MGCL 技術情報

1. MGCL のモジュール構成

MGCLのクラス、関数は下記のようなモジュールに分類されています:

(1) Base Class

MGCLのベースとなるクラス群で、行列演算、ベクタ、各種コンテナ、ボックス枠などが中心となります.

(2) Functions for MGCL

MGCLで利用している数値計算関数など有用な関数をまとめてあります.

(3) Geometry(sub) classes

点(MGPoint)、線(MGCurve)、面(MGSurface)の幾何表現のためのクラスで MGGeometry のサブクラスになります

(4) Topology(sub) classes

頂点(MGPVertex, MGBVertex)、エッジ(MGEdge)、ループ(MGLoop)、トリム曲面(MGFace)、シェル(MGShell)の位相構造表現のためのクラスで MGTopology のサブクラスになります

(5) GeoRelated classes

Geometery クラスの入出力パラメータに利用される各種クラス(MGCoons, MGPPRep, 端末条件、など)が分類されます.

(6) TopoRelated classes

Topology クラスの入出力パラメータに利用される各種クラス(MGFOuterCurve, MGLPoint)や、曲面のトリム処理のユーティリティクラスの MGTrimLoop などのクラスが分類されます *

(7) Object Related classes

Geometry, Topology に共通(MGFSurface)や、その分類に当てはまらないクラス(MGSTL)、MGObject のピックのための情報コンテナクラスが分類されます.

(8) Gel Related classes

MGGel は Group Element(Gel)となりうるクラスの抽象クラスで、すべての MGObject 関連の最高位の抽象クラスとなります。グループ内の Gel の位置を表現する MGGelPosition などのコンテナクラス、MGGel 配下のすべてのクラス分類のための MGAbstractGel, MGAbstractGels などが分類されます.

(9) File InputOutput classes

MGGel 配下のクラスはまた、MGCL の用意するファイル入出力の対象でもあります。このためにクラス MGOfstream, MGIfstream, そして IGES データの入出力のクラス(MGIgesFstream, GIgesIfstream, MGIgesOstream)が分類されます.

(10)Intersection Container classes

点(MGPoint)、線(MGCurve)、面(MGSurface)の幾何表現同士、または MGEdge, MGLoop, GFace, MGShell などの Topology クラスとの交点とその配列を表現します.

(11)UseTessellation classes

面(MGSurface, MGFace)を表示するためにはそれらを細かな三角形で近似する必要があります。

UseTessellation は三角形近似のためのツールクラスを提供します。 mgTLData, gTLDataVector がそれで、この配下に、この機能の実現のために多くのクラスがありますが、一般利用者はこのふたつのクラスで十分です.

(12)Display Handling classes

MGCL では図形の表示に OpenGL, イメージデータの実現のために GDI-plus、マンマシンインタフェースに MFC(Micrsoft Foundation Class)を利用しています。Display Handling Classes にはMGCLのオブジェクトの表示のためのクラスと表示のためのアトリビュートなどが分類されています.

2. ベクタ、ポジションクラス(MGVector, MGPosition)

2.1 コンストラクタ

コンストラクタの最初のアーギュメントに sdim(空間次元、space dimension の略)として数値を引数とするものが提供されているが、この数値は空間次元を指定するものである。

```
//Void constructor.
explicit MGVector(size_t sdim=0);

// 初期値 v ですべてのエレメントを初期化してベクトルを生成する。
//Vector of same value for each coordinate element.
MGVector(size_t sdim, double v);

// double の配列でエレメントを初期化してベクトルを生成する。
//Vector from array of double v[sdim].
//***** This is the fundamental constructor.*****
MGVector(size_t sdim, const double* v);
```

空間次元を指定するのは、その空間次元の領域を確保するため。Operator()(size_t i)がいずれのクラスにも用意されているが、operator()は左辺値用であるので、指定できる i はコンストラクタで指定した次元の数よりも小さな数に制限される。Operator[]にはこの i の値としてどのような正の値でも使用可能であるが、定義されている空間次元よりも大きな次元を参照すれば、値としてゼロ(0. 0)が返される。

(例)

MGVector v3(3); //v3 は3次元のベクタで v3(0)=1.; v3(1)=2.; v3(2)=3.;の記述が可能。これらの記述は
//それぞれ、x, y, z 座標値を 1., 2., 3.に設定している。
MGVector v2(2); //v2 は 2 次元のベクタで v2(0)=1.; v2(1)=2.;の記述は可能であるが、
//v2(2)=3.;の記述はメモリーアクセス違反となる。しかし
double zvalue=v2[2];//として v2 のz値を参照すれば、zvalue=0.となる。

