付録 B. ライブラリ関数

第一版　　平成１０年３月
第二版　　平成２２年３月

はじめに

第４章で解説したように、景観シミュレーションのシステムは、大きく見て、８のライブラリと、ダイアログ毎の８０程度のハンドラ・ルーチンがその大半を占めている。前者は、開発初期の頃から一貫したデータ構造を処理するために維持・改良されてきたものである。可搬性が高いＡＮＳＩ－Ｃ言語をベースとして記述されている。初期の頃と比較すると、データ規模も大きくなってきており、またマシンのメモリ容量、ハードディスク容量、処理速度などは比較にならない程進歩したが、基本的な構造は何も変わっていない。

一方、後者のダイアログ・ハンドラは、操作性等のインターフェースに係る現場のニーズや、新たなＯＳや開発環境に適応しつつ、柔軟に機能追加や改良を行った結果、現状に至っている。その多くは、C++で書かれ、開発環境(Wizard 等)が自動的に生成するソースコードや、開発者向けのサンプルから引用した部分も含んでいる。

長期的に見ると、知的財産としてのソフトウェアの価値の大きな部分は、いわばインフラストラクチャとしてのライブラリ関数に存在する。一方、ユーザーから見える部分に相当するダイアログ・ハンドラは、将来の OS やニーズの変化に伴って、今後も更に柔軟に修正を求められる可能性があり、より鮮度が求められる消耗品的な性格が強い。

この付録Ｂでは、個々のライブラリ関数に関して解説する。

ライブラリ関数はそれぞれの目的別に分かれていて、ＳＭＬ（シーン管理ライブラリ)、ＤＭＬ（データ管理ライブラリ)、ＤＢＩＬ（景観データベース・インターフェース・ライブラリ)、Ｇ３ＤＲＬ（レンダリングライブラリ)、ＩＰ（インタプリタライブラリ)、Ｕ３（ユーティリティライブラリ)、ＥＮＶ（環境設定ライブラリ)、Ｚ３ＥＲＲ（メッセージライブラリ）がある。

ＳＭＬ　：　シーンデータを管理するライブラリ
ＤＭＬ　：　ジオメトリデータを管理するライブラリ
ＤＢＩＬ　：　景観データベースを管理するライブラリ
Ｇ３ＤＲＬ　：　データの描画に関するライブラリ
ＩＰ　：　インタプリタのライブラリ（ファイルの入出力）
Ｕ３　：　ユーティリティ関連で、計算等のライブラリ
ＥＮＶ　：　システムの環境を管理するライブラリ
Ｚ３ＥＲＲ　：　メッセージウインドウを表示させるライブラリ

最も素朴なシステム構成であるビューワにおいては、以下のような処理の流れとなる。

①最初に以下の関数を始めに１度だけ呼び出す。

```
e3Initialize();
s3Initialize();
d3Initialize();
g3Initialize();
dbInitialize(NULL);
z3LoadMessage();
dbSetDB(0);
dbOpen(0, NULL);
dbSetDB(1);
dbOpen(1, NULL);
dbSetDB(2);
dbOpen(2, NULL);
dbSetDB(3);
dbOpen(3, NULL);
```

②プログラム終了時には以下の関数を呼び出す。

```
dbSetDB(0);
dbClose();
dbSetDB(1);
dbClose();
dbSetDB(2);
dbClose();
dbSetDB(3);
dbClose();
dbExit();
z3CloseMessage();
```

③あるシーンを表示したい場合は、先ずシーンファイルをロードし、シーン配列のアドレスを取得する。

```
s3Scene **scn_array;
int scn_num = dbLoadScene("sample", &scn_array);
```

④経年変化情報をセットするには以下のようにする。

```
dbChangeInfo info;
s3Time *t = s3GetSceneTime(scn_array[0]);
s3GetTimeParam(t, &info.date);
```

```
    dbSetChangeInfo(&info);
```
⑤次にマテリアル及びテクスチャをロードする。
```
    g3LoadMaterialAll();
    g3LoadTextureAll();
```
⑥最後にシーンを描画する。
```
    g3AssignDrawarea((void*)&m_hdc);
    g3SetScene(scn_array[0]);
    wg3Redraw((void*)&m_hdc);
```

初版では、関数の仕様に、構想段階のものがあり、実際の関数の動作とは異なっているものがあり、第二版では訂正した。また、初版では、メモリ管理の一貫性がまだ不十分であり、終了時のメモリ解放処理が不足している部分があったが、これを補った。

アプリケーションライブラリに関しては、本文で解説した通り、ライブラリ関数とダイアログ・ハンドラの中間を媒介する機能として、その後多くの関数が作成されたため、これらを追記すると共に、編成を改めた。

目次（付録B）

Ｂ－１．シーン管理ライブラリ（ＳＭＬ）

シーン管理ライブラリ（ＳＭＬ）とは、ＬＳＳ-Ｓファイルの内容をメモリ上に管理するものである。

（１）システム関数

char *s3Version();
ＳＭＬのバージョンを取得する。
返り値 バージョンを示す文字列

int s3Initialize();
ＳＭＬを初期化する。
返り値 成功時１　エラー時０

（２）データ構築関数

s3Scene *s3CreateScene(char *name, int type);
シーンを生成する
name シーン名称
type シーンタイプ
返り値 メモリ・ブロックのアドレス　エラー時 NULL

s3Image *s3CreateImage(char *name, char *filename);
イメージを読み込む
name イメージ名称
filename イメージファイル名称
返り値 メモリ・ブロックのアドレス　エラー時 NULL
s3Image には既にデータが格納される。以下の通り。

```
typedef struct _s3Image {
 char *name;
 char *filename;
 int width;
 int height;
 int format;
 int type;
 void *pixels;
} s3Image;
```

width～pixels はＯｐｅｎＧＬの glDrawPixels のパラメータである。

イメージファイルを読み込む為にＤＢＩＬの dbLoadImage を用いる

s3Model *s3CreateModel(char *name, char *filename);
LSS-G ファイルをロードし、モデルを作成する。
name モデル名称
filename 形状ファイル名称
返り値 メモリ・ブロックのアドレス　エラー時 NULL
s3Model には既にデータが格納されている。以下の通り。

```
typedef struct _s3Model {
  char *name;
  char *filename;
  d3Group **root; /* array of root groups */
  int num;/* numbers of root groups */
} s3Model;
```

num は通常 1 である。
形状ファイルを読み込む為にＤＢＩＬの dbLoadGeometry を用いる

s3Camera *s3CreateCamera(char *name);
視点データを生成する。
name 視点名称
filename 形状ファイル名称
返り値 メモリ・ブロックのアドレス　エラー時 NULL

s3Light *s3CreateLight(char *name, int type);
光源ユニットを生成する。
name 光源名称
type 光源タイプ
返り値 メモリ・ブロックのアドレス　エラー時 NULL

s3LightGroup *s3CreateLightGroup(char *name);
光源グループを生成する。
name 光源グループ名称
返り値 メモリ・ブロックのアドレス　エラー時 NULL

s3Effect *s3CreateEffect(char *name, int type);
効果ユニットを生成する。

name 効果ユニット名称
type 効果タイプ
返り値 メモリ・ブロックのアドレス　エラー時 NULL

s3EffectGroup *s3CreateEffectGroup(char *name);
効果グループを生成する。
name 効果グループ名称
返り値 メモリ・ブロックのアドレス　エラー時 NULL

s3Time *s3CreateTime(char *name);
時間を生成する。
name 時間名称
返り値 メモリ・ブロックのアドレス　エラー時 NULL

void s3InitSceneName();
ユニークなシーン名称作成機能を初期化する。

const char *s3NewSceneName();
ユニークなシーン名称を作成する。
返り値 メモリ・ブロックのアドレス
新たに割り当てたメモリ・ブロック上に「SCNnnn」(nnn は整数値) という形式の文字列を生成し、これを全てのシーンの名称と比較する。もし一致するシーンが無ければ、このメモリ・ブロックを返す。一致するものがあれば、整数値を一つ増加させ同じ比較を行う。整数値はスタティック変数に保持され、次に関数が呼び出された時は前回よりも一つ大きな値から出発する。s3InitSceneName 関数により、スタティック変数を１にリセットする。

void s3InitImageName();
ユニークなイメージ名称作成機能を初期化する。

const char *s3NewImageName();
ユニークなイメージ名称を作成する。
返り値 メモリ・ブロックのアドレス

void s3InitModelName();
ユニークなモデル名称作成機能を初期化する。

const char *s3NewModelName();
ユニークなモデル名称を作成する。
返り値 メモリ・ブロックのアドレス

void s3InitCameraName();
ユニークなカメラ名称作成機能を初期化する。

const char* s3NewCameraName();
ユニークなカメラ名称を作成する。
返り値 カメラ名称のアドレス

void s3InitLightGroupName();
光源グループ名称の初期化

const char *s3NewLightGroupName();
ユニークな光源グループ名称を作成する。
返り値 メモリ・ブロックのアドレス。

void s3InitLightName();
ユニークな光源ユニット名称作成機能を初期化する。

const char *s3NewLightName();
ユニークな光源ユニット名称を作成する。
返り値　メモリ・ブロックのアドレス　エラー時 NULL

void s3InitEffectGroupName();
ユニークな効果グループ名称作成機能を初期化する。

const char *s3NewEffectGroupName();
ユニークな効果グループ名称の名前を作成する。
返り値 メモリ・ブロックのアドレス

void s3InitEffectName();
ユニークな効果ユニット名称作成機能を初期化する。

const char *s3NewEffectName();

