt(),
                                                v0->position3DAt() );

   e0->setCurve(line0);

   AGK_BrepEdge* e1    =      object.createEdge();
   e1->setStartVertex(v2); e1->setEndVertex(v1);
   AGK_Curve3D* line1 = sample_nurbs_line(v2->position3DAt(),
                                                v1->position3DAt() );
   e1->setCurve(line1);

   AGK_BrepEdge* e2    =      object.createEdge();
   e2->setStartVertex(v2); e2->setEndVertex(v3);
   AGK_Curve3D* line2 = sample_nurbs_line(v2->position3DAt(),
                                                v3->position3DAt() );
```

```
e2->setCurve(line2);


AGK_BrepEdge* e3    =      object.createEdge();
e3->setStartVertex(v2); e3->setEndVertex(v0);
AGK_Curve3D* line3 = sample_nurbs_line(v2->position3DAt(),
                                                    v0->position3DAt() );
e3->setCurve(line3);

AGK_BrepEdge* e4    =      object.createEdge();
e4->setStartVertex(v3); e4->setEndVertex(v1);
AGK_Curve3D* line4 = sample_nurbs_line(v3->position3DAt(),
                                                    v1->position3DAt() );
e4->setCurve(line4);

AGK_BrepEdge* e5    =      object.createEdge();
e5->setStartVertex(v0); e5->setEndVertex(v3);
AGK_Curve3D* line5 = sample_nurbs_line(v0->position3DAt(),
                                                    v3->position3DAt() );
e5->setCurve(line5);

////////////////////////////////////////////
//４つのループを作成する　//
////////////////////////////////////////////
int    orient_true = 1;
int    orient_false = 0;

AGK_BrepLoop* loop0    =      object.createLoop();
//ループの向きとエッジの向きが等しい場合には，orient_true
//ループの向きとエッジの向きが逆の場合には，orient_false

loop0->addEdge(orient_true,e3);
loop0->addEdge(orient_false,e0);
loop0->addEdge(orient_false,e1);

AGK_BrepLoop* loop1    =      object.createLoop();
loop1->addEdge(orient_true,e0);
loop1->addEdge(orient_true,e5);
loop1->addEdge(orient_true,e4);

AGK_BrepLoop* loop2    =      object.createLoop();
loop2->addEdge(orient_true,e1);
loop2->addEdge(orient_false,e4);
loop2->addEdge(orient_false,e2);

AGK_BrepLoop* loop3    =      object.createLoop();
loop3->addEdge(orient_false,e3);
loop3->addEdge(orient_true,e2);
loop3->addEdge(orient_false,e5);




////////////////////////////////////
```

```
//４つの面を作成する　//
//////////////////////////////////////
AGK_BrepFace* f0　=　　object.createFace();
//面にループを追加する時には，１件目の追加は、外側ループ、２件目以降は内側ループである
f0->addLoop(loop0);
AGK_Surface3D* surf0 =　sample_nurbs_plane(p2,p0,p1);
f0->setSurface(surf0);

AGK_BrepFace* f1　=　　object.createFace();
f1->addLoop(loop1);
AGK_Surface3D* surf1 =　sample_nurbs_plane(p1,p0,p3);
f1->setSurface(surf1);


AGK_BrepFace* f2　=　　object.createFace();
f2->addLoop(loop2);
AGK_Surface3D* surf2 =　sample_nurbs_plane(p2,p1,p3);
f2->setSurface(surf2);


AGK_BrepFace* f3　=　　object.createFace();
f3->addLoop(loop3);
AGK_Surface3D* surf3 =　sample_nurbs_plane(p2,p3,p0);
f3->setSurface(surf3);

//////////////////////////////////////////////
//１つのシェルを作成する　//
//////////////////////////////////////////////
AGK_BrepShell* s0　=　　object.createShell();
s0->addFace(f0);
s0->addFace(f1);
s0->addFace(f2);
s0->addFace(f3);

/////////////////////////////////////////
//１つの立体を作成する　//
/////////////////////////////////////////
AGK_BrepSolid* b0　=　　object.createBody();

//外側のShellを追加する.
b0->addOuterShell(s0);

//内側のShellを追加する場合には,
//b0->addInnerShell(s0);

//各要素に識別番号を持たせる
object.renum();
```

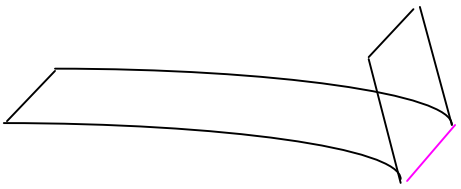
```
    meshGenerator.setSolid(&object,&surface_patches);

    //稜線の分割
    if(meshGenerator.curveDivision(&density)){
       cerr<<"--- err    edge division    ---"<<endl;
       return 1;
    }else{
       cerr<<"created boundary vertex = "<<surface_patches.getVertexSize()<<endl;
    }

    //表面パッチの作成
    //面毎に表面パッチを作成する

    AGK_IntArray    falseFace;
    int numberOfPatch;

    for(int i = 0; i < object.faceSize();i++){
       if(meshGenerator.makeSurfacePatch(i,&density) == false){
          //エラーの場合
           falseFace.push_back(i);
       }else{
          //正常の場合
          numberOfPatch =
                   meshGenerator.getSurfaceGenerator(i)->getTriangleSize();
          cerr<<"faceID = "<<i<<" / "<<object.faceSize();
          cerr<<"    generated patch = "<<numberOfPatch<<endl;
       }
    }

    //表面パッチ作成が問題ないかを確認する
    if(falseFace.size() == 0){
       cerr<<"---    patch generater normally ended ---"<<endl;
    }else{
       cerr<<"---    err    patch generater abnormally    ended ---"<<endl;
       cerr<<" err face size = "<<falseFace.size()<<endl;
       cerr<<"err faceID = ";
       for(int i = 0; i < falseFace.size();i++)
          cerr<<falseFace[i]<<"    ";
       cerr<<endl;
    }