2.2 Operator()と operator[]

MGVector, MGPosition の各座標値への添え字によるアクセスのため、operator()と operator[]が用意されているが、この二つは役割が異なる。Operator()は左辺値用(すなわち、その添え字の値を変更するため)のものであるが operator[]は右辺値用(すなわち参照用)であり、左辺値には使

用できない。利用者は座標値の一つを参照するだけで、更新をしないのであれば、operator[]を使用するのがパフォーマンス上良い。

（例）
v3(0)=1.;//v3 のx座標値を 1.に変更する。しかし、v3[0]=1.;の記述はコンパイルエラーとなる。
If(v[0]>=1.){…};は参照だけであるので、OKである。

2.3 空間次元０（ゼロ）

空間0のベクタ、Position はヌル（null）であると呼ぶ。デフォルトコンストラクタは null なものを生成する。これを調べるために is_null()、設定のため set_null()関数が用意されている。

(is_null()、set_null()に関しては MGMatrix、MGTransf、MGBox に関しても同様である)

2.4 データ領域

ベクタ、position ともに空間次元には制約がない。しかし、パフォーマンス向上のために、3次元までは自動メモリ領域（automatic memory, またはスタックメモリとも呼ばれる）が利用され、4次元以上は自由記憶領域（dynamic memory, またはヒープメモリと呼ばれ、new. delete により確保、解放される）が利用される。ベクタ、Position の内容をデバッガで見るとき、3次元までは MGVector の m_data に記憶されるが、4次元以上では m_data の領域は使用されないで、m_element ポインタで示される領域に記憶されるので、注意が必要。

2.5 MGPoint と MGPosition

空間上の点を表現するクラスは MGPoint と MGPosition とふたつ提供されている。MGPoint は MGObject のサブクラスとして幾何学図形を表現するためのもので、MGObject としての各機能を備えていて重いクラスとなっている。この重さを避けるために同じ点を表現するクラスとして MGPosition が用意されている。利用者はできるだけ MGPoint の使用を避け、MGPosition を利用すべきである。

2.6 MGVector の外積

MGVector には外積 operator*が用意されている。外積は3次元空間だけに定義され、4次元以上の空間には一般に定義されない。そこでMGCLでは次のように外積を拡張している:

（1） 2次元以下のベクタに関しては欠けているエレメントの値を 0.と考え3次元とする。

（2） 4次元以上に関しては次の性質を持つベクトル $c = a * b$ をベクトル a と b との外積であるとする。

1） c は a または b と直角である。すなわち、内積 $c\%a = 0$、かつ $b\%a = 0$ である

2） a と b とのなす角度を $a\%b = |a||b|\cos\theta$ から得られる θ とすると、$|c| = |a||b|\sin\theta$ が成立する

3） $c = a * b = -(b * a)$

3. MGMatrix と MGTransf

MGCL ではマトリクス機能として MGMatrix（移動を伴わない座標変換）と MGTransf(移動を伴う一般的な座標変換）が用意されている。座標変換（マトリクス）にはその順序が重要であるが、

MGCL では次のスキーマを採用している:

$[x \quad y \quad z][M]$ または $[x \quad y \quad z \quad 1][T]$ （3次元の場合）

ここで、Mは MGMatrix であり、Tは MGTransf である。すなわち、座標変換対象のオブジェクトは M

または T の左から乗算される。これは$[x \quad y \quad z]$を MGObject の各オブジェクトに置き換えても同様である。$[T]\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$ ではないので注意を要する。

空間次元を3とすると(2、または4以上も同様である):

$[M]=\begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{bmatrix}$ と表現され、$[T]=\begin{bmatrix} t_{00} & t_{01} & t_{02} \\ t_{10} & t_{11} & t_{12} \\ t_{20} & t_{21} & t_{22} \\ x_0 & y_0 & z_0 \end{bmatrix}$ と表現できる。$[T]$を行列演算する際には $[T]=\begin{bmatrix} t_{00} & t_{01} & t_{02} & 0 \\ t_{10} & t_{11} & t_{12} & 0 \\ t_{20} & t_{21} & t_{22} & 0 \\ x_0 & y_0 & z_0 & 1 \end{bmatrix}$ と考えて演算される。$[T]$ は M を用いて

$[T]=\begin{bmatrix} m_{00} & m_{01} & m_{02} & 0 \\ m_{10} & m_{11} & m_{12} & 0 \\ m_{20} & m_{21} & m_{22} & 0 \\ x_0 & y_0 & z_0 & 1 \end{bmatrix}$ と考えることができる。

4. MGGel 配下のクラス分類について

クラス MGGel の Gel は Group Element の略で、MGGroup にメンバとして含めることができる各クラスを総称するための抽象クラスである。