ユニークな効果ユニット名称を作成する。

void s3InitTimeName();
ユニークな時間名称の作成機能を初期化する

const char *s3NewTimeName();
ユニークな時間名称を作成する。
返り値 メモリ・ブロックのアドレス

（３）データ定義/更新関数

void s3SetSceneImageBack(s3Scene *scn, s3Image *img);
シーンに背景イメージをセットする。
scn 対象となるシーンのアドレス
img セットするイメージのアドレス
イメージは１つだけセットできる

void s3SetSceneImageFront(s3Scene *scn, s3Image *img);
シーンに前景イメージをセットする。
scn 対象となるシーンのアドレス
img イメージのアドレス
イメージは１つだけセットできる

void s3SetSceneModel(s3Scene *scn, s3Model *mdl);
シーンにモデルをセットする。
scn シーンのアドレス
mdl モデルのアドレス
モデルは１つだけセットできる

void s3SetSceneCamera(s3Scene *scn, s3Camera *cam);
シーンに視点をセットする。
scn シーンのアドレス
cam 視点のアドレス
視点は１つだけをセットできる

void s3SetSceneLightGroup(s3Scene *scn, s3LightGroup *lg);
シーンに光源グループをセットする。

scn シーンのアドレス
lg 光源グループのアドレス
光源グループは１つだけセットできる

void s3SetSceneEffectGroup(s3Scene *scn, s3EffectGroup *eg);
シーンに効果グループをセットする。
scn シーンのアドレス
eg 効果グループのアドレス
効果グループは１つだけセットできる

void s3SetSceneTime(s3Scene *scn, s3Time *t);
シーンに時間をセットする。
scn シーンのアドレス
t 時間のアドレス

void s3SetCameraParam(s3Camera *cam,double eyex, double eyey, double eyez,
  double centerx, double centery, double centerz,double upx, double upy, double upz,
  double fovy, double aspect, double zNear, double zFar);

s3Camera 構造体にパラメータをセットする。
cam カメラ情報を格納するアドレス
eyex,eyey,eyez　視点座標
centerx,centery,centerz　注視点座標
upx,upy,upz アップベクトル（カメラのレンズ軸回りの傾きを示す）
（以上９変数は OpenGL の gluLookAt 関数の引数に対応する）
fovy　垂直方向の視野角（度）
aspect　画面の横サイズ／縦サイズ
zNear　表示前後範囲の下限距離
zFar　表示前後範囲の上限距離
（以上４変数は OpenGL の gluPerspective 関数の引数に対応する）

void s3SetLightGroupLight(s3LightGroup *lg, s3Light *lit);
光源グループに光源ユニットをセットする。
lg 光源グループのアドレス
lit　光源ユニットのアドレス
光源ユニットは複数個セットできる

void s3SetLightColor(s3Light *lit, float r, float g, float b);
光源ユニットに色をセットする。
lit 対象となる光源ユニットのアドレス
r,g,b 色 (0.0≦≦1.0)

void s3SetLightPosition(s3Light *lit,float x, float y, float z, float w);
光源に位置をセットする
lit 光源ユニットのアドレス
x,y,z 座標値
w 光源の種類 (0.0 平行光源　1.0 点光源)

void s3SetEffectGroupEffect(s3EffectGroup *eg, s3Effect *ef);
効果グループに効果ユニットをセットする
eg 効果グループのアドレス
ef 効果ユニットのアドレス
効果ユニットは複数個セットできる

void s3SetEffectParam(e3Effect *e, int i, char *param);
効果ユニットにパラメータをセットする
e 対象とする効果ユニット
i セットするパラメータの番号
param セットするパラメータ (文字列) の内容
(すでにパラメータがセットされていた場合には、これを解放する。引数で示したパラメータのコピー(メモリ・ブロック)を作成し、セットする。)

void s3SetTimeParam(s3Time *t, float date);
時間に日数をセットする。
t 時間のアドレス
date 日数 (不動小数) で記述した時間

(4) データ取得関数÷

int s3GetSceneNum();
シーンの個数を得る。
返り値 シーンの総数

s3Scene *s3GetScene(int index);
一つのシーンを取得する。
index シーンの通し番号
返り値 メモリ・ブロックのアドレス　エラー時 NULL (index の値が範囲外)

```
全てのシーンを得る方法：
n = s3GetSceneNum();
for (i = 0; i < n; ++i) {
  scn = s3GetScene(i);
   [scn に対する処理]
}
```

s3Image *s3GetSceneImageBack(s3Scene *scn);
シーンの背景イメージを取得する。
scn シーンのアドレス
返り値 メモリ・ブロックのアドレス　エラー時 NULL

s3Image *s3GetSceneImageFront(s3Scene *scn);
シーンの前景イメージを取得する。
scn シーンのアドレス
返り値 前景イメージのアドレス

s3Model *s3GetSceneModel(s3Scene *scn);
シーンのモデルを取得する。
scn シーンのアドレス
返り値 モデルのアドレス

s3Camera *s3GetSceneCamera(s3Scene *scn);
シーンのカメラ情報を取得する。
scn シーンのアドレス
返り値 カメラ情報 (構造体) のアドレス

s3LightGroup *s3GetSceneLightGroup(s3Scene *scn);
シーンの光源グループを取得する。
scn シーンのアドレス
返り値 光源グループのアドレス

s3EffectGroup *s3GetSceneEffectGroup(s3Scene *scn);
シーンの効果グループを取得する。
scn シーンのアドレス
返り値 効果グループのアドレス

s3Time *s3GetSceneTime(s3Scene *scn);
シーンの時間を取得する。
scn シーンのアドレス
返り値　時間構造体のアドレス

void s3GetCameraParam(s3Camera *cam,double *eyex, double *eyey, double *eyez, double *centerx, double *centery, double,*centerz, double *upx, double *upy, double *upz, double *fovy, double *aspect,double *zNear, double *zFar);
視点のパラメータを s3Camera 構造体から取得し、各構成要素に分解する。
cam カメラ情報のアドレス
eyex,eyey,eyez　視点座標
centerx,centery,centerz　注視点座標
upx,upy,upz アップベクトル（カメラのレンズ軸回りの傾きを示す）
（以上９変数は OpenGL の gluLookAt 関数の引数に対応する）
fovy　垂直方向の視野角（度）
aspect　画面の横サイズ／縦サイズ
zNear　表示前後範囲の下限距離
zFar　表示前後範囲の上限距離
（以上４変数は OpenGL の gluPerspective 関数の引数に対応する）

int s3GetLightGroupLightNum(s3LightGroup *lg);
光源グループに割り当てられた光源ユニットの個数を取得する。
lg 光源グループのアドレス
返り値 光源ユニット個数

s3Light *s3GetLightGroupLight(s3LightGroup *lg, int index);
光源グループの光源ユニットを取得する。
lg 光源グループのアドレス
index 光源グループにおける光源ユニットの通し番号
返り値 光源ユニットのアドレス エラー時 NULL(index が範囲外)

光源グループの全ての光源ユニットを得る方法：

```
n = s3GetLightGroupLightNum(lg);
for (i = 0; i < n; ++i) {
  lit = s3GetLightGroupLight(lg, i);
   [lit に対する処理]
}
```

void s3GetLightColor(s3Light *lit,float *r, float *g, float *b);
光源ユニットの色を取得する。
lit 光源ユニットのアドレス
r,g,b 色の格納先アドレス

void s3GetLightPosition(s3Light *lit, float *x, float *y, float *z, float *w);
光源ユニットの位置を取得する。
lit 光源ユニットのアドレス
x,y,z,w のアドレス

int s3GetEffectGroupEffectNum(s3EffectGroup *eg);
効果グループに割り当てられた効果ユニットの個数を取得する。
eg 効果グループのアドレス
返り値 効果ユニット個数

s3Effect *s3GetEffectGroupEffect(s3EffectGroup *eg, int index);
効果グループから一つの効果ユニットを取得する。
eg 効果グループのアドレス
index 効果ユニットの通し番号
返り値 効果ユニットのアドレス エラー時 NULL(index が範囲外)

効果グループの全ての効果ユニットを得る方法：

```
n = s3GetEffectGroupEffectNum(eg);
for (i = 0; i < n; ++i) {
  ef = s3GetEffectGroupEffect(eg, i);
}
```

void s3GetEffectParam(s3Effect *e, int index);
効果ユニットのパラメータを取得する
e 効果グループのアドレス
index 求めるパラメータの通し番号

void s3GetTimeParam(s3Time *t, float *date);
時間（構造体）から時間（日数）を取得する。
t 時間（構造体）のアドレス（取得先）
date 時間のアドレスの格納先

int s3SearchLightGroupLightName(s3LightGroup *lg, char *name);
光源グループ内で、指定した名称と同名の光源ユニットを検索する。
lg 光源グループのアドレス
name 光源ユニット名称
返り値 一致した光源ユニットの通し番号 ないとき－1

int s3SearchEffectGroupEffectName(s3EffectGroup *eg, char *name);
効果グループ内で、指定した名称と同名の効果ユニットを検索する。
eg 効果グループのアドレス
name 効果ユニット名称
返り値 一致した光源ユニットの通し番号 ないとき－1

int s3IsSameScene(const char *name);
同じシーン名称がすでにあるかチェックする。
name シーン名称
返り値 1:ある 0:ない

int s3IsSameImage(int scnno, const char *name);
同じイメージ名称がすでにあるかチェックする。
scnno 検査から外すシーンの番号
name イメージ名称
返り値 1:ある 0:ない

int s3IsSameModel(int scnno, const char *name);
同じ名称のモデルがすでにあるかチェックする。
scnno 検査から外すシーンの番号
name モデル名称
返り値 1:ある 0:ない

int s3IsSameCamera(int scnno, const char *name);

同じ名称のカメラ情報がすでにあるかチェックする。
scnno 検査から外すシーンの番号
name カメラ名称
返り値 1:ある 0:ない

int s3IsSameLightGroup(int scnno, const char *name);
同じ光源グループ名称がすでにあるかチェックする。
scnno 検査から外すシーンの番号
name 光源グループ名称
返り値 1:ある 0:ない

int s3IsSameEffectGroup(int scnno, const char *name);
同じ効果グループ名称がすでにあるかチェックする。
scnno 検査から外すシーンの番号
name 効果グループ名称
返り値 1:ある 0:ない

int s3IsSameLight(const char *name);
同じ光源ユニット名称がすでにあるかチェックする。
name 光源ユニット名称
返り値 1:ある 0:ない

int s3IsSameEffect(const char *name);
同じ効果ユニット名称がすでにあるかチェックする。
name 効果ユニット名称
返り値 1:ある 0:ない

int s3IsSameTime(int scnno, const char *name);
同じ時間名称がすでにあるかチェックする。
scnno 検査から外すシーンの番号
name 時間名称
返り値 1:ある 0:ない