    //表面パッチの頂点と表面パッチをまとめて,リナンバリングする
    meshGenerator.mergeVertexFace();
    surface_patches.renum();
    cout<<"created vertex = "<<surface_patches.getVertexSize()<<endl;
    cout<<"created triangular patch = "<<surface_patches.getFaceSize()<<endl;
    //ファイルに出力する
    ofstream fout;
    AGK_String    fileName("exsample1.pch");
    fout.open(fileName.rep(),ios::out);
    surface_patches.printADVForm(fout);
    fout.close();
    return 0;
}
```

6.3　形状モデルに関する留意点

形状モデルに関する留意点を以下に挙げる。この留意点は，表面パッチがうまく作成できない可能性があるので気をつけておいた方が良い。

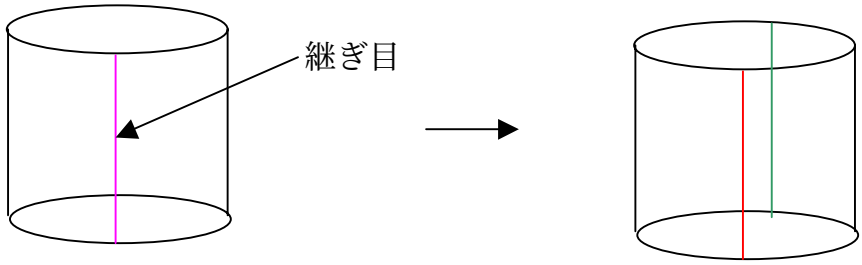
(1) 曲面が折れている

曲面が折れている場合には、折れ部分の表面パッチがうまく作成できない場合がある。



(2) 曲面は閉じていない方がベター

例えば，円柱の側面が１つの曲面（面１つ）で表現されているよりは，２つの曲面（２つの面）の方が良い。（現行でも問題は少ないが推奨する）



7　表面パッチ作成

7.1　概要

maskMelon では、境界表現（B-Reps）のソリッドモデルを入力して３角形表面パッチを生成する。現在の機能の特徴として、以下のものがあげられる。

(1)３角形表面パッチの節点は、必ず入力形状上に存在する。
(2)前記４で説明した位相要素 Face,Edge,Vertex と作成された節点及び表面パッチには関連性がある。（関連情報は、表面パッチグループデータファイルとして出力）
(3)節点の粗密の制御が可能。

(2)の特徴は，逆に表面パッチ作成の制約条件でもあるため入力形状に微小稜線（稜線の長さが非常に短いもの），微小面（面の面積が非常に小さいもの），鋭角を持つ面等が含まれる場合には，低品質の表面パッチを作成することがあるので留意すること。

以下は，簡単な形状であるトーラスの表面パッチを作成した例である。
図 7-1～図 7-3 に表面パッチ，稜線，面毎の塗りつぶし図の例を示す。

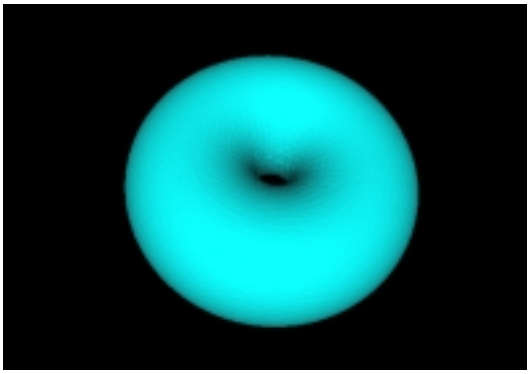


図 7-1　表面パッチ

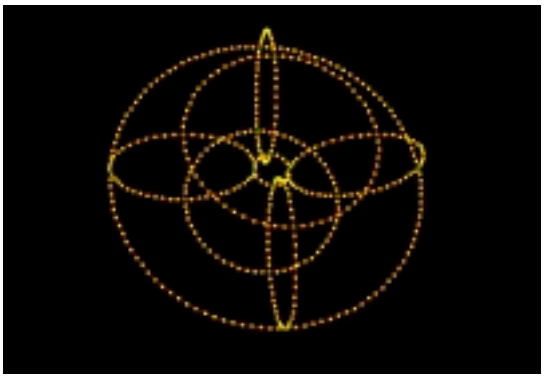


図 7-2　面の境界

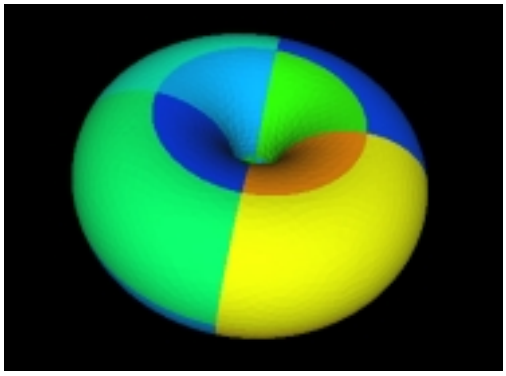


図 7-3　面毎の塗りつぶし

7.2　表面パッチ生成手順

表面パッチ生成の手順を以下に示す。

(1)頂点上に節点生成（図 7-4）
(2)稜線上に節点生成 (図 7-5)
(3)面上に節点生成及び３角形表面パッチ作成(図 7-6)

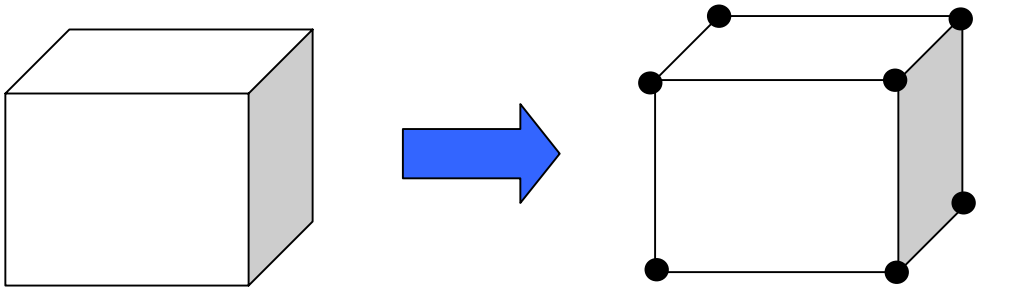


図 7-4　頂点上に節点生成

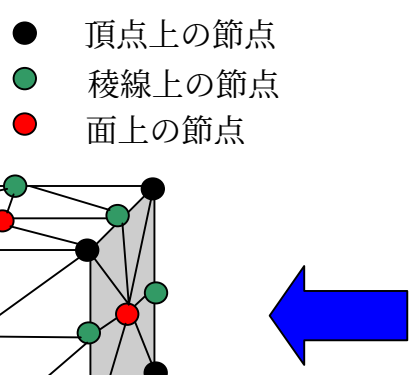


図 7-5　稜線上に節点生成

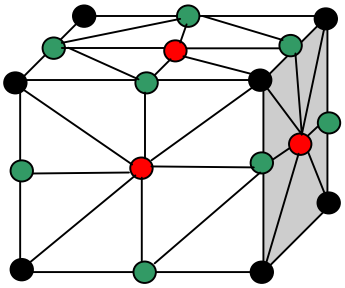


図 7-6　頂点上に節点生成

7.3　**粗密制御**