4.1 Type id

MGCL のオブジェクトは type id により管理される。32ビットの unsigned integer の16進表示にて 0xmmttnnkkL と表現したとき、mm により、次のようにクラスが分類される。

mm=0: MGObject,

=1: MGGroup,

=2: MGAttrib

また、クラスの種類によらず、nn は多様体次元を表す。たとえば、点は nn=00, 線は nn=01, 面は nn=02 である。

MGObject の分類

MGObject は 0xmmttnn00L の mm=00 であり、MGGeometry は tt=10, MGTopology は tt=2x である。MGTopology のサブクラスの場合 tt=2x の x は Cell に対して x=0, Complex に対して x=1 となる。

Type id 0xmmttnnkkL

	mm クラスの大分類	tt	nn:多様体次元 00: 点、01: 線、02: 面	kk 個別種類
MGObject	00	10: MGGeometry 2x: MGTopology 20: Cell 21: Complex	0000: MGPoint 01kk: MGCurve 02kk: MGSurface	
MGGroup	01		00	00:MGGroup 02:MGApprearance
MGAttrib	02	01: MGGLAttrib		

ファイル"types.h"

4.2 MGAbstractGel と MGAbstractGels

ピック処理するときなど、オブジェクトを分類し、その分類を指定する必要があることがある。このために用意されているのが MGAbstractGel と MGAbstractGels である。

```
//MGAbstractGel is a class to specify what kind of abstract gel group.
//Let MGAbstractGel agel(gel_kind, gel_tid), then
//gel_kind is either MGALL_GELL, MGTOP_KIND, MGMANIFOLD, MGGEO_TOPO,
//MGGEO_KIND, or MGLEAF_KIND. And gel_tid is the specific type of
//the gel_kind. For the value of gel_tid, see MGGEL_KIND above.
//Possible combinations are defined in MGDefault.h(as mgAll_xxxx).
//See the definition.
typedef std::pair<MGGEL_KIND,MGGEL_TID> MGAbstractGel;
```

MGAbstractGel

MGAbstractGel は上記のような typedef で MGGEL_KIND と MGGEL_TID のペアである。MGGEL_KIND は type id の分類方法により、どのような MGGel サブクラスを指定するかを定義する。下記は MGEL_KIND の enum のリストだが、ここでは32ビットの16進表示を定義していて、そのビット位置は上記の type id に対応している。調査すべき type id のビット位置の値が 16 進の f(すなわち2進数ですべて 1)となっており、MGGEL_KIND と調査対象の MGGel の type id とで AND 演算を施したとき、調査すべきフィールドのビット位置のデータが残る。Type id は MGAbstractGel の MGGEL_TID で指定され、調査対象の type id と MGGEL_KIND と AND 演算の結果が MGGEL_TID と比較され、一致していればその MGAbstractGel のので分類される MGGel のサブクラスであるとされる。この MGAbstractGel の具体的なものは mgAll_xxxx として Default.h に定義されている。また、MGGEL_KIND, MGGEL_TID は type.h に定義されている。

```
//MGGEL_KIND_TID is used to specify what kind of group is used to identify gels.
enum MGGEL_KIND{
MGALL_GELL = 0x00000000L,//all of the gels
MGTOP_KIND=     0xff000000L,
    //subkind is MGOBJECT_TID, MGGROUP_TID, MGATTRIB_TID(
MGMANIFOLD=     0xff00ff00L,
    //subkind is MG0MANIFOLD, MG1MANIFOLD, MG2MANIFOLD, MG3MANIFOLD
MGFSURFACE_KIND=0xff0fff00L,
    //subkind is MGFSURFACE_TID(MGFace or MGSurface, MG2MANIFOLD that is not
    //MGShell)
MGGEO_TOPO=     0xfff00000L,//MGGeometry, or MGTopology.
    //subkind is MGGEOMETRY_TID, MGTOPOLOGY_TID
MGGEO_KIND=     0xffffff00L,
    //subkind is MGPOINT_TID0, MGCURVE_TID, MGSURFACE_TID
MGLEAF_KIND=0xffffffffL
//subkind is all of the leaf class MGGEL_TID, that is:
```

```
  //MGPOINT_TID,MGSTRAIGHT_TID,MGELLIPSE_TID,MGLBREP_TID,MGRLBRE
P_TID,
  //MGSRFCRV_TID,MGTRMCRV_TID,MGCOMPCRV_TID,MGPLANE_TID,MGSPHE
RE_TID,
  //MGSBREP_TID,MGRSBREP_TID,MGCYLINDER_TID,MGPVERTEX_TID,
  //MGBVERTEX_TID,MGEDGE_TID,MGFACE_TID,MGLOOP_TID,MGSHELL_TID
  };
```

MGAbstractGels

MGAbstractGels は MGAbstractGel を配列化したものであり、複数の MGAbstractGel を指定したいときに用いる。

5. MGGel 配下の新しいクラスの追加

新しいクラスを追加するにあたって、必要な仮想関数を記述する。

5.1 MGGel の仮想関数

（1） 代入文：virtual MGGel& operator=(const MGGel& gel2);