（５）データ削除関数

void s3DeleteScene(s3Scene *scn);
シーンを削除する。

scn シーンのアドレス

void s3DeleteImage(s3Image *img);
イメージを削除する。
img イメージのアドレス

void s3DeleteModel(s3Model *mdl);
モデルを削除する。
mdl モデルのアドレス

void s3DeleteCamera(s3Camera *cam);
視点を削除する。
cam カメラ情報のアドレス

void s3DeleteLightGroup(s3LightGroup *lg);
光源グループを削除する。
他のグループに登録されていない光源ユニットも削除する
lg 光源グループのアドレス

void s3DeleteLightGroup2(s3LightGroup *lg);
光源グループをシーンから登録抹消する。ただし、光源ユニットは削除しない。
lg 光源グループのアドレス

void s3DeleteLightGroupLight(s3LightGroup *lg, int index);
光源グループの光源ユニットを削除する。
lg 光源グループのアドレス
index lg 内の光源ユニットの通し番号

void s3DeleteLightGroupLight2(s3LightGroup *lg, int index);
光源グループの光源ユニットの登録を解除する。
lg 光源グループのアドレス
index lg 内の光源ユニットの通し番号

void s3DeleteLightGroupLight2(s3LightGroup *lg, int index);
光源グループの光源ユニットを登録抹消する。
lg 光源グループのアドレス

index 光源グループ内の光源ユニットの通し番号
（光源ユニットのメモリ・ブロック自体は残す）

void s3DeleteLight(s3Light *lit);
光源ユニットを削除する。
lit 光源ユニットのアドレス

void s3DeleteEffectGroup(s3EffectGroup *eg);
効果グループを削除する。配下の効果ユニットに関して、他の効果グループからの重複利用状況を検査し、兼業していない効果ユニットだけ削除する。
eg 効果グループのアドレス

void s3DeleteEffectGroup2(s3EffectGroup *eg);
効果グループを削除する。ただし、配下の効果ユニットは削除しない。
eg 効果グループのアドレス

void s3DeleteEffectGroupEffect(s3EffectGroup *eg, int index);
効果グループの通し番号で指定した効果ユニットを削除する。
eg 効果グループのアドレス
index 処理対象とする eg 配列内の効果ユニットの番号

void s3DeleteEffectGroupEffect2(s3EffectGroup *eg, int index);
効果グループの効果ユニットを登録解除する。効果ユニット自体は存置する。
eg 効果グループのアドレス
index 処理対象とする eg 配列内の効果ユニットの番号

void s3DeleteEffect(s3Effect *ef);
効果ユニット(メモリ・ブロック)を解放する。
ef 効果ユニットのアドレス

void s3DeleteTime(s3Time *t);
時間を削除する。
t 時間のアドレス

void s3AllClear();
全てのシーンを削除する。

Ｂ－２．データ管理ライブラリ（ＤＭＬ）

データ管理ライブラリ（ＤＭＬ）は、地物をモデリングするためのデータ構造（ＬＳＳ-Ｇファイルとして保存される）をメモリ上で管理する。基本的な単位は、グループであり、その配下に面、線、頂点から構成される幾何学的要素、カラー、マテリアル、テクスチャから成る表面仕上げを扱う。更に、外部ファイル参照や、地面、住宅物件、物理・化学的属性等、任意に追加可能な属性データを処理するための基本的機能を提供している。

ＤＭＬライブラリにおいてじゃ、d3db[]というスタティック変数(配列)を起点としたグループのテーブルを管理しており、ここに全てのグループを登録する。全てのグループを対象とした検索等の処理は、このテーブルを用いるのが速い。G3DRL ライブラリが管理する表示処理においても、グループ間のリンク構造を辿って全てのグループにアクセスする処理を行っているが、表示のためのリンクでは、一つの子グループが複数の親を有することができる構造となっているため、同じグループが異なる文脈で複数回アクセスされる。

グループのテーブル d3db の実体は、DML ライブラリの中にスタティック変数として定義された d3DB 構造体の配列である。一つの構造体は、グループのリストと、グループ総数を管理している。グループのリストには、グループへのポインタと削除フラグがあり、処理速度が求められる削除処理に当たって、このフラグを用いて実際のグループに関連したメモリ・ブロックの解放処理（リンク構造の確認を含むやや複雑な処理）を先送りすることができる。

（１）システム関数

char *d3Version();
ＤＭＬのバージョンを取得する
返り値 バージョンを示す文字列

int d3SetDB(int no);

グループのテーブル番号をセット（選択）する。デフォルトは０である。このコマンド以降に実行される関数は、選択されたテーブル、即ち一つの d3DB 構造体に対して適用される。Ver.2.09 の基幹部分(sim.exe)においては、一つのテーブルしか使用していない。
no テーブル番号 (0≦≦9)
返り値 成功値 1 エラー時 0

int d3GetDB();
グループのテーブル番号を取得する。
no テーブル番号 (0≦≦9)
返り値 データベース番号

int d3Initialize();

現在選択されているグループのテーブルに登録されている全てのグループとこれらに関連したメモリ・ブロックを解放し、テーブルを初期化する。
返り値 成功値1 エラー時0

void d3Bebas();
現在選択されているグループのテーブルに登録されている全てのグループと、これらに関連する各種メモリ・ブロックを解放する。終了時に呼び出す。

（2）データ構築関数

d3Group *d3CreateGroup(char *name, int type, char *kind);
グループを生成する。
name グループ名称 （ユニークな名前）
type グループタイプ
kind グループ種別
返り値 グループ(メモリ・ブロック)のアドレス, エラー時NULL

d3Link *d3CreateLink(d3Group *parent, d3Group *child);
リンクを定義する
変換マトリクスは単位行列となる
parent 親グループ
child 子グループ
返り値 リンクのアドレス, エラー時NULL

int d3RegisterMaterial(char *name);
マテリアルを登録する
name マテリアル名称
返り値 マテリアルID番号(＞0)
エラー時0
既に同じ名称が存在する場合には、同じID番号が返る

本関数は名称とIDを対応付けするのみであり、他には何も行わない
実際のデータを使用したい場合はMILを用いる

int d3RegisterTexture(char *name);
テクスチャを登録する
name テクスチャ名称

返り値　テクスチャ ID 番号(＞0), エラー時 0
既に同じ名称が存在する場合には、同じ ID 番号が返る

本関数は名称と ID を対応付けするのみであり、他には何も行わない
実際のデータを使用したい場合は TIL を用いる

int d3CalcBoundingBox(d3Group *g);
グループのバウンディングボックスを計算し、セットする
g グループのアドレス
返り値 成功時 1
エラー時 0

int d3CalcBoundingBoxInc(d3Group *g, d3face *f);
面のバウンディングボックスをグループにセットする
g グループのアドレス
f 面のアドレス
返り値 成功時 1
エラー時 0

const char *d3NewGroupName();
ユニークなグループ名称を取得する
返り値 ユニークなグループ名称のアドレス

d3Group *d3GetNewGroupFromPoints(int count, double *points);
頂点配列からの新しいグループ（面を一つもつ）を作成する
count 頂点の数
points 頂点配列（頂点数分）
返り値 新しいグループのアドレス

d3Group *d3BangunFromPoints(int count, double *points, double t);
頂点配列からの新しいグループ（頂点列を下底とし、高さを t とする立体）を作成する。
count 頂点の数
points 頂点配列（頂点数分）
t 高さ
返り値 新しいグループのアドレス

int isconcave(d3Face *f);
面が凹ポリゴンかどうかを検査する。
f 検査対象とする面
返り値　－１（凹）　０（凸）

ketemu(double A0[3], double A1[3], double B0[3], double B1[3]);
線分 A0-A1 と線分 B0-B1 の交差を検査する。
戻り値
　３：通常交差　（通常、０＜戻り値を以て交差と見なす）
　０：同軸上で重複する場合または接続する場合
－１：交差なし（A0 と A1 が B0B1 に関して同じ側）
－２：交差なし（B0 と B1 が A0A1 に関して同じ側）
－１０：同軸上で離れている場合

（３）初期化関数

int d3Initface(d3face *f);
面データを初期化する
f グループ名称　（ユニークな名前）
返り値　成功時 1 エラー時 0
d3SetGroupface をコールする前に使用する

int d3InitVertex(d3Vertex *v, int size);
頂点データを初期化する
v 頂点配列
size 配列のサイズ
返り値　成功時 1 エラー時 0
d3SetGroupface をコールする前に使用する

void d3InitGroupName();
グループ名称作成の初期化

（４）データ定義/更新関数

int d3SetGroupName(d3Group *g, char *name);
グループに名称を付加する
g グループのアドレス
name グループ名称

返り値 成功時1 エラー時0

int d3SetGroupMaterial(d3Group *g, int id);
グループにマテリアル属性を付加する
g グループのアドレス
id マテリアルID番号
返り値 成功時1 エラー時0

int d3SetGroupTexture(d3Group *g, int id);
グループにテクスチャ属性を付加する
g グループのアドレス
id テクスチャID番号
返り値 成功時1 エラー時0

int d3SetLinkMatrix(d3Link *l, d3Matrix mat, int mode);
リンクに変換マトリクスを定義する
l リンクのアドレス
mat 変換マトリクス
mode 変換モード
D3_MM_LOAD　－　初期化する
D3_MM_PRE　－　前から乗ずる
D3_MM_POST　－　後から乗ずる
返り値 成功時1 エラー時0
変換マトリクスの要素は以下の順序で格納されるものとする
0　4　8　12
1　5　9　13
2　6　10　14
3　7　11　15
変換されるベクトルは列（column）ベクトルとする

d3face *d3SetGroupface(d3Group *g, d3face *f, d3Vertex *v);
グループに面を割り当てる
g グループのアドレス
f 面のアドレス
ｖ 頂点配列（頂点数分がある）
返り値 割り当てられた面のアドレス