maskMelon では、節点の粗密制御が可能である。図 7-7 に例を示す。

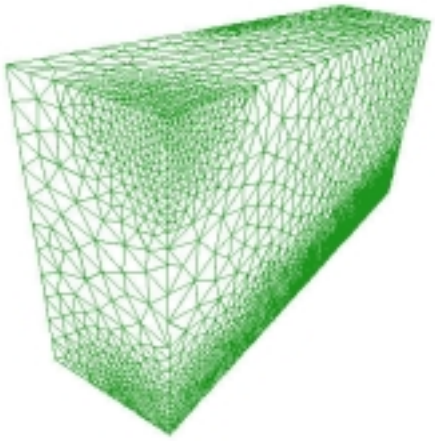


図 7-7　粗密制御例

節点の粗密に関わる項目として，以下のものがある。

(1)基本節点間隔

　表面パッチのグローバルな稜線長

(2)ローカル節点間隔

　形状の一部分の節点数を制御する際の節点間隔

上記 2 つ制御に関わる項目は、重ね合わせが可能でありまた，節点座標が決まれば，節点間隔が一意に決まる。（その際の節点間隔は，ベース節点間隔及びローカル節点間隔の最も小さいものが採用される）

ローカル節点間隔については，数種類のタイプを用意してある。詳細については，後記 10「10.5 節点密度ファイル」を参照。

＜ソースヒント＞

・ **密度制御**

AdvTriPatch-x.x/IGES5.3/density 以下のソースを参照。

7.4　稜線上の節点生成

稜線上の節点生成は，頂点上の節点生成後に行われる。処理手順は，以下のようになる。

1)稜線の長さを求める。
2)基本節点間隔で稜線を初期分割する。図 7-8 を参照。
3)以下の処理を，適切な稜線上の節点間隔となるまで反復して行う。
　分割された稜線の中点（図 7-9 の P1）が以下の条件を満たす場合，中点を節点として生成する。
　P1 の節点間隔　＜　(P1 から P0 の長さ　あるいは　P2 から P1 の長さ)

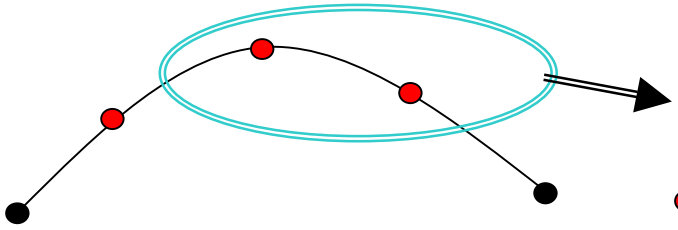


図 7-8　稜線上の初期分割

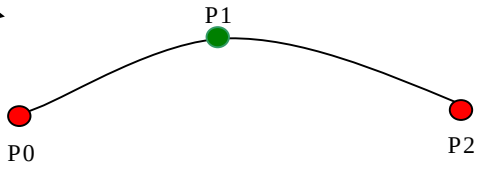


図 7-9　稜線に節点生成

＜ソースヒント＞
・ 稜線上の節点生成
AdvTriPatch-x.x/IGES5.3/triangulation3D の下の
Agk_curve_division.h(cpp)
を参照。

7.5　面上の表面パッチ生成

面上の表面パッチ作成は，図 7-10～図 7-13 の手順で行われる。まず図 7-10 にように稜線と面と曲面の関係がある。白い部分が曲面，赤い部分が稜線，赤い部分で囲まれた部分が面である。曲面は，パラメトリック曲面（NURBS）を取り扱う。

まず，前述した既に節点生成処理が終了した稜線上の節点を曲面上に射影する。図 7-11 のようになる。曲面上のパラメータ値を利用して曲面上で図 7-12 のように初期三角形分割を行う。この三角形内に節点を挿入し表面パッチを生成(図 7-13)する。

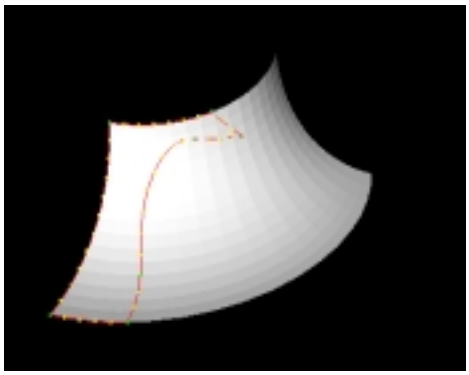


図 7-10　面と曲面

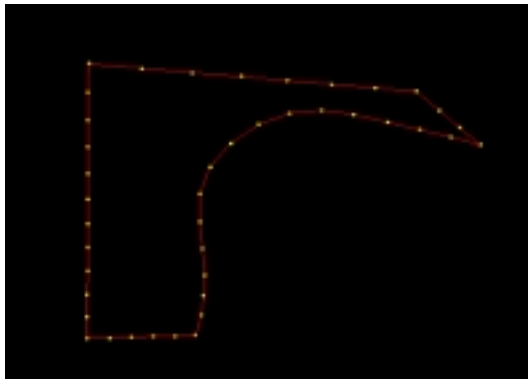


図 7-11　射影された節点

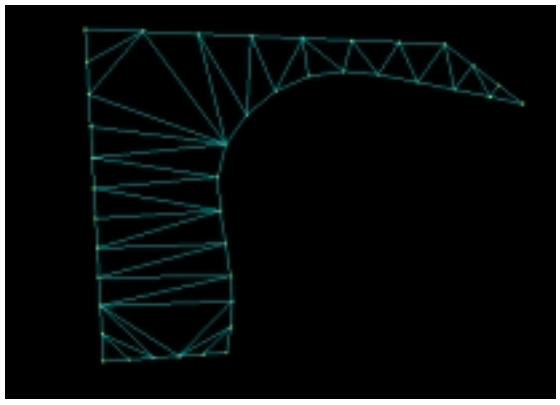


図 7-12　初期三角形分割

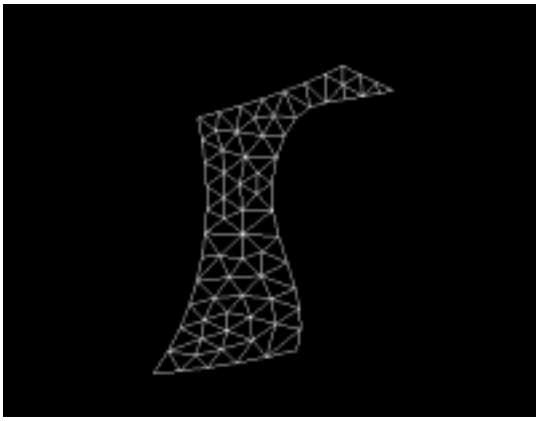


図 7-13　面上の表面パッチ

＜ソースヒント＞

・ 初期三角形分割

AdvTriPatch-x.x/IGES5.3/triangulation2D の下のソースを参照。

・ その他

AdvTriPatch-x.x/IGES5.3/triangulation3D の下のソースを参照。

8　コマンドについて

8.1　コマンド一覧

以下にデバッグの際に, よく利用するコマンドを表 8-1 に示す。これらのコマンドは, 十分なテストを行っていないものも含まれるが，インプリメントを参照することでソースの理解を深める際に非常に役に立つ。本文書の多くの図も，コマンドを利用して作成したものである。