MGGel のサブクラスのふたつのオブジェクトの代入については、そのリーフクラス種類が一致するものについては代入を許し、異なる種類のリーフクラスについては代入処理を無視するように、実装する。たとえば、クラス階層の最下層（リーフの）二つのクラス MGStraight と MGEdge を例にとると：

```
MGStraight Sl1, Sl2; MGEdge E1, E2;
          ⋮
Sl1=Sl2; //代入文が実行される。
Sl2=E1; //この代入文は無視され、何も実行されない。
```

MGGel::operator=(const MGGel& gel2);の仮想関数による実装では、同じリーフクラスであれば、そのリーフクラスへは結局アーギュメントが const LeafClass& の代入文定義を実行することとなるが、クラス階層のなかでその振り分けが必要となる。これを仮想関数機能と dynamic_cas により実現する。この代入文の実装は次のようになされる：

MGGel は次の関数宣言をしている：

```
//Assignment.
//When the leaf objects of this and gel2 are not equal, this assignment
//does nothing.
virtual MGGel& operator=(const MGGel& gel2){return *this;};
```

各クラスは、この default な機能を、クラスの種類が異なる時に利用（これが呼び出される）する。従い、次の、次の 2 つの種類の関数を宣言し、実装する：

① アーギュメントが MGGEL(const MGGel& gel2)の代入文。

この関数は MGGel に宣言されている default な代入文を override することとなる。この実装は次のようにする(ここでは MGStraight を例にしている):

```
MGStraight& MGStraight::operator=(const MGGel& gel2){
 const MGStraight* gel2_is_this=dynamic_cast<const MGStraight*>(&gel2);
 if(gel2_is_this)
       operator=(*gel2_is_this);
 return *this;
 }
```

Operator= (const MGGel&)は、MGGel 内で、MGStraight をアーギュメントとする関数宣言が仮想関数として宣言されていれば、関数のオーバーライドで、if 文や、dynamic_cast を利用することなく、実装可能であるが、これには、MGGel での関数宣言に、そのサブクラスのに対応するすべての代入演算子宣言が必要となる欠点がある。たとえば、新しいサブクラスを追加する度に MGGel の宣言を変更する必要がある。これを避けるため、ここでは、上記のプログラムリストのように daynamic_cast を利用し、アーギュメントのクラスが自身と同じクラスであった場合、だけ、自身のクラスの代入文を実行するようにしている。

② アーギュメントを自分と同じクラス名とする代入文(次は MGStraight の例)

```
//Assignment.
MGStraight& MGStraight::operator=(const MGStraight& sl2){
 if(this==&sl2)
       return *this;

 MGCurve::operator=(sl2);//親クラスの代入文の実行
            ⋮
 //各クラスのメンバーデータの代入処理
            ⋮
 return *this;
 }
```

(2) 論理比較演算子:operator== (), operator!= (), operator< (), operator> ()

MGGel ではこれらの比較演算子をサポートするため、次の関数宣言がなされている:

```
//Comparison.
virtual bool operator==(const MGGel& gel2)const{return false;};
virtual bool operator!=(const MGGel& gel2)const{return !(operator==(gel2));};
virtual bool operator>(const MGGel& gel2)const{return gel2<(*this);};
virtual bool operator<(const MGGel& gel2)const{
 long id1=identify_type(); long id2=gel2.identify_type(); return id1<id2;}
```

・ 特に定義のない比較演算子 operator<() は、上記の定義でわかるように、(特に異なるクラスの MGGel サブクラスについては)idyntify_type()の返り値の順序として定義される。

・　比較演算子は、かならずしもサポートする必要はないが、サポートする場合、上記の宣言を利用する。すなわち、各クラスは、次の仮想関数宣言に対応する関数を宣言、実装する:

```
virtual bool operator==(const MGGel& gel2)const{return false;};
virtual bool operator<(const MGGel& gel2)const;
```

これから、実際に実装が必要な関数は次のようになる(MGStraight の例)

```
//comparison
bool operator==(const MGStraight& sl2)const;
bool operator==(const MGGel& gel2)const;
bool operator<(const MGStraight& gel2)const;
bool operator<(const MGGel& gel2)const;


//
// 論理演算子の多重定義
bool MGStraight::operator==(const MGStraight& sl2)const{
//ふたつの MGStraight がおなじかどうかの判定処理
}
bool MGStraight::operator==(const MGGel& gel2)const{
 const MGStraight* gel2_is_this=dynamic_cast<const MGStraight*>(&gel2);
 if(gel2_is_this)
        return operator==(*gel2_is_this);
 return false;
 //gel2 を MGStraight に cast して、MGStraight の場合だけ、比較処理。その他はすべて
//false とする。
}

bool MGStraight::operator<(const MGStraight& gel2)const{
//ふたつの MGStraight の大小比較判定処理
}
bool MGStraight::operator<(const MGGel& gel2)const{
 const MGStraight* gel2_is_this=dynamic_cast<const MGStraight*>(&gel2);
 if(gel2_is_this)
        return operator<(*gel2_is_this);
 return MGGel::operator<(gel2);;
 //gel2 を MGStraight に cast して、MGStraight の場合だけ、比較処理。その他はすべて
// MGGel::operator<(const MGGel& gel2)を利用する。
}
```