f、v は内容が関数内バッファへコピーされる
f は d3Initface にて初期化しておくこと
v は d3InitVertex にて初期化しておくこと
f で指定すべきメンバは以下の通り
 unsigned long va_f (面の属性フラグ：D3_VA_***のビット OR で指定)
 unsigned long va_v (頂点の属性フラグ：D3_VA_***のビット OR で指定)
 int vnum(頂点数)
 int material(マテリアル ID)
 int texture(テクスチャ ID)
 float n[3](法線ベクトル、正規化必要)
 float c[4](色、0.0≦ ≦1.0)
v で指定すべきメンバは以下の通り
 unsigned long va(各頂点の属性フラグ：D3_VA_***のビット OR で指定)
 double v[3] (座標)
 float n[3] (法線ベクトル)
 float t[2] (テクスチャ座標)
 flaot c[4] (色)

d3Face *d3SetGroupFaceConcave(d3Group *g, d3Face *f, d3Vertex *v);
グループにコンケーブ属性を付けて面を割り当てる
g グループのアドレス
f 面のアドレス
v 頂点配列 (頂点数がある)
返り値 割り当てられた面のアドレス

int d3SetGroupUserInfo(d3Group *g, void *u, int no);
グループに任意の情報を割り当てる
g グループのアドレス
u 任意情報のアドレス
no 情報番号 (0≦ ≦9)
返り値 成功時 1 エラー時 0
u は別ライブラリにより管理される

int d3SetGroupDispInfo(d3Group *g, void *d);
グループに表示情報を割り当てる
g グループのアドレス

d 表示情報のアドレス
返り値 成功時 1 エラー時 0
d は別ライブラリにより管理される

int d3SetMaterialName(int id, char *name);
指定された ID のマテリアルを変更する
id マテリアル ID
name マテリアル名称
返り値 成功時 1 エラー時 0
指定された ID は既に d3RegisterMaterial によって取得されていなければならない

int d3SetMaterialInfo(int id, void *info);
マテリアルに任意の情報を割り当てる
id マテリアル ID
info 任意情報のアドレス
返り値 成功時 1 エラー時 0

int d3SetMaterialTexture(int id, int id_tex);
マテリアルにテクスチャＩＤを割り当てる
id マテリアル ID
id_tex テクスチャ ID
返り値 成功時 1 エラー時 0
本関数は、マテリアルにテクスチャが付随しているような場合に利用できる

int d3SetTextureName(int id, char *name);
指定されたＩＤのテクスチャを変更する
id テクスチャ ID
name テクスチャ名称
返り値 成功時 1 エラー時 0
指定された id は既に d3RegisterTexture によって取得されていなければならない

int d3SetTextureInfo(int id, void *info);
テクスチャに任意の情報を割り当てる
id テクスチャ ID
info 任意情報のアドレス
返り値 成功時 1 エラー時 0

int d3AddTravMatrixToTailLink(int travLevel, d3TravInfo *trav, d3Matrix m);
トラバースの最後のリンクマトリクスにマトリクスを乗じる。
tranLevel トラバースのレベル
trav トラバース情報のアドレス
m マトリクス
返り値 成功時1 エラー時0

int d3AddMatrixToTailLink(d3PickPath *path, d3Matrix m);
選択したオブジェクトのリンク連鎖の最低レベルのリンクマトリクスにマトリクスを乗じる。
path 選択
m マトリクス
返り値 成功時1 エラー時0

int d3UVMatrix(double ori[3], double u[3], double v[3], double uscale, double vscale, d3Matrix m);
テクスチャ座標を計算するためのＵＶマトリクスを、パラメータ群から生成する。
ori 原点
u U軸ベクトル
v V軸ベクトル
uscale Ｕ方向スケール
vscale Ｖ方向スケール
m 算出されたマトリックスの格納先
返り値　1

int d3GetUV1x1Frame(d3Matrix uvMat, double *vertices);
テクスチャ座標を計算するためのＵＶマトリクスに対応する、１×１の大きさの正方形の４頂点の座標を計算する。
uvMat ＵＶマトリクス
vertices 算出した４頂点の格納先
返り値　1

void d3SetFaceTextureCoord(d3Matrix uvMat, d3Matrix pathMat, d3Face *f);
面にテクスチャ座標をセットする
uvMat テクスチャのＵＶマトリクス

pathMat パスマトリクス
f 面のアドレス

void d3ClearFaceTextureCoord(d3Face *f);
面のテクスチャ座標を削除する
f 面のアドレス

void d3SetGroupTextureCoord(d3Matrix uvMat, d3Matrix pathMat, d3Group *g);
面にテクスチャ座標をセットする
uvMat テクスチャのＵＶマトリクス
pathMat パスマトリクス
g グループのアドレス

void d3ClearGroupTextureCoord(d3Group *g);
グループのテクスチャ座標を削除する
g グループのアドレス

（５）データ取得関数÷

int d3GetGroupNum();
グループ数を得る
返り値 グループ数

int d3GetGroup(d3GGFunc func);
グループを全て得る
func グループ毎に呼び出される関数
返り値 正常終了時１ 途中中断時０

d3GGFunc は以下のように定義される
typedef int (*d3GGFunc)(d3Group *g);
g 現在のグループのアドレス
返り値　正常終了時１ 途中中断要求時０

int d3GetRootGroup(d3GGFunc func);
ルートグループを全て得る
func ルートグループ遭遇毎に呼び出される関数
返り値 正常終了時１ 途中中断時０

int d3TraverseGroup(d3Group *g, d3TGFunc func);
グループの木構造をトラバースする
深さ優先探索で行う
g トラバースルートグループのアドレス
func トラバース毎に呼び出される関数
返り値 正常終了時1 途中中断時0
d3TGFuncは以下のように定義される

typedef int (*d3TGFunc)(d3Group *g, d3Matrix mat);
g　現在のグループのアドレス
mat　現在の変換マトリクス
返り値 正常終了時1 途中中断要求時0

int d3TraverseGroupRec(d3Group *g, d3TGFunc func, d3Matrix mp, d3Matrix ml);
グループの木構造をトラバースする
g グループのアドレス
func トラバース毎に呼び出される関数
mp マトリクス
ml マトリクス
返り値 端終了時1
途中中断時0

int d3TraverseGroup2(d3Group *g, d3TGFunc func);
グループの木構造をトラバースする
深さ優先検索で行う
g トラバースルートグループのアドレス
func トラバース毎に呼び出される関数
返り値 正常終了時1 途中中断時0

int d3TraverseGroupRec2(d3Group *g, d3TGFunc func, d3Matrix mp, d3Matrix ml);
グループの木構造をトラバースする
g グループのアドレス
func トラバース毎に呼び出される関数
mp マトリクス
ml マトリクス

返り値 端終了時 1 途中中断時 0

void d3GetTravMatrix(int travLevel, d3TravInfo *trav, d3Matrix m);
指定したトラバースまでのマトリクスを取得する
travLevel トラバースのレベル
trav トラバース情報のアドレス
m マトリクス

void d3GetPathMatrix(d3PickPath *path, d3Matrix m);
ピッキングパスまでのマトリクスを取得する
path ピッキングのアドレス
m マトリクス

void d3GetLayoutMatrix(double trn[3], double rot[3], double scl[3], double center[3], d3Matrix layMat);
配置操作におけるスケール、回転、平行移動の操作を合成したマトリックスを計算する。
trn 平行移動
rot 回転
scl スケール
center 拡大縮小・回転の中心
layMat 結果を格納するマトリックス (配列)
void d3GetLocalLayoutMatrix(d3Matrix pathMat, d3Matrix worldLayMat, d3Matrix localMat);
ワールド座標系で表現された配置マトリックス（移動・回転・スケール）と同じ結果を、ローカル座標系において実現するために、配置オブジェクトに対して適用するマトリックスを計算する。
pathmat ローカル座標系を記述するマトリクス
worldLayMat ワールド座標系で表現した配置マトリクス
localMat 算出したローカル座標系で表現した配置マトリクスの格納先

int d3GetGroupMaterial(d3Group *g);
グループを指定してマテリアルを得る
g グループのアドレス
返り値 マテリアル ID 番号

char *d3GetGroupName(d3Group *g);

グループを指定して名称を得る
g グループのアドレス
返り値 グループ名称

int d3FilingFiles(FILE *fp);
ファイル属性を有するグループのリストを出力する。
fp 出力先のファイル
戻り値 ファイル数

int d3GetGroupTexture(d3Group *g);
グループを指定してテクスチャを得る
g グループのアドレス
返り値 テクスチャ ID 番号

d3Link *d3GetGroupULink(d3Group *g, d3Link *l_prev);
グループを指定して上位のリンクを得る
g グループのアドレス
l_prev 直前のリンクのアドレス
NULL は先頭からの意
返り値 リンクのアドレス
存在しない場合 NULL
全てのリンクを得るには以下のようにする。

```
d3Link *l;
for (l = 0; l = d3GetGroupULink(g, l); ) {
          ...
}
```

d3Link *d3GetGroupDLink(d3Group *g, d3Link *l_prev);
グループを指定して下位のリンクを得る
g グループのアドレス
l_prev 直前のリンクのアドレス
NULL は先頭からの意
返り値 リンクのアドレス 存在しない場合 NULL

全てのリンクを得るには以下のようにする

```
d3Link *l;
```

```
for (l = 0; l = d3GetGroupDLink(g, l); ) {
        ***
}
```

```
int d3GetBoundingBox(d3Group *g, double *xmin, double *xmax, double *ymin, double
*ymax,double *zmin, double *zmax);
```

グループのバウンディングボックスを得る
g グループのアドレス
xmin バウンディングボックスの最小 x 座標のアドレス
xmax バウンディングボックスの最大 x 座標のアドレス
ymin バウンディングボックスの最小 y 座標のアドレス
ymax バウンディングボックスの最大 y 座標のアドレス
zmin バウンディングボックスの最小 z 座標のアドレス
zmax バウンディングボックスの最大 z 座標のアドレス
返り値 成功時 1 エラー時 0

```
int d3GetLinkMatrix(d3Link *l, d3Matrix mat);
```

リンクを指定して変換マトリクスを得る
l リンクボックス
mat 変換マトリクス
返り値 成功時 1 エラー時 0

```
d3Group *d3GetLinkUGroup(d3Link *l);
```