コマンドの形式は，おおよそ

コマンド　＋　サブコマンド　＋　引数

の形をとる。

例えば,

view1　　***-vsp***　　***5***

コマンド　サブコマンド　引数

補足）

多くのコマンドは，VRML 形式(Ver1.0)の wrl ファイルを出力するので，VRML のブラウザを用意する。私は vrweb を利用している。

・MS-Windows ならば

ftp://ftp.iicm.edu/pub/VRweb/ から入手できる。

・Linux ならば

例えば debian ならば，deb パッケージがあるので

% apt-get install vrweb

で入手できる。

表 8-1　コマンド一覧

NO	コマンド	説明	コマンド
1	**read** 読み込み	IGES 読み込み	*read　IGES* ***ファイル名*** *ex)* *read　test.igs*
2	**info** 立体情報	立体のバウンディングボックを表示	*info　-bb*
		立体の位相要素の数を表示	*info　-ss*
3	**density** 節点間隔	ベースの節点間隔を設定	*density -base* ***節点間隔*** *ex)* *density　-base　10.0*
4	**boundary** 稜線の分割	稜線の分割	*boundary　–density* ***＊****density* ***コマンドが既に実行されているいることが条件***
5	**patch** 表面パッチ作成	表面パッチ作成	*patch　–density* ***＊****density* ***コマンドが既に実行されているいることが条件***
6	**view1** 幾何情報及び表面パッチの一部作成 *vew1 コマンドは、VRML(Ver1.0)で出力される	曲面情報の出力	*view1　-allSurface*
		曲面の一部出力	*view1　–vsp 面番号* ***＊****boundary* ***コマンドが既に実行されているいることが条件***
		曲面と曲線の関係の出力	*view1　–vsc 面番号* ***＊****boundary* ***コマンドが既に実行されているいることが条件***
		曲面上の３角形の出力	*view1　-patch2D 面番号* ***＊****boundary* ***コマンドが既に実行されているいることが条件***
		曲面上の３角形の出力	*view1　-patch3D 面番号* ***＊****boundary* ***コマンドが既に実行されているいることが条件***
7	**write** 保存	表面パッチを pch　形式で保存	*write – adventure* ***＊****patch* ***コマンドが既に実行されているいることが条件***
		pcg 形式で保存	*write　–groupInfo* ***＊****patch* ***コマンドが既に実行されているいることが条件***
		VRML 形式で保存	*write　–drawFace* ***＊****patch* ***コマンドが既に実行されているいることが条件***
8	**end** 終了	MaskMelon の終了	*end*

8.2　コマンドのサンプル

図 8-1 の図形に対して，view１コマンドを利用していろいろ作図してみる。サンプル形状は，上面及び側面及び下面の３つの面から構成される。また頂点数は２，稜線数は 3 である。図 8-2 では全ての曲面と稜線の関係を把握することができる。図 8-3 では，指定した面のみ把握できる。図 8-4 では，指定した面と稜線の関係を把握することができる。