6. 交線計算

6.1 線（MGCurve）と線(MGCurve)

MGCurve::intersect(const MGCurve& curve2); 最も汎用的な交線計算

実体のある交線計算一覧表

クラス	アーギュメント
MGCurve	MGCurve(デフォルトな交線計算の提供) (実体は MGCurve::intersect())
MGStraight	MGCurve(curve2=MGStraight は isect(MGStraight), その他は curve2.isect(const MGStraight&))
	Const MGStraight& sl
MGEllipse	MGCurve（curve2=MGStraight は isect(MGStraight), curve2=MGEllipse は isect(MGEllipse), その他は curve2.isect(const MGEllipse&)）
	MGStraight
	MGEllipse
MGLBRep (MGLBRep は自身が C0 級の 曲線であることを許容している。 そこで、C0 級の曲線を C1 級に 分解して交線計算をしている)	MGCurve（C1 級の部分曲線に分解して isect_1span(MGCurve)）
MGLBRep::isect_1span	MGCurve（ curve2=MGStraight は:isect_1span(MGStraight), curve2=MGLBRep は:isect_1span(MGLBrep), curve2=MGRLBRep は intersect (MGRLBRep), curve2=MGEllipse は intersect (MGEllipse), その他は curve2.isect(const MGLBrep&))
	MGStraight
	MGLBRep
	MGRLBRep
MGRLBRep	MGCurve（ curve2=MGStraight は:isect(MGStraight), その他は curve2.isect(const MGRLBRep&))
	MGStraight
MGBSumCurve	実体のある交線計算なし
MGCompositeCurve	MGCurve(構成曲線との交点の和集合としている)
MGTrimmedCurve	MGCurve(構成曲線の交点のうち、パラメータ範囲に入 っているものだけを選択)
MGSurfCurve	MGStraight
	その他(const MGCurve& curve2)は curve2.isect(const MGSurfCurve&)

	Straight	RLBRep	Ellipse	LBRep	SurfCurve	BSumCurve
Straight	Straight::Isect(Straight)	Curve2.Isect(Straight)	Curve2.Isect(Straight)	Curve2.Isect(Straight)	Curve2.Isect(Straight)	Curve2.Isect(Straight)
RLBRep	RLBRep::isect(Straight)	intersect(RLBRep)	Curve2.isect(RLBRep)	Curve2.isect(RLBRep)	Curve2.isect(RLBRep)	Curve2.isect(RLBRep)
Ellipse	Ellipse::isect(Straight)	intersect(RLBRep)	Ellipse::isect(Ellipse)	Curve2.isect(Ellipse)	Curve2.isect(Ellipse)	Curve2.isect(Ellipse)
LBRep (1span に変換後)	LBRep::isect(Straight)	Intersect(RLBRep)	Intersect(Ellipse)	Intersect(LBRep)	Curve2.isect(LBRep)	Curve2.isect(LBRep)
SurfCurve	SurfCurve::isect(Straight)	intersect(RLBRep)	intersect(Ellipse)	intersect(LBRep)	SurfCurve::isect(SurfCurve)	intersect(BSumCurve)
BSumCurve	intersect(Straight)	intersect(RLBRep)	Intersect(Ellipse)	Intersect(LBRep)	Intersect(SurfCurve)	Intersect(BSumCurve)

線(MGCurve)と面(MGSurface)

	Plane	Sphere	Cylinder	SBRep	RSBRep	BSumSurf
Straight	MGPlane::isect	MGSphere::isectSl()	MGCylinder::isectSl	MGSBRep::isectSl	MGRSBRep::isectSl)	MGSurface::isectSl
RLBRep	MGRLBRep::isect	MGSurface::intersect	MGSurface::intersect	MGSurface::intersect	MGSurface::intersect	MGSurface::intersect
Ellipse	MGPlane::isect	MGSurface::intersect	MGSurface::intersect	MGSurface::intersect	MGSurface::intersec	MGSurface::intersec