リンクを指定して上位のグループを得る
l リンクのアドレス
返り値 グループのアドレス エラー時 NULL

```
d3Group *d3GetLinkDGroup(d3Link *l);
```

リンクを指定して下位のグループを得る
l リンクのアドレス
返り値 グループのアドレス エラー時 NULL

```
d3face *d3GetGroupface(d3Group *g, d3face *f_prev);
```

グループを指定して面を得る
g グループのアドレス
f_prev 直前の面のアドレス

NULL は先頭からの意
返り値 面のアドレス 存在しない時 NULL
全ての面を得るには以下のようにする

```
d3face *f;
for (f = 0; f = d3GetGroupface(g, f); ) {
    ・・・・・・・・・処理・・・・・・・・・・
}
```

```
d3Vertex *d3GetfaceVertex(d3face *f);
```
面データから頂点データを得る
f 面のアドレス
返り値 頂点配列（頂点数分がある） エラー時 NULL

```
void *d3GetGroupDispInfo(d3Group *g);
```
グループの表示情報を得る
g グループのアドレス
返り値 表示情報のアドレス

```
int d3Gwaktu(float tim);
```
時間をパラメータとする外部関数の形状を再生成する。
tim 新たな時間
返り値　再生成の対象となったグループの数

```
void *d3GetGroupUserInfo(d3Group *g, int no);
```
グループの属性情報を得る。
g グループのアドレス
no 属性番号（0:ファイル属性 1:その他の任意属性）
返り値 属性情報のアドレス

```
int d3GetMaterialNum();
```
登録されたマテリアル数を取得する
#0 も登録と見なす
返り値 マテリアル数

```
char *d3GetMaterialName(int id);
```

マテリアル名称を取得する
id マテリアル ID 番号
返り値 マテリアル名称

void d3DaftarMaterialName(FILE *fp);
使用されているマテリアル名称の一覧をファイルに出力する。
fp 出力するファイル

void *d3GetMaterialInfo(int id);
マテリアルの任意情報を取得する
id マテリアル ID 番号
返り値 任意情報のアドレス

int d3GetMaterialTexture(int id);
マテリアルに割り当てられたテクスチャＩＤを取得する
id マテリアル ID 番号
返り値 テクスチャ ID　割り当てられていない場合 0 が返る
※本関数は、マテリアルにテクスチャが付随しているような場合に利用できる

int d3GetTextureNum();
登録されたテクスチャ数を取得する
#0 も登録と見なす
返り値 テクスチャ数

char *d3GetTextureName(int id);
テクスチャ名称を取得する
id テクスチャ ID 番号
返り値 テクスチャ名称

void *d3GetTextureInfo(int id);
テクスチャの任意情報を取得する
id テクスチャ ID 番号
返り値 任意情報のアドレス

d3Group *d3SearchGroup(const char *gname);
グループ名でグループを検索する

gname グループ名のアドレス
返り値 発見時グループのアドレス　未発見時 NULL

int d3GetTravInfo(d3TravInfo **trav);
トラバースの情報を取得する
trav トラバース情報のアドレスのアドレス
返り値 トラバース情報の数

void d3PrintMatrix(d3Matrix m);
マトリクスをコンソールにプリントする
m マトリクス

void d3PrintPointd(double *p);
頂点をコンソールにプリントする
p 頂点(x、y、z)

void d3PrintPointf(float *p);
頂点をコンソールにプリントする
p 頂点(x、y、z)

int d3FindNormal(d3face *f, double *n);
面の法線のベクトルを取得する
f 面のアドレス
n 法線のアドレス
返り値 成功時 1　エラー時 0

int d3FindNormalf(d3face *f, float *n);
面の法線のベクトルを取得する
f 面のアドレス
n 法線のアドレス
返り値 成功時 1　エラー時 0

int d3CalcNormal(double p[][3], int n, double *v);
頂点から法線ベクトルを求める
p 頂点配列
n 頂点数

v 法線ベクトル（単位ベクトル）
返り値 成功時１　エラー時０

int d3CalcNormalf(double p[][3], int n, float *v);
頂点から法線ベクトルを求める
p 頂点配列
n 頂点数
v 法線ベクトル（単位ベクトル）
返り値 成功時１　エラー時０

（６）データ削除関数

int d3DeleteGroup(d3Group *g);
グループを消去する
g グループのアドレス
返り値 成功時１　エラー時０
本関数は指定されたグループの上位のリンクを削除する
更に下位のリンクが無ければグループそのものも削除する

int d3DeleteGroupAll(d3Group *g);
グループを削除する
g グループのアドレス
返り値 成功時１　エラー時０
本関数は以下のものを削除する
　指定されたグループの上位リンク
　指定されたグループ
　指定されたグループの下位リンク及びグループ全て

int d3DeleteGroupCepathabis(d3group *g);
グループとその配下のグループを全て削除する。他のグループからのリンクの検査を省略し、迅速に処理する。ルートグループ以下全てのグループを省略する場合に使用する。
g 削除するリンク連鎖の頂上のグループ
戻り値　1

int d3DeleteLink(d3Link *l);
リンクを削除する
l リンクのアドレス

返り値 成功時１　エラー時０

int d3DeleteGroupBebas(d3Group *g);
グループとその配下のグループを削除する。但し、子グループが別の親グループからもリンクされている場合には、その子グループは存置する。
g 削除するグループ
返り値　１

int d3DeleteGroupFace(d3Group *g, d3Face *f);
グループの面を消去する
g グループのアドレス
f 面のアドレス
返り値 成功時１　エラー時０

int d3DeleteGroupFaceAll(d3Group *g);
グループの面を全て消去する
g グループのアドレス
返り値 成功時１　エラー時０

（７）データ複写関数

d3Group *d3CopyGroup(d3Group *g);
グループをコピーする
g グループのアドレス
返り値 グループのアドレス

（８）ピッキング関数

int d3PickGroup(float sx, float sy, d3Group *root, d3Matrix projmat, d3Matrix viewmat,
  d3Matrix r2wmat, d3PickInfo *pi, int pi_num);
スクリーン座標からグループなどを得る
sx スクリーンｘ座標（-1.0≦≦1.0）
sy スクリーンｙ座標（-1.0≦≦1.0）
root ルートグループのアドレス
projmat 投影変換マトリクス
viewmat 視野変換マトリクス
r2wmat ルートグループ座標系からワールド座標系への変換マトリクス
pi ピッキング情報が格納される配列

pi_num pi の配列数
返り値 ピッキング情報数
ピッキング情報は以下のメンバが利用できる
 d3Group *g（ピッキングされたグループのアドレス）
 d3face *f（ピッキングされた面のアドレス）
 double p[3]（ピッキングされた面上の座標値）
ピッキング情報は距離の短い順にソートされる
即ち pi[0]が最も前にある

void d3PickBebas();
ピッキング情報を解放する。
int d3PickSegment(float sx, float sy, d3Group *root, d3Matrix projmat, d3Matrix viewmat, d3Matrix r2wmat, d3PickInfo *pi, int pi_max);
スクリーン座標からグループなどを得る
sx スクリーン x 座標（-1.0≦≦1.0）
sy スクリーン y 座標（-1.0≦≦1.0）
root ルートグループのアドレス
projmat 投影変換マトリクス
viewmat 視野変換マトリクス
r2wmat ルートグループ座標系からワールド座標系への変換マトリクス
pi ピッキング情報が格納される配列
pi_num pi の配列数
返り値 ピッキング情報数

int d3GetPickPath(d3PickPath **path);
ルートグループからピッキングされたグループまでのパスを取得する
path ピッキングパスのアドレスのアドレス
返り値 ピッキング数

int d3PickLine2Cube(double *p, double *d, double xmin, double xmax, double ymin,
  double ymax, double zmin, double zmax, double q[][3], int *status);
直方体と直線の交点を求める
p 直線上の点
d 直線の方向ベクトル
x,y,zmax 直方体を定義する最大値
x,y,zmin 直方体を定義する最小値

q 交点座標
status 各面（６面）の交点の有無　1:交点あり　0:交点なし
返り値 ピッキング情報が格納される配列
pi_num pi の配列数
返り値　　1:交点あり　0:交点なし

int d3PickLine2Patch(double *p, double *d, double *v0, double *v1, double *v2, double *n, int kind, double *q);
面と直線の交点を求める
p 直線上の点
d 直線の方向ベクトル
v0,v1v,2 面の３頂点
n 面の法線ベクトル
kind 1:平行四辺形
0:三角形
q 交点座標
返り値　　1:交点あり　0:交点なし

（９）問い合わせ関数

int d3IsGroupFace(d3Group *g);
グループに面があるか調べる
g グループのアドレス
返り値　　面がある場合１　面がない場合０

int d3IsSameGroup(const char *name);
同じグループ名称がすでにあるかチェックする
name グループ名称
返り値 1:ある　0:ない

const char* d3NewGroupName();
まだ使用されていないグループ名称を作成する（「GRP[番号]」の形式）
返り値：メモリ・ブロックのアドレス

（１０）マトリクス/ベクトル関数

void d3IdentMatrix(d3Matrix m);
与えられた行列を単位行列にする

m 行列

void d3CopyMatrix(d3Matrix m1, d3Matrix m2);
行列 m2 を行列 m1 にコピーする(m1=m2)
m1 行列(to)
m2 行列(flom)

void d3MultMatrix(d3Matrix m1, d3Matrix m2, d3Matrix m3);
行列 m2 と m3 を掛け合わせ、結果 m1 に返す(m1=m2*m3)
m1 行列 (結果)
m2 行列 (前から掛ける)
m3 行列 (後ろから掛ける)

void d3TransposeMatrix(d3Matrix m1, d3Matrix m2);
行列 m2 の転置行列を求め、結果 m1 に返す(m1=m2T
m1 行列 (結果)
m2 行列(from)

int d3InverseMatrix(d3Matrix m1, d3Matrix m2);
行列 m2 の逆行列を求め、結果を m1 に返す(m1=m2-1
m1 行列 (結果)
m2 行列(from)

void d3TransformPointd(double *p1, double *p2, d3Matrix m);
点を変換する
p1 変換された点 (３次元)
p2 変換される点 (３次元)
m 変換マトリクス

void d3TransformPointf(float *p1, float *p2, d3Matrix m);
点を変換する
p1 変換された点 (３次元)
p2 変換される点 (３次元)
m 変換マトリクス

void d3TransformVectord(double *v1, double *v2, d3Matrix m);