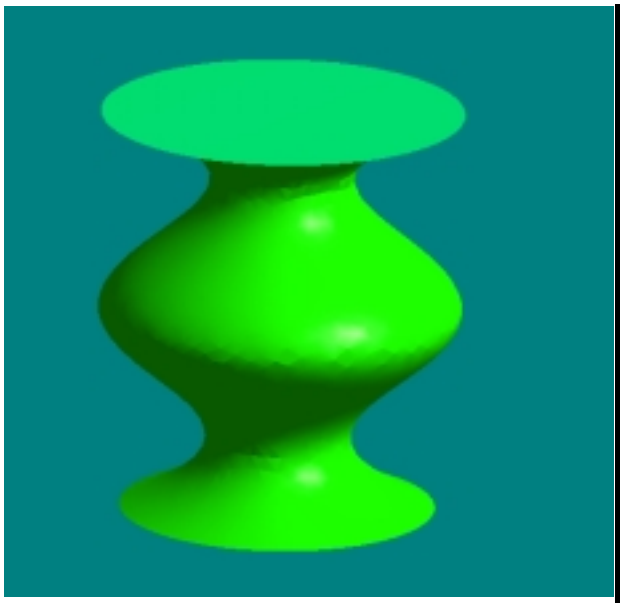


図 8-1　形状

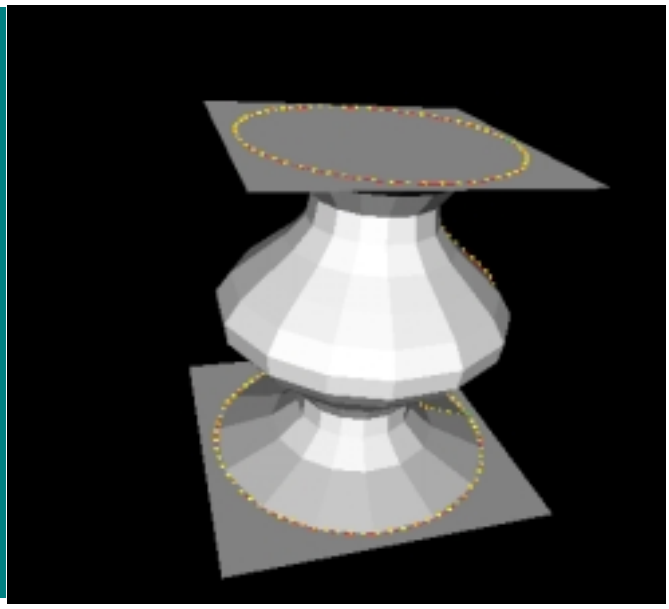


図 8-2　view1 –allSurface 使用

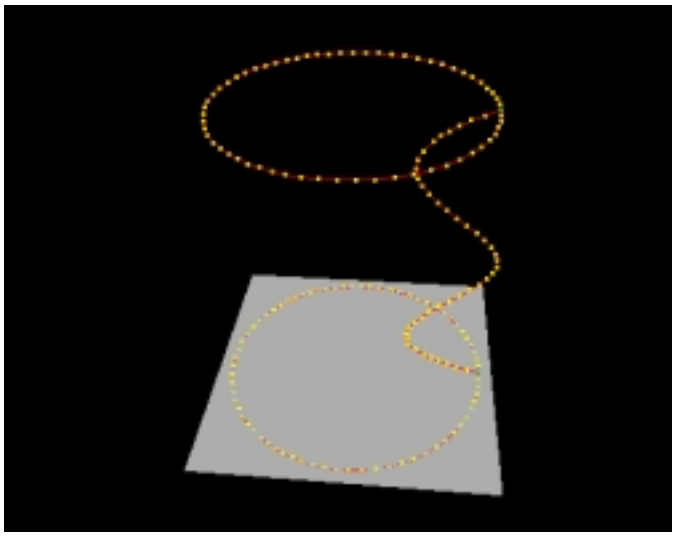


図 8-3　view1 –vsp 使用

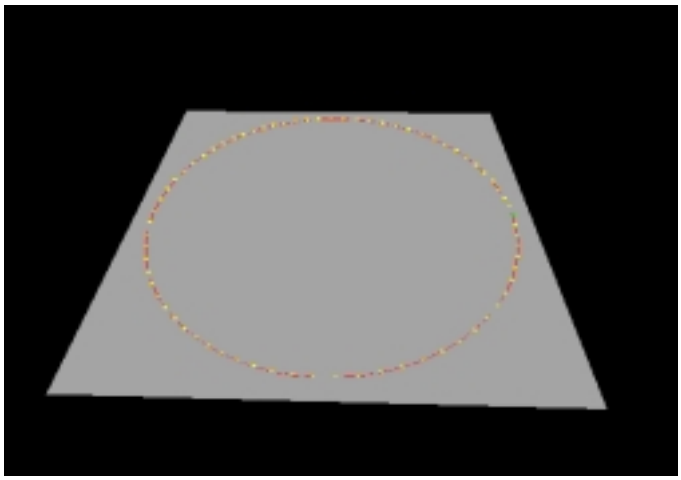


図 8-4　view1 –vsc 使用

図 8-5, 図 8-6 では，指定した面（形状の側面）の表面パッチが曲面上（パラメータ空間上）でどのように構成されているかを把握できる。図 8-7 では，指定した面（形状の側面）の表面パッチが把握できる。

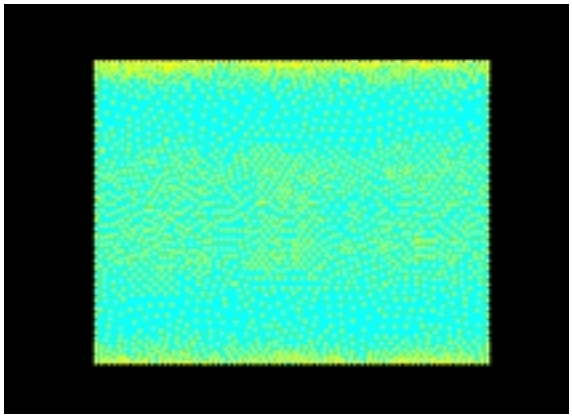


図 8-5　　view1 –patch2D 使用

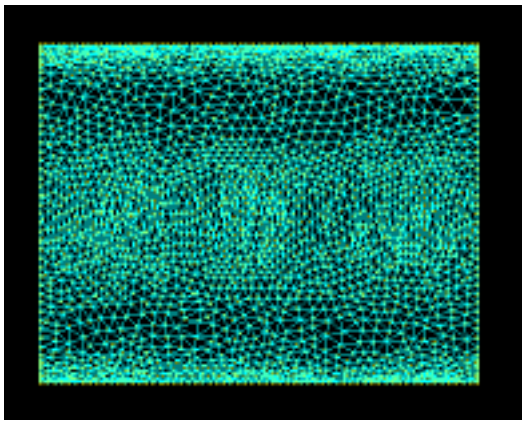


図 8-6　　view1 –patch2D 使用

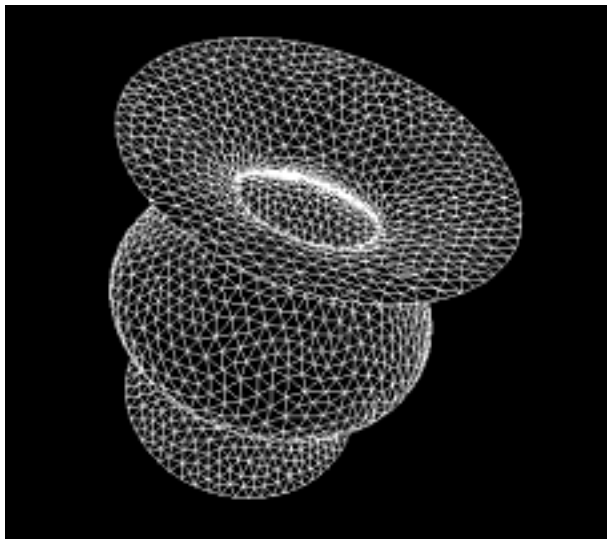


図 8-7 view1-patch3D 使用

9　表面パッチを作ろう

(1) ADVENTURE の文字

節点数 126,050 で表面パッチ数 252,076 で作成した結果を図 9-1, 図 9-2 に示す。

図 9-1 表面パッチの面塗り

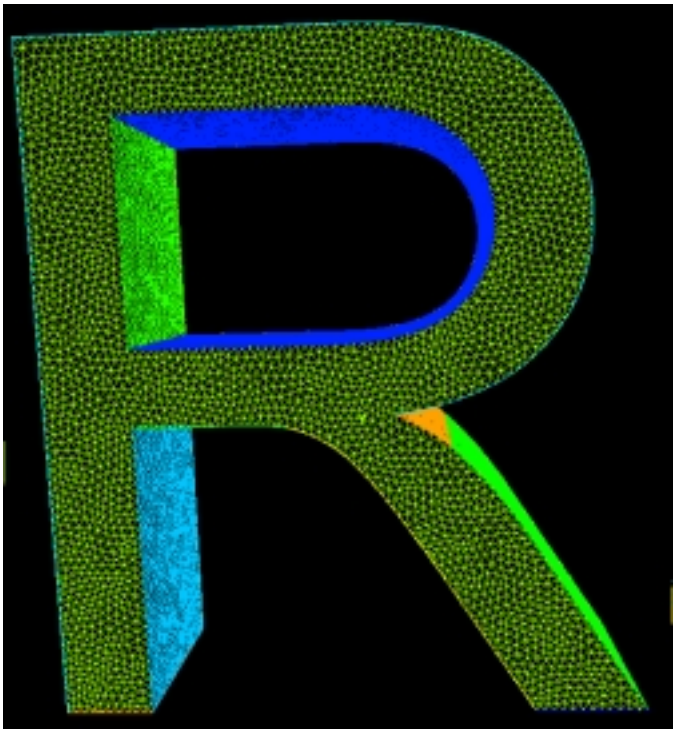


図 9-2　　R 文字

10 ファイル仕様

本プログラムで使用するファイルを以下に示す。

ファイル名	ファイルの概要
IGES データファイル	CAD で作成された IGES フォーマットのファイル。
節点密度データファイル	３角形パッチの疎密制御に使用するデータファイル
表面パッチデータグループファイル	表面パッチのグループ化情報を含むファイル
表面パッチデータファイル（旧）	本プログラムで出力される節点座標，３角形パッチの情報、を含むデータファイル
表面パッチデータファイル	本プログラムで出力される節点座標，3角形パッチ，領域を含むデータファイル。

10.1 IGES データファイル

(1) IGES 仕様書 Ver5.3 に準拠。(ASCII フォーマット)

(2) NURBS（有理化 B スプライン）曲面ベースのソリッドの入力に対応。
- IGES データが、ソリッド作成されている場合、エンティティ番号 186 が存在する
- エンティティ番号 186 が存在しない場合、本プログラムでは、エラーとなる。

(3) 本プログラムは、以下の CAD から出力される IGES ファイルを想定している。
a. I-DEAS MasterSerise 8
b. MicroCADAM V4R2

(4) 対応しているエンティティ

No	エンティティ番号	エンティティ名
1	100	円弧
2	110	線
3	124	変換マトリックス
4	126	有理化 B スプラインカーブ
5	128	有理化 B スプラインサーフェース
6	186	多様体ソリッド B-Rep オブジェクト
7	502	頂点
8	504	辺
9	508	ループ
10	510	面
11	514	シェル

10.2 表面パッチデータファイル（旧）

以下に表面パッチデータファイルのフォーマットを示す。

・表面パッチの法線ベクトルは、形状の内部方向に向かうように設定されている。

・拡張子は pch