					t	t
LBRep	MGCurve::intersect_with_plane	MGSurface::intersect	MGSurface::intersect	MGSurface::intersect	MGSurface::intersect	MGSurface::intersect
SurfCurve	MGSurfCurve::isect_noCompo	MGSurface::intersect	MGSurface::intersect	MGSurface::intersect	MGSurface::intersect	MGSurface::intersect
BSumCurve	MGCurve::intersect_with_plane	MGSurface::intersect	MGSurface::intersect	MGSurface::intersect	MGSurface::intersect	MGSurface::intersect
TrimmedCurve	MGTrimmedCurve::isect	MGTrimmedCurve::isect	MGTrimmedCurve::isect	MGTrimmedCurve::isect	MGTrimmedCurve::isect	MGTrimmedCurve::isect
CompositeCurve	MGCompositeCurve::isect	MGCompositeCurve::isect	MGCompositeCurve::isect	MGCompositeCurve::isect	MGCompositeCurve::isect	MGCompositeCurve::isect

面(MGSurface)と面(MGSurface)

	Plane	Sphere	Cylinder	SBRep	RSBRep	BSumSurf
Plane	Plane::isect	Sphere::isect(plane)	Cylinder::isectPl(Plane)	Surface::intersectPl(Plane)	Surface::intersectPl(Plane)	Surface::intersectPl(Plane)
Sphere	Sphere::isectPl(plane)	Surface::intersect	Surface::intersect	Surface::intersect	Surface::intersect	Surface::intersect
Cylinder	Cylinder::isectPl(Plane)	Surface::intersect	Surface::intersect	Surface::intersect	Surface::intersect	Surface::intersect
SBRep	Surface::intersectPl(Plane)	Surface::intersect	Surface::intersect	Surface::intersect	Surface::intersect	Surface::intersect
RSBRep	Surface::intersectPl(P	Surface::intersect	Surface::intersect	Surface::intersect	Surface::intersect	Surface::intersect

	lane)					
BSum Surf	Surface::intersectPl(Plane)	Surface::intersect	Surface::intersect	Surface::intersect	Surface::intersect	Surface::intersect

	Surface	Face
Surface	Surface::isect(surface)	Face::isect(Surface)
Face	MGFace::intersect	MGFace::intersect

7. Perps(線と線がお互いに垂点となる線上の点の対を求める)

	Straight	RLBRep	Ellipse	LBRep	SurfCurve	BSumCurve
Straight	Straight::perps(Straight)	RLBRep.perps(Straight)	Ellipse.perps(Straight)	Perpendiculars(Straight)	Perpendiculars(Straight)	Perpendiculars(Straight)
RLBRep	perpsSl(Straight)	Perpendiculars(RLBRep)	Perpendiculars(RLBRep)	Perpendiculars(RLBRep)	Perpendiculars(RLBRep)	Perpendiculars(RLBRep)
Ellipse	Ellipse::perps(Straight)	Perpendiculars(RLBRep)	Perpendiculars(Ellipse))	Perpendiculars(LBRep)	Perpendiculars(SurfCurve)	Perpendiculars(BSumCurve)
LBRep	perpsSl(Straight)	Perps(RLBRep)	Perpendiculars(Ellipse)	perps_with_C1LB(LBRep)	Perpendiculars(SurfCurve)	Perpendiculars(BSumCurve)
SurfCurve	perpsSl(Straight)	Perpendiculars(RLBRep)	Perpendiculars(Ellipse)	perps_by_split_to_C1(LBRep)	Perpendiculars(SurfCurve)	Perpendiculars(BSumCurve)
BSumCurve	perpsSl(Straight)	Perpendiculars(RLBRep)	Perpendiculars(Ellipse)	Perpendiculars(RLBRep)	Perpendiculars(SurfCurve)	Perpendiculars(BSumCurve)

8. OpenGL を利用した表示とピック処理

8.1 ディスプレーリスト名

MGGelのメンバー関数としてディスプレーリスト名を求めるdlist_name()が用意されている。

dlist_name()には、３つの連続した番号として利用できるリスト名を返すことが要求される。

それらの番号（リスト名）は次のように利用される：

（１） dlist_name()が返却するリスト名：ワイヤモード（線描画）での表示用リスト名

（２） dlist_name()が返却するリスト名+1：強調表示用の色のアトリビュート処理のないワイヤモード（線描画）での表示用リスト名

（３） dlist_name()が返却するリスト名＋２：シェーディング表示用のリスト名

このディスプレーリスト名は各オブジェクトに用意される関数 dlist_name()により返される値を上記の最小のリスト名として使用する。

次のコードは現在の MGGel::dlist_name()のデフォルト名コードである。現在このようにしているのは、dlist_name()で返却される unsigned integer からその対象の MGGel＊を求めることが容易なためであり、64ビット対応の際にはこれを実装する必要がある。すなわち、