ベクトルを変換する
v1 変換されたベクトル（３次元）
v2 変換されるベクトル（３次元）
m 変換マトリクス

void d3TransformVectorf(float *v1, float *v2, d3Matrix m);
ベクトルを変換する
v1 変換されたベクトル（３次元）
v2 変換されるベクトル（３次元）
m 変換マトリクス

int d3TransformNormald(double *n1, double *n2, d3Matrix m);
法線を変換する
n1 変換された法線（３次元）
n2 変換されれる法線（３次元）
m 変換マトリクス

int d3TransformNormalf(float *n1, float *n2, d3Matrix m);
法線を変換する
n1 変換された法線（３次元）
n2 変換されれる法線（３次元）
m 変換マトリクス

double d3DotProductd(double *v1, double *v2);
ベクトルの内積を計算する
v1 ベクトル１（３次元）
v2 ベクトル２（３次元）
返り値 内積

float d3DotProductf(float *v1, float *v2);
ベクトルの内積を計算する
v1 ベクトル１
v2 ベクトル２
返り値 内積

void d3CrossProductd(double *v1, double *v2, double *v3);

ベクトルの外積を計算する (v1=v2×v3)
v1 ベクトル (結果)
v2 ベクトル (３次元)
v3 ベクトル (３次元)

void d3CrossProductf(float *v1, float *v2, float *v3);
ベクトルの外積を計算する(v1=v2×v3)
v1 ベクトル (結果)
v2 ベクトル (３次元)
v3 ベクトル (３次元)

int d3UnitVectord(double *v1, double *v2);
単位ベクトルを計算する
v1 ベクトル (結果)
v2 ベクトル (３次元)
返り値　成功時１　エラー時０

int d3UnitVectorf(float *v1, float *v2);
単位ベクトルを計算する
v1 ベクトル (結果)
v2 ベクトル (３次元)
返り値　成功時１　エラー時０

double d3VectorLengthd(double *v);
ベクトルの長さを計算する
v ベクトル (３次元)
返り値 長さ

float d3VectorLengthf(float *v);
ベクトルの長さを計算する
v ベクトル (３次元)
返り値 長さ

void d3Translated(d3Matrix m, double x, double y, double z);
平行移動のマトリクスを得る
m 変換マトリクス

x 移動 x 成分
y 移動 y 成分
z 移動 z 成分

void d3Translatef(d3Matrix m, float x, float y, float z);
平行移動のマトリクスを得る
m 変換マトリクス
x 移動 x 成分
y 移動 y 成分
z 移動 z 成分

void d3Rotated(d3Matrix m, double angle, double x, double y, double z);
回転のマトリクスを得る
m 変換マトリクス
angle 回転角(degree)
x 回転軸 x 成分
y 回転軸 y 成分
z 回転軸 z 成分

void d3Rotatef(d3Matrix m, float angle, float x, float y, float z);
回転のマトリクスを得る
m 変換マトリクス
angle 回転角 (degree)
x 回転軸 x 成分
y 回転軸 y 成分
z 回転軸 z 成分

void d3Scaled(d3Matrix m, double x, double y, double z);
スケールのマトリクスを得る
m 変換マトリクス
x スケール x 成分
y スケール y 成分
z スケール z 成分

void d3Scalef(d3Matrix m, float x, float y, float z);
スケールのマトリクスを得る

m 変換マトリクス
x スケール x 成分
y スケール y 成分
z スケール z 成分

void expo(double A[3], double B[3], double H[3]);
Ｈ＝Ａ×Ｂ

void inpo(double A[3], double B[3])
返り値　Ａ・Ｂ

（１１）メモリ関数

void *d3Calloc(size_t nobj, size_t size);
malloc に準ずる
但し NULL は返らない

void *d3Malloc(size_t size);
malloc に準ずる
但し NULL は返らない

void *d3Realloc(void *p, size_t size);
realloc に準ずる
但し NULL は返らない

void d3Free(void *p);
メモリ・ブロックを解放する。free 関数を使用するが、デバッグモードにおいては、メモリ・ブロックのリストを作成し、終了時点で検出されるメモリリークの発生源を特定するための情報を提供する。
p メモリ・ブロックのアドレス

void d3Frei(void **p);
メモリ・ブロックを指すポインタが NULL でなければ、これを解放すると共に、ポインタに NULL を代入する。
p　メモリ・ブロックのアドレス。

char *d3Strdup(char *s);

文字列のコピーを作成する。元の文字列が NULL の場合には、エラー表示し、エラーを意味する文字列を生成する。
s コピー元の文字列のアドレス
返り値　成功時　コピー文字列(メモリ・ブロック)のアドレス
　　失敗時 NULL(メモリ不足)　または、エラーを明示する文字列（引数不適）

int d3GetJumlah();
使用中のメモリ・ブロックの総数を返す。
返り値　メモリ・ブロック総数

void d3ResetJumlah();
メモリ・ブロック数のカウントをゼロにリセットする。

Ｂ－３．景観データベースＩＦライブラリ（ＤＢＩＬ）

景観データベースに関する各種操作を行うために、CDataBase クラス(database.cpp)を支える機能を果たしている他、景観シミュレータで使用する外部ファイルの入出力処理機能を担っている。景観データベースにアクセスする際は、景観データベースＩＦライブラリ（ＤＢＩＬ）によりアクセスすることにより一貫性が保持される。

主に、３種類の景観データベース検索機能(yuu.exe, kou.exe 及び zai.exe)と、編集エディタ(editor.exe)で用い、データベースの登録内容は、外部ファイル com.txt の形で保存される。この外部ファイルに直接アクセスすることのない基幹部分(sim.exe)においても、モデルや画像やマテリアル情報を格納した各種の外部ファイルアクセスに際して、ＤＢＩＬライブラリの諸関数を用いている。

データベースは、ＤＢＩＬライブラリのスタティック配列 DBtab[]を基点として管理されている。DBtab は、dbDataBase 構造体の配列となっており、一つの要素が一つのデータベース（構成は異なるものであってよい）となっている。

ひとつの dbDataBase 構造体は、中心となる rechead メンバーの他に、外部ファイル名、操作履歴などに関する情報が登録されている。

rechead メンバーは、dbRecordHead 構造体へのポインタの配列となっている。要素の数は、登録物件数に対応しており、dbRecordHead 構造体(メモリ・ブロック)が、一つの建設事業や、景観構成要素などの物件の単位を記述している。

dbRecordHead 構造体には、rec メンバーがあり、これは、dbRecord 構造体の配列となっていて、その長さは登録項目数となっている。この他のメンバーとして、削除や変更を記録するフラグ等がある。

rec 配列の要素である dbRecord 構造体は、その項目に登録されたデータの配列を表す data メンバと、データの個数を表す cnt メンバーから成る。data メンバは、様々のデータ

形式を格納しうる union として定義されており、上記の rec 配列の何番目の項目であるかによって、決定される。

要約すると、DBtab から、求めるデータにアクセスする手順は次のようになる。

DBtab[データベース番号].rechead[物件番号]->rec[項目番号].data[データ番号]->型名

詳細については、dbms.h における構造体定義を参照されたい。

景観データベースの一つの特徴は、ひとつの物件のある項目に関して、複数のデータを登録できる点にある。例えば、Ａという建設事業を担当したコンサルタントが複数者であった場合、「Ａ社とＢ社」のように、ひとつの文字列にまとめることなく、「Ａ社」「Ｂ社」のように複数のデータとして、設計会社の項目の下に無制限個登録できる。

検索に際しては、各項目に関して、候補となる条件を複数設定することができる。例えば、担当者に関して「Ｉ氏」「Ｊ氏」「K氏」を検索条件として設定した上で検索操作を行うと、「G氏」「H氏」「Ｉ氏」のようにこの内の１者でも登録されている物件はヒットする。

検索条件が設定されていない項目については、常にヒットする。全ての検索項目に関してヒットした物件が最終的に検索結果となる。

ＡＮＤ検索においては、前回の検索結果を対象セットとして、その中から新たに設定された条件に合致する物件だけに絞り込みが行われる。また、ＯＲ検索においては、前回検索結果を存置して、これに新たな条件に合致する物件が追加される。

（１）システム関数

```
int dbInitialize(char *path);
```

タイムスタンプを初期化する(2.09 では意味を失っている)。

（２）データ構築関数

```
int dbCreateSelectionList(int itemno, int sort,int *count,  int *datatype, dbData **list);
```