```
1023                              <- 節点数
0.000000e+00  1.104000e+01  2.290000e+00   <- 0 番目の節点の X,Y,Z 座標
0.000000e+00  1.104000e+01  5.509000e+01
0.000000e+00  1.204000e+01  5.509000e+01

       ～ 省略 ～

3.651350e+00  -1.636939e+01  1.442530e+00
9.217112e+00  -1.397895e+01  1.320754e+00
1.344038e+01  -9.927832e+00  1.158236e+00  <- 1022 番目の節点の X,Y,Z 座標
2046                              <-表面パッチ数
   153      55     412             <-表面パッチを構成する節点番号の並び
   413     416     272
   419     418     417

       ～ 省略 ～

  1022     145     151
   124     145    1022
   121    1022     151
```

10.3　**表面パッチデータファイル**

以下に表面パッチデータファイルのフォーマットを示す。

・表面パッチの各領域（ボリューム）のコネクティビティは、形状の外からみて右周り。

・拡張子は pcm

・ボリューム境界の表現については、「図 10-1　表面パッチデータファイルのボリューム境界の表現」を参照。

```
NV　0　NR                    ←　　頂点数、予備（0 を入力））、領域数
x[0]　y[0]　　z[0]           ←　　頂点座標（NV 行）
x[1]　y[1]　　z[1]
x[2]　y[2]　　z[2]
～　　省略　　～
x[NV-1]　y[NV-1] z[NV-1]
（以下のブロック NR 回繰り返し）
  NP0　0　　0                ←　表面パッチ数、予備(0 を入力)、予備（0 を入力）
  e1[0]　e2[0]　e3[0]        ←　パッチコネクティビティ（NP0）
  e1[1]　e2[1]　e3[1]
  e1[2]　e2[2]　e3[2]
～　　省略　　～
  e1[NP0 - 1]　e2[NP0 - 1]　e3[NP0 - 1]
```

表面パッチデータファイルのフォーマットのサンプル

```
*pcm ver. 1.0              <---ファイルフォーマットのバージョン
347  0   2                 <--- 頂点数   予備１    領域数
0.0  0.0  0.0              <---0 番目の頂点座標（X,Y,Z 座標）
1.0  9.0  88.0             <---1 番目の頂点座標（X,Y,Z 座標）
          ～　省略　～
84.05 34.6  98.1           <---346 番目の頂点座標（X,Y,Z 座標）
5786  0  0                 <--- ボリューム０番目の表面パッチ数　予備２　予備３
153   55    412           <-0 番目の表面パッチを構成する節点番号の並び
567   45    34            <-1 番目の表面パッチを構成する節点番号の並び
          ～　省略　～
567   45    34            <-5785 番目の表面パッチを構成する節点番号の並び
456  0  0                 <--- ボリューム１番目の表面パッチ数　予備２　予備３
99    42    765           <-0 番目の表面パッチを構成する節点番号の並び
19    32    67            <-1 番目の表面パッチを構成する節点番号の並び
          ～　省略　～
99    23    21            <-455 番目の表面パッチを構成する節点番号の並び
```

ボリューム０番目の表面パッチ情報

ボリューム１番目の表面パッチ情報

注）予備１、予備２、予備３については予備項目です。（現在は全て０。）

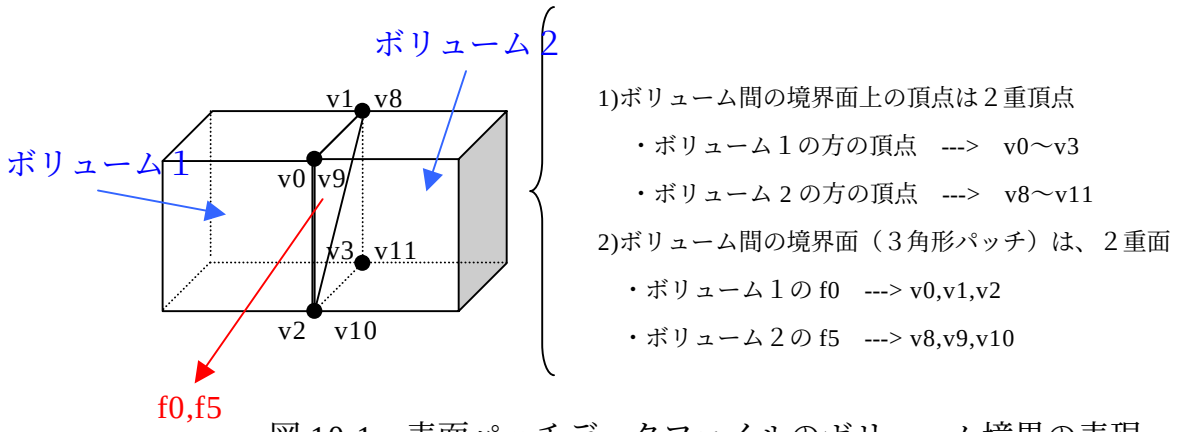


図 10-1　表面パッチデータファイルのボリューム境界の表現

10.4　表面パッチデータグループファイル