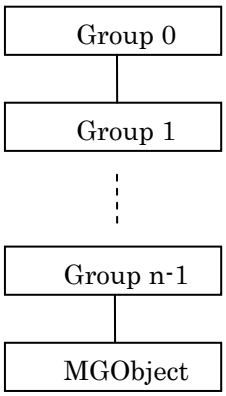
- Dlist_neme()から MGGel＊を求める関数
- MGGel＊から3つの連続した未使用のディスプレーリストを求める

```
//Obtain display list name. 0(null) means this gel need not to be displayed.
//The default is the pointer of this gel.
virtual size_t MGGel::dlist_name()const{return size_t(this);};
```

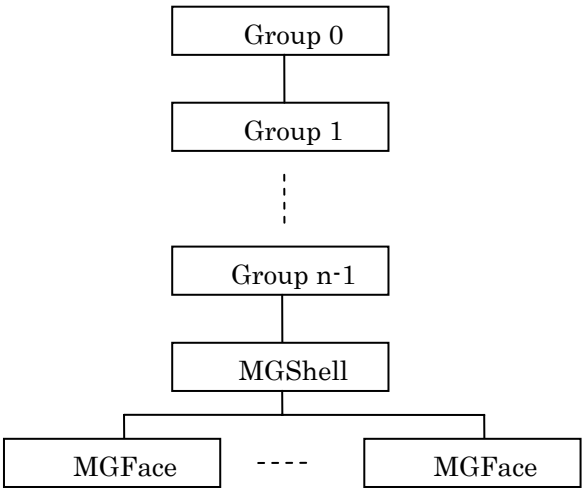
8.2 表示処理とディスプレーリスト名

表示処理の make_display_list()は表示処理の際に、同時に選択処理（ピック処理）のためにディスプレーリスト名を OpenGL の記憶させるための処理：glPushName()を実施しおり、make_display_list()処理により作成されてリストはオブジェクトの選択処理にもそのまま利用される。glPushName()により、ネームスタックと呼ばれるスタックには次のようにディスプレーリスト名が積まれる。

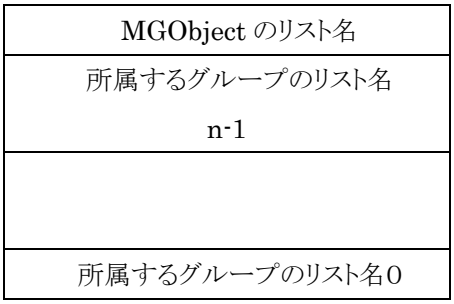
注意が必要なのは、MGShell 以外はすべてオブジェクトのディスプレーリスト名そのものがスタックに積まれるのに対し、MGShell だけはその配下の MGFace のリスト名が積まれることである。必要であれば、MGFace からその親の MGShell を用意されている関数で知ることができる。



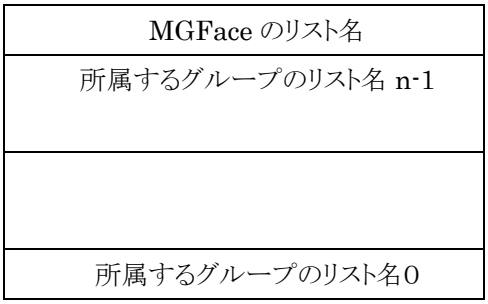
MGShell 以外のオブジェクトの階層構造



MGShell の階層構造



Shell 以外のオブジェクト



Shell 内の各 Face に対して

OpenGL のネームスタックへの積まれ方とオブジェクトの階層構造

次のリストは MGGroup::make_display_list()のワイヤフレーム表示のためのコードだけを取り出したプログラムコードである。

```
//Make a display list of this group.
size_t MGGroup::make_display_list(
double span_length//span length to approximate by polyline.
)const{
size_t glname=dlist_name();//display list name of this group
glNewList(glname, GL_COMPILE);
      glPushName(name);
      MGGroup::const_iterator i,is=begin(), ie=end();
      for(i=is; i!=ie; i++)
            glCallList((*i)->dlist_name());//call object.
      glPopName();
glEndList();
MGGroup::const_iterator i,is=begin(), ie=end();
//Make display list of the gel that has an object.
```

```
for(i=is; i!=ie; i++)
    (*i)->make_display_list(span_length);
return name;
}
```

次のプログラムリストは MGObject の make_display_list()のプログラムコードである
(ワイヤモード表示とシェーディングモード表示の双方を含む)

```
//Make 2 types of display list of this gel(wire and shading).
//Return is the display list name.
size_t MGObject::make_display_list(
double span_length//span length to approximate by polyline.
)const{
size_t name=dlist_name();
size_t glname=name;

//Draw wire mode display list.
glNewList(glname, GL_COMPILE);
    glPushName(name);
    draw3D(span_length);
    glPopName();
glEndList();

//Draw shading mode display list.
glname=name+2;
glNewList(glname, GL_COMPILE);
    glPushName(name);
    if(manifold_dimension()<2)
        glCallList(name);//wire is the only possible mode to draw.
    else
        shade(span_length);
    glPopName();
glEndList();
return name;
}
```