検索条件リストを作成する。

itemno 項目番号

datetype データタイプのアドレス（キーワードとそれ以外で処理が異なる）

list 検索条件に関するデータ配列のアドレス

返り値 成功時１　エラー時０

```
int dbReadSearchedRecord(int recno);
```

検索条件に合致するレコードを一つ読み込む。

recno 現在のレコード番号。これより以降を探す。

返り値 時: 合致する次のレコードの番号　ない場合:-2(レコード末尾)

void dbCutSpace(char *str, char *label);
文字列から余分なスペースを除去する（見出し用文字列作成)。
str 元の文字列
label 新しい文字列

void dbReadDeleteData();
削除したファイル(text.del)を読み込む(非使用)。

void dbReadSortType();
ソートファイル(text.srt)を読み込む(非使用)。

int dbLoadScene(char *name, s3Scene ***scn);
シーンファイル(LSS-S 形式)を読み込む。
name ファイル名
scn シーンのアドレス配列のアドレス
返り値 シーン数

dbImage *dbLoadImage(char *name);
イメージファイルを読み込む。
name 画像ファイル名
返り値 メモリ・ブロックのアドレス

int dbLoadGeometry2(char *name, d3Group ***grp);
ジオメトリファイルを読み込む
name ファイル名
grp グループのアドレス配列ののアドレス
返り値 成功時 1　エラー時 0

dbMaterial *dbLoadMaterial(char *name);
マテリアルを読み込む
name マテリアル名称
返り値 マテリアル情報のアドレス

dbTexture *dbLoadTexture(char *name);
テクスチャを読み込む
name ファイル名

返り値 テクスチャ情報のアドレス

int dbMakeDataBase(char *path);
データベースを作成する
path ファイルパス
返り値 成功時１　エラー時０

int dbMakeClassMenu(char *fname, int itemno);
クラスファイルを読み込み、クラステーブルを作成する
fname クラスファイル名
itemno 項目番号
返り値 成功時１　エラー時０

void dbMakeLongListTBL();
費用／価格情報テーブルを作成する。

void dbAllocItemData(dbRecord *rec);
データ格納領域を確保する
rec レコードのアドレス

unisigned long *longimagedata(char *name,long *width, long *height);
ＳＧＩ形式の画像ファイルを読み込む。
name ファイル名（拡張子.sgi）
width 幅の格納先アドレス
height 高さの格納先アドレス
返り値：画像データ（32ビット×ピクセル数）

unsigned long　*tifimagedata(char *name, long *width, long *height);
ＴＩＦＦ形式の画像ファイルを読み込む。
name ファイル名（拡張子.sgi）
width 幅の格納先アドレス
height 高さの格納先アドレス
返り値：画像データ（32ビット×ピクセル数）

unsigned long　*jpgimagedata(char *name, long *width, long *height);
ＪＰＥＧ形式の画像ファイルを読み込む。

name ファイル名（拡張子.sgi）
width 幅の格納先アドレス
height 高さの格納先アドレス
返り値：画像データ（32ビット×ピクセル数）

unsigned long　*bmpimagedata(char *name, long *width, long *height);
ＢＭＰ形式の画像ファイルを読み込む。
name ファイル名（拡張子.sgi）
width 幅の格納先アドレス
height 高さの格納先アドレス
返り値：画像データ（32ビット×ピクセル数）

（３）初期化関数

char *dbVersion();
ＤＢＩＬのバージョンを取得する。
返り値 バージョンを示す文字列

（４）データ定義/更新関数

int dbSetDB(int no);
使用するデータベースの番号をセットする。
no データベース番号（0≦≦9）
返り値 成功時１　エラー時０

int dbOpen(int type, char *name);
データベースを初期設定する。ロードは行わない。
type 対象となるデータベースの種類
　DB_SCENE 優良事例ＤＢ
　DB_GEOMETRY 景観構成要素ＤＢ
　DB_MATERIAL 景観材料ＤＢ
name データベース名称（関数内では使用していない）
返り値 成功時１　エラー時０
　不適切な type が指定された場合エラーとなる。

void dbClose();
何もしない。

void dbSetChangeInfo(dbChangeInfo *info);
時間をセットする。

dbChangeInfo *dbGetChangeInfo();
時間を取得する
返り値　構造体のアドレス

int dbExit();
何もしない。
返り値　常に１を返す

int dbSetSearchKeywd(int itemno, char *keywd);
特定の項目に検索のためのキーワードを追加する。いくつでも登録できる。
itemno 項目番号
keywd キーワードのアドレス
返り値 成功時１　エラー時０

int dbSetSearchString(int itemno, char *string)
特定の項目に検索のための文字列を追加する。いくつでも登録できる。
itemno 項目番号
string 文字列のアドレス
返り値 成功時１　エラー時０

int dbSetSearchRange(int itemno, dbData *from, dbData *to);
特定の項目に検索のための範囲をセットする。既にセットされている検索条件に上書きされる。
itemno 項目番号
from 範囲の下限
to 範囲の上限
返り値 成功時１　エラー時０

int dbSetSearchClass(int itemno, char *path);
特定の項目に検索のための分類をセットする。既にセットされている分類に上書きされる。
itemno 項目番号
path クラスパス
返り値 成功時１　エラー時０

void dbSetKeyList(char *keywd, char *menustr, int kind, int itemno);
検索のためのキーワードリストにキーワードと表示用文字列を追加する。
keywd キーワード
menustr 表示用文字列
kind 検索種別
 SELECT_KEYLIST 0 キーワード一覧からの選択
 SELECT_INPUT　1 手入力
itemno 項目番号

char *dbSetTextData(int recno, int mode);
物件の全項目を文字列として整形出力する。
recno レコード番号
mode モード
 DB_SAVE 0 ファイル保存用
 DB_DISP 1 文字情報画面出力用
返り値 整形した文字列(メモリ・ブロック)のアドレス

void dbSetHistory(char *str, int cnt, int *no);
検索履歴を追記する。DBtab[DBno].history[]の str, cnt,no[0-cnt]メンバーに書き込み、
カウンターhiscnt をインクリメントする。
str 検索条件を表現した文字列
cut 検索結果レコード数
no 検索結果レコード番号を格納した配列

void dbAllSpecifykeywd(char *keywd);
全てのレコードにキーワードを指定する
keywd キーワード

void dbSpecifykeywd(int recno, char *keywd);
一つのレコード（物件）にキーワードを設定する。
recno レコードの通し番号
keywd キーワード

void dbSetKeywd(dbStringData *str, char *keywd);
キーワードを、文字データ構造体のキーワードメンバ（配列）の末尾に追加し、カウンタ

をインクリメントする。
str 文字データ構造体のアドレス
keywd キーワード

void dbSave();
データベースをファイルにセーブする。

void dbSetModifyTime(int recno, int itemno)
変更した日時をセットする。対象とするレコードの対象項目の最初のデータに時間形式で書き込む。
recno 変更時間をセットするレコードの通し番号
itemno 変更時間をセットする項目番号

void dbWriteDeleteData();
削除データをファイル(text.del)に書き込む。

void dbWriteSortType();
ソートファイル(text.srt)に書き込む。

void dbSetDeleteData(char *data);
削除データをセットする。DBtab[DBno].deldata[]の末尾に追記し、delcnt をインクリメントする。

char *dbCutSpace2(char *data);
タブ、スペース、改行を取り除く。
data 文字列
返り値 整形後の文字列

char *dbCutSpace3(char *data);
タブ、スペースを取り除く。
data 文字列
返り値 編集後の文字列

char *dbCutBackSlush(char *data);
「¥」を取る
data 文字列

返り値 編集後の文字列

char *dbSetSpace(char *data);
文字列にスペースを入れて１行の文字数を調整する。
data 文字列
返り値 編集後の文字列

int dbSaveScene(s3Scene **scn, int num, char *name);
シーンをセーブするような関数名だが、現在何も処理しない。
scn シーンのアドレス配列のアドレス
num シーン数
name 保存に用いるファイル名
返り値 常に１

int dbSaveGeometry(d3Group **grp, int num, char *name);
ジオメトリをセーブするような関数名だが、現在何も処理しない。
grp グループのアドレス配列のアドレス
num ルートグループ数
name 保存に用いるファイル名
返り値 常に１を返す

void dbSetTopData(dbData *dbdata, char *data);
代表項目情報をセットする。
dbdata データのアドレス
data 項目タイプ配列

void dbSetColorData(dbData *dbdata, char *data);
色彩情報をセットする。
dbdata データのアドレス
data 項目タイプ配列

void dbSetLoadData(dbData *dbdata, int type, char *data);
読み込みタイプデータをセットする。
dbdata データのアドレス
type データタイプ
data 項目タイプ配列

void dbSetString(char **setbuf, char *data);
文字列をコピーしたメモリ・ブロックのアドレスをセットする。
setbuf 新しいメモリ・ブロックのアドレスの格納先
data セット元文字列のアドレス

void dbSetTitle(int itemno);
項目のタイトルをセットする。
itemno 項目番号

void dbSetRange(dbData *dbdata, int type, char *data);
範囲をセットする。範囲（数値）は、文字列から解析する。
dbdata セット先のデータのアドレス。
type データタイプ（入力値）
data 情報源文字列のアドレス

void dbSetKeywords(dbStringData *str, char *data);
キーワードをセットする。
str 設定先のアドレス
data キーワード文字列のアドレス

void dbSetClassName(dbClass *lev1, dbClass **lev2, int *c2, char *name, dbClass *cls1);
CLS ファイルから、検索のための分類項目のプルダウン・メニュー構成を構築するために、一つの分類名をセットする機能である。lev2 は lev1 のサブメニューに対応する。name 文字列を、lev2 の名称に設定すると共に、lev1 のサブメニュー・リストの末尾に、lev2 を追加する。
lev1 セットするクラスのアドレス
ilev2 セットされるクラスのアドレス
c2 階層
name クラス名
cls1 クラス情報のアドレス（単位、規格、基準に関する情報の取得先）

void SaveImageData(char *name, long xsize, long ysize, long zsize, unsigned long *pixels);
画像をイメージファイル（ＳＧＩ形式）に保存する。

name ファイル名
xsize,ysize,zsize イメージサイズ
pixels ピクセルデータのアドレス

（５）データ取得関数

dbDataBase *dbGetDataBaseAddr();
システムのデータベース配列のアドレスを得る。
返り値　スタティック配列のアドレス

int dbGetDB();
現在アクティブなデータベースの番号を得る。
返り値 データベース番号（0≦≦9）

dbData *dbGetData(int recno, int itemno, int datano, int *datatype);
データを取得する。
recno レコード番号
itemno 項目番号
datano データ番号
datatype データタイプのアドレス
返り値 成功時:データのアドレス　エラー時:NULL

dbClass *dbGetClass(int itemno);
データベースの検索条件として、指定した項目に設定されたクラス（分類）を取得する。
itemno 項目番号
返り値 クラスのアドレス

char *dbGetModifyData(int recno, char *m_type);
修正履歴リストに表示する文字列を取得する。
recno レコード番号
m_type 修正タイプ文字列
返り値 リスト表示文字列

char *dbGetItemDataStr(dbRecordHead *rechead, int itemno, int datano, char *format);
項目データを取得する。
rechead レコードヘッドのアドレス
itemno 項目番号

datano データ番号
format フォーマット文字列（例：”%d”）
返り値 文字列

char *dbGetTitle(int itemno, int *type);
項目のタイトルを取得する(2.09 では非使用)。
itemno 項目番号
type 項目タイプ
返り値 項目タイトル文字列

int dbGetMaterialFileName(char ***name);
データベースの種類毎のディレクトリにある、マテリアルファイル名の一覧表を作成する。
name マテリアルファイル名配列のアドレス格納先
返り値 マテリアルファイル数

int dbGetMaterialName(char *name, char ***matname);
指定したマテリアルファイルの中に登録されたマテリアル名称を取得し一覧表を作る。
name マテリアルファイル名
matname マテリアル名称配列のアドレス格納先
返り値 マテリアル名称数

int dbGetItems (char *data, char ***items);
文字列をトークン（スペースまたはタブ）で区切られた項目のリストに変換する。
data 解析対象とする入力文字列
items 項目データのアドレスのアドレスのアドレス
返り値 項目データ数

int dbGetItemNo(char *data);
ラベルから、項目番号を取得する。
data 項目タイプ配列
返り値 成功時:項目番号　エラー時:-1

（６）データ削除関数

int dbClear();
DBno で選択されている一つのデータベースをクリアし、検索条件やヒストリーに至るまで、これに係る全てのメモリ･ブロックを解放する。