```
#mainVertexInfo
mainVertexN 299          ＜－ メイン節点の数（注-1）
0                        ＜－ 0 番目のメイン節点
1                        ＜－ 1 番目のメイン節点
        ～ 省略 ～
10
27
        ～ 省略 ～
2161
2162                ＜－ 299-1 番目のメイン節点

#edgeGroupInfo
edgeGroupN 305      ＜－ エッジグループの数
edgeGroup 2         ＜－ 0 番目のエッジグループを構成する節点の数
0                   ＜－ 0 番目のエッジグループの 0 番目の節点
1                   ＜－ 0 番目のエッジグループの 1 番目の節点
edgeGroup 2         ＜－ 1 番目のエッジグループを構成する節点の数
0                   ＜－ 1 番目のエッジグループの 0 番目の節点
10                  ＜－ 1 番目のエッジグループの 1 番目の節点
        ～ 省略 ～
edgeGroup 2         ＜－ 305-1 番目のエッジグループを構成する節点の数
9                   ＜－ 305-1 番目のエッジグループの 0 番目の節点
30                  ＜－ 305-1 番目のエッジグループの 1 番目の節点
#faceGroupInfo
faceGroupN 8        ＜－ 面グループの数
faceGroup 470       ＜－ 0 番目の面グループを構成するパッチの数
0                   ＜－ 0 番目の面グループの 0 番目のパッチの番号
1                   ＜－ 0 番目の面グループの 1 番目のパッチの番号
        ～ 省略 ～
469                 ＜－ 0 番目の面グループの 470-1 番目のパッチの番号
        ～ 省略 ～
faceGroup 39        ＜－ 8-1 番目の面グループを構成するパッチの数
4283                ＜－ 8-1 番目の面グループの 0 番目のパッチの番号
4284                ＜－ 8-1 番目の面グループの 1 番目のパッチの番号
        ～ 省略 ～
4321                ＜－ 8-1 番目の面グループの 39-1 番目のパッチの番号
```

（注-1）メイン節点とはモデルの形状を特徴づける代表的な節点です。

10.5 節点密度データファイル

(1) 節点密度データの概要

節点密度データは、ベース節点間隔とローカル節点密度に分類されます。

a. ベース節点間隔

表面パッチの稜線長を指定します。この長さに従うように表面パッチが作成されます。

b.ローカル節点密度

入力形状の任意の個所の表面パッチを細かくしたい場合に利用します。ローカル節点密度は、“点からの距離に反比例”、“線分からの距離に反比例”（２パターン）があります。ローカル節点密度を指定する場合、節点密度を設定する場所（*x,y,z* 座標）、適用範囲、密度の強さのパラメータを設定します。

(2) 節点密度適用例

図 10-2～図 10-4 に節点密度適用例（注）を示します。節点密度適用例は、“点からの距離に反比例 ”と “線分からの距離に反比例（２パターン）”の計３パターンについて示してあります。

・各パターンの左図は、適用結果図、右図は密度と距離の関係図を示しています。
・密度と距離の関係図の横軸 r または r_1～r_4 は距離、縦軸 d は密度です。
・密度と距離の関係図の距離とは、“点からの距離に反比例”の場合、指定した点からの距離を示し、“線分からの距離に反比例”の場合、指定した線分からの距離を示します。
・“点からの距離に反比例”、“線分からの距離に反比例”は距離と密度の関係が反比例です。（但し、“線分からの距離に反比例”の１パターンについては、線分からの距離で、密度を制御することが可能です。）

＜例＞

例として図 10-2 “点からの距離に反比例 ”をピックアップして説明します。この密度を適用すると点からの距離が離れるに従って、密度が低下していきます。(点から離れると、節点間隔が大きくなる.)

（注）

トップディレクトリの下の *sample_data* に本“節点密度適用例 ”で使用したサンプルデータが格納されています。

（*adventure_manual_data02.igs, adventure_manual_data02.ptn*）

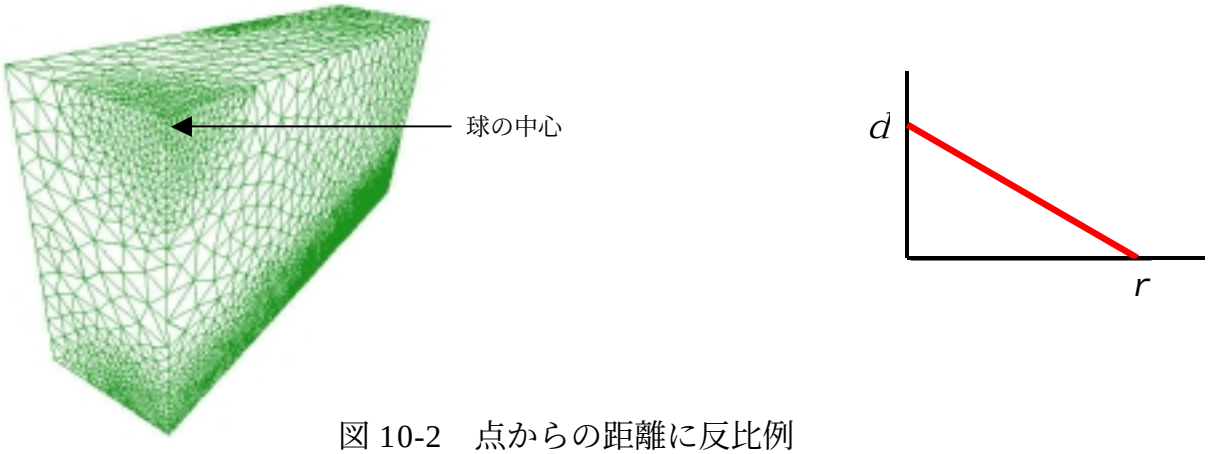


図 10-2　点からの距離に反比例

（*NodalPatternOnPoint* を使用）

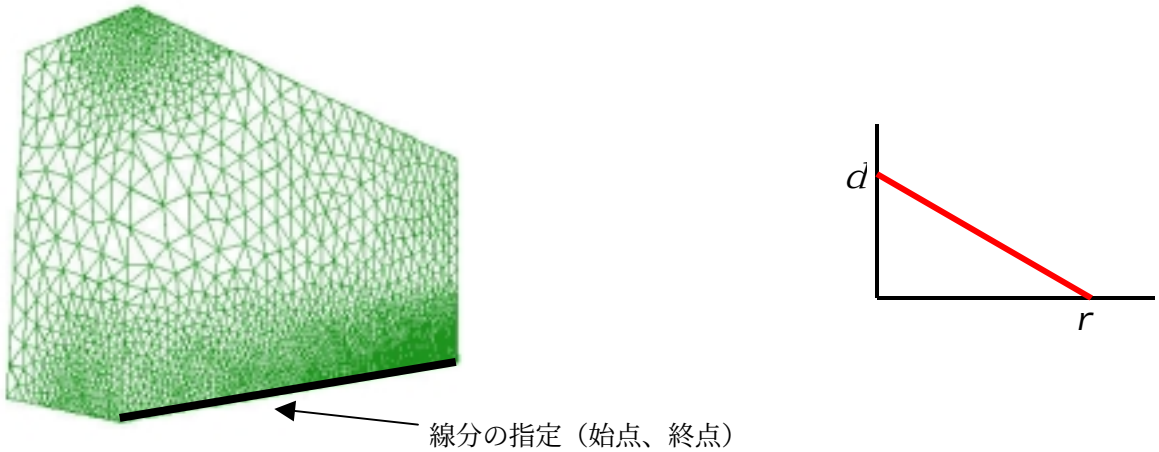


図 10-3　線分からの距離に反比例

（*NodalPatternOnLine* を使用）

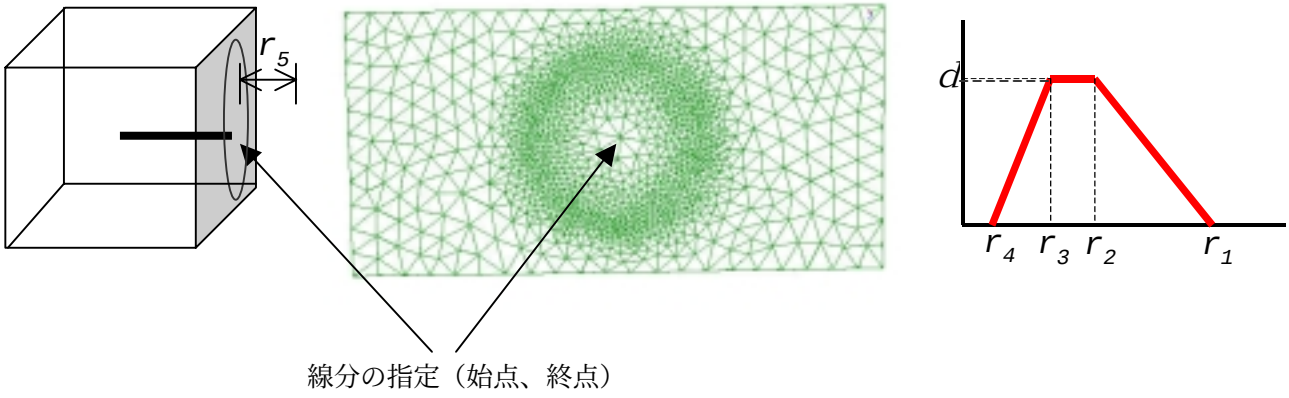


図 10-4　線分からの距離に反比例

（*NodalPatternOnCylinder* を使用）

(3) 節点密度データファイルのフォーマット

以下に節点密度データファイルのフォーマットを示します。