8.3 オブジェクトの選択処理

(1) ヒットレコードの作成

1. 1のように作成されたディスプレーリストを使用し、OpenGL のセレクションとフィードバックの機能を用いて、MGOpenGLView::pick_to_select_buf()はオブジェクトの選択(ピック)処理を実現している。ここではピック検知されたオブジェクトのディスプレーリスト名(ヒットレコード)が selectBuf に返却され、関数の戻り値として、オブジェクトの数が返される。

```
//Function's return value is the number of hit records.
int MGOpenGLView::pick_to_select_buf(
int sx, int sy, //Screen coordinates. (left, bottom) is (0,0).
double aperturex, double aperturey,
size_t display_list,      //display list of a group that includes pick objects.
size_t buf_size, GLuint* selectBuf//selection buffer and the size.
){
glSelectBuffer(buf_size,selectBuf);
GLint viewport[4];
glGetIntegerv(GL_VIEWPORT, viewport);
glRenderMode(GL_SELECT);

glInitNames();
glMatrixMode(GL_PROJECTION);
glPushMatrix();
glLoadIdentity();
gluPickMatrix((GLdouble)sx,(GLdouble)sy,aperturex,aperturey, viewport);
set_frustum();//Set the same transformation matorix to the one to display objects.

glMatrixMode(GL_MODELVIEW);
glPushMatrix();
      set_model_matrix();//set_model_matrix
      //Draw objects to pick.
      glCallList(display_list);
glPopMatrix();

glMatrixMode(GL_PROJECTION);
glPopMatrix();
return glRenderMode(GL_RENDER);
}
```

（２）　ヒットレコードの構造

OpenGL により作成されるヒットレコードの構造は次の通りである：

0	ヒットしたときのネームスタック上のネームの数 m
1	視体積と交差したプリミティブのＺ座標値の最大値
2	視体積と交差したプリミティブのＺ座標値の最小値
3	所属するグループのリスト名０
・	所属するグループのリスト名 1
	・ ・ ・
	MGObject/MGFace のリスト名 m-1

このレコードが何個バッファに格納されたかが、pick_to_select_buf の関数戻り値として返却される。

8.4 MGOpenGLView::drawScene()の表示処理

```
  //1. 描画基本環境設定
  glClearColor(Bcolr[0], Bcolr[1], Bcolr[2], Bcolr[3]);//背景色の指定
  glClearDepth(1.0);                                    //デプスバッファ値の指定
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);   //背景色、デプスバッファのクリア
  glEnable(GL_DEPTH_TEST);                              //デプステストの有効化
  glDepthFunc(GL_LEQUAL);                               //デプステスト式の指定

  //2. 射影行列の設定
  glMatrixMode(GL_PROJECTION);glLoadIdentity();
  if(m_perspective) glFrustum(left,right,bottom,top,znear,zfar); //透視投影
  else glOrtho(left,right,bottom,top,znear,zfar); //平行投影

  //3. モデルビュー行列の設定
  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
                                                        glGetDoublev(GL_MODELVIEW_MATRIX,
currentMat);                                            //現在のモデルビュー行列の取得
                                                        glLoadIdentity();  //モデルビュー
行列の初期化
  gluLookAt(eye[0],eye[1],eye[2],0.,0.,0., up[0],up[1],up[2]);
                                                          if(zebra){//ゼブラマッピングがある
時
       glDisable(GL_COLOR_MATERIAL);                    // COLOR_MATERIALの抑止
  glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
  glTexGenfv(GL_S, GL_EYE_PLANE, fvPlane);
                                                          }
  glMultMatrixd(currentMat);
                                                        glTranslated(-cntr[0], -cntr[1],
-cntr[2]);                                              //ボックス枠センターへの移動

  //4. 制御平面（方眼紙）の描画
                                                        m_cplane.draw();   //制御面(方眼
紙)の描画

  //5. モデルの表示
                                                        glColor4fv(m_Gcolor);  //オブジェ
クトカラーの指定
                                                        glCallList(m_display_list); //描
画用ディスプレーリストの実行

  //6. 一時表示オブジェクトの表示
  m_sysgllist.draw_list();      // 仮表示データの表示(ゼブラ表示を含む)

  //7.強調表示オブジェクトの表示
```

```
  // COLOR_MATERIALの抑止
```

示色の指定

```
は強調表示データ)
  glEnable(GL_DEPTH_TEST);
  glPopMatrix();
  glDisable(GL_DEPTH_TEST);
glDisable(GL_COLOR_MATERIAL);

glColor4fv(m_Hcolor);  //強調表

glCallList(nm);    //強調表示(nm
```