```
BaseDistance                          <----- ベース節点間隔
  1.00E+00

NodalPatternOnPoint                    <----- 点からの距離に反比例
  2.00E+01      4.7                    <----- 球の中心からの範囲(r)    密度の強さ
  1.00000E+01   0.00000E+00   0.00000E+00   <----- 球の中心座標

NodalPatternOnLine                     <----- 線分からの距離に反比例
  2.00E+01      4.7                    <----- 線分からの範囲(r)       密度の強さ
  1.00000E+01   0.00000E+00   0.00000E+00   <----- 線分の始点座標
  1.00000E+01   2.00000E+00   0.00000E+00   <----- 線分の終点座標

NodalPatternOnCylinder                 <-----線分からの距離に反比例（節点密度の範囲指定が可能）
12.0    10.0    9.0    8.0    3.0    1.5    <--- 範囲１～範囲５(r1～r5), 密度の強さ
347.1    0.0    100.0                      <----- 線分の始点座標
406.1    0.0    100.0                      <----- 線分の終点座標
```

・プログラムを実行する際、必須項目は ***BaseDistance*** です。

・　その他の項目（*NodalPatternOnPoint*、*NodalPatternOnLine*、*NodalPatternOnCylinder*）は、入力形状の任意の箇所の表面パッチを細かくしたい場合に利用します。

11 参考文献

[1] 鳥谷浩志,千代倉弘明.　３次元 CAD の基礎と応用.　共立出版株式会社,1995.

[2] USPRO.　Initial　Graphics　Exchange　Specification 5.3 . USPRO,1997.

[3]川合慧.コンピュータグラフィックス.日刊工業新聞社.1997.

[4]Hao Chen,Jonathan Bishop. Delaunary Triangulation for Curved Surfaces. Proceedings, 6th International Meshing Roundtable, Sandia National aboratories, pp.115-127, 1997