返り値 常に1

void dbBebas()
変更履歴情報をクリアする。

void dbAllDeleteRecord();
DBnoで選択されているデータベースの全ての登録物件を削除する。

void dbDeleteSelectionList(int count, dbData **list);
検索条件リストを削除する(2.09では非使用)。
count 削除する検索条件の数、即ち配列の長さ
list 検索条件の配列のアドレス

void dbResetSearchKeywd(int itemno, char *keywd);
指定したキーワードをリスト（配列）から削除する(2.09では非使用)。
itemno 項目番号
keywd キーワード

void dbResetSearchItem(int item);
特定の項目の検索情報を削除する。
item 削除対象とする項目番号

void dbResetSearch();
全ての項目の検索情報を削除する。

void dbResetKeyList(int index);
特定のキーワードを削除する。
index 削除するキーワードの通し番号

void dbDeleteKeyList();
全てのキーワードを削除する。

void dbResetHistory(int index);
特定の検索履歴情報を削除する。
index 削除する検索履歴情報の通し番号

void dbResetItemData(dbRecordHead *rechead, int itemno, int index);
ある登録物件の特定項目の特定登録データを削除し、登録解除する。
rechead 一つのレコード（物件）のアドレス
itemno 削除するデータ項目の番号
index 削除するデータの通し番号

void dbResetRecord(int recno);
特定のレコード（物件）を削除し、データベースから登録解除する。
recno レコードの通し番号

void dbDeleteRecord(dbRecordHead *rechead);
レコード（物件）を削除し、関連するメモリ･ブロックを解放する。
rechead レコード（物件）のアドレス

void dbDeleteItemData(dbRecord *rec, int itemno, int no);
ある登録物件の特定項目の特定登録データを削除し、関連するメモリ･ブロックを解放する。
rec レコード（物件）の全項目（配列）のアドレス
itemno 項目番号
no データの通し番号

（7）データ複写関数

void dbCopyRecord(dbRecordHead *motohead, dbRecordHead *sakihead);
レコード（物件）をコピーする。
motohead コピー元のレコード（物件）のアドレス
sakihead コピー先のレコード（物件）のアドレス

（8）比較/検索関数

int dbDataCmp(dbData *a, dbData *b);
データの文字列部分を比較する。
a データのアドレス
b データのアドレス
返り値 strcmp に準ずる

int dbSearch(int mode);
指定したモードで検索を実行する。
mode 検索モード

DB_SR_AND 絞込検索（AND 検索）
DB_SR_OR 補追検索（OR 検索）
返り値 発見数

int dbSearchRecData(dbRecord *rec);
レコード（物件）が検索条件と合致するかどうか検査する。
rec レコードの項目配列のアドレス
返り値 全ての項目が合致:1　それ以外:0
（検索条件に含まれていない項目に関しては合致として扱う）

int dbSearchString(dbRecord *rec, dbSearchData *s);
レコード（物件）の特定項目を文字列検索する。検索データは文字列の配列であり、特定項目に登録された複数データ（文字列）の一つが、検索データの文字列のいずれかと合致すれば合格とする。但し、キーワードが複数設定されている場合には、全てのキーワードがいずれかの登録データと一致するのでなければ不合格とする。
rec レコードの特定項目のアドレス
s 検索データ（文字列等の配列）のアドレス
返り値 成功時１　エラー時０

int dbSearchClass(dbRecord *rec, dbSearchData *s);
レコード（物件）の分類クラスを検索する。
rec レコードの特定項目のアドレス
s 検索データのアドレス
返り値 成功時:1　エラー時:0

int dbSearchLong(dbRecord *rec, dbSearchData *s);
レコードを long 情報検索する。検索条件は一つだけとし、条件に設定された数値範囲（不等式）が成立すれば合格とする。
rec レコードの特定項目のアドレス
s 検索条件のアドレス
返り値 成功時:1　エラー時:0

int dbSearchFloat(dbRecord *rec, dbSearchData *s);
レコードを Float 情報検索する。検索条件は一つだけとし、条件に設定された数値範囲（不等式）が成立すれば合格とする。
rec レコードの特定項目のアドレス

s 検索データのアドレス
返り値 成功時:1　エラー時:0

int dbSearchTime(dbRecord *rec, dbSearchData *s);
レコードを時間情報検索する。検索条件は一つだけとし、条件に設定された期間（不等式）にレコードの全てのデータが合致すれば合格、一つでも期間外なら不合格とする。条件として期間が設定されていなければ(その場合開始年と終了年が共にゼロ)、合格とする。
rec レコードの特定項目のアドレス
s 検索データのアドレス
返り値 成功時:1　エラー時:0

int dbMenuListCmp(dbKeywdList *a, dbKeywdList *b);
キーワードを文字列比較する。
a キーワードのアドレス
b キーワードのアドレス
返り値 strcmp に準ずる

int dbHistCmp(dbHistory *a, dbHistory *b);
検索履歴を比較する。検索履歴オルソトするための比較関数として用いる。
a 検索履歴のアドレス
b 検索履歴のアドレス
返り値 strcmp に準ずる

void dbNameSort();
データベースを「あいうえお」順にソートする。比較関数として dbNameSort 関数を指定する。

int dbNameCmp(dbRecordHead **a, dbRecordHead **b);
二つの物件の名称を比較する。
a レコードヘッド（物件の配列）のアドレス
b レコードヘッド（物件の配列）のアドレス
返り値 strcmp に準ずる

void dbTimeSort();
データベースを時間順にソートする。比較関数として dbTimeSort 関数を指定する。

int dbTimeCmp(dbRecordHead **a, dbRecordHead **b);
時間（rec[5]）を比較する
a レコードヘッド（物件の配列）のアドレス
b レコードヘッド（物件の配列）のアドレス
返り値 strcmp に準ずる

int dbCheckKeywd(dbRecord *rec, char *keywd);
一つのレコードの特定項目のデータ（配列）のいずれかにキーワードが含まれるかチェックする。
rec レコードの特定項目のアドレス
keywd キーワード
返り値 いずれかのデータにある:1　どのデータにもない:0

int dbCheckData(float *min, float *max, char *set);
マテリアルファイルの定義行を解析し、情報を取得する。
min 最小値（日数） の格納先
max 最大値（日数） の格納先
set 定義内容（数値等）の格納先
返り値 min-max あり：1、min-max なし:0、エラー時-1

void dbDaftarTexture(FILE *fp);
現在使用されているテクスチャの一覧表をファイルに出力する。
fp 出力先のファイル

(dbms.c)

void　MaterialInit(dbMaterial *m);
dbMaterial 構造体をデフォルト値で初期化する。

void　SetDiffuse(dbMaterial *m, dbMaterial *s);
マテリアルのカラー情報をコピーする。
m コピー先マテリアルのアドレス
s コピー元マテリアルのアドレス
※diffuse 拡散色(点光源や平行光源に対する拡散反射の色)

void　SetAmbient(dbMaterial *);

マテリアルの拡散色から環境色を自動計算する。
m マテリアルのアドレス
※ambient 環境色（環境光に対する色）

void　SetSpecular(dbMaterial *m, dbMaterial *s;
マテリアルの鏡面色情報をコピーする。
m コピー先マテリアルのアドレス
s コピー元マテリアルのアドレス
※specular 鏡面色

void　SetEmission(dbMaterial *, dbMaterial *);
マテリアルの発光色情報をコピーする。
m コピー先マテリアルのアドレス
s コピー元マテリアルのアドレス
※emission 発光色

int　SetMaterial(dbMaterial *m, dbMaterial *s, char *str);
マテリアルファイルの１行分の解析を行う。
m 設定先マテリアルのアドレス
s コピー元マテリアルのアドレス
str 解析するマテリアルファイルの１行
返り値
DBERROR エラー
ENAK 内容充実
BUSUK 内容腐敗
NO コメント行、空白行
YES マテリアル名称が１カラムから定義された宣言行

void　CheckPrint(void);
現在選択されているデータベースをコンソールにデバッグ出力する(2.09 では非使用)。

Ｂ－４．レンダリングライブラリ（Ｇ３ＤＲＬ）

（１）システム関数

char *g3Version();
G3DRL のバージョンを取得する。
返り値 バージョンを示す文字列

（２）データ構築関数÷

int g3CombineGroup(d3Group *g, float value, float ratio);
グループ内の面を結合する(2.09 では、land.dll　に移管)
g グループのアドレス
value 値
ratio レート
返り値 成功時１　エラー時０

int g3CutGroup(d3Group *g, d3Group *setGrp,
　int count, double *points, int inside);
グループの形状（配下の面）を、面で切断する(2.09 では、land.dll に移管)。
g 処理するグループ（入力）
setGrp 処理結果を格納するグループ（出力）
count 切断面の頂点数
points 切断面座標
inside 内外選択フラグ（0:外側　1:内側）
返り値 成功時１　エラー時０

int g3DivtriGroup(d3Group *g, double area);
グループを三角形分割する(2.09 では、land.dll に移管)。
g グループのアドレス
area 三角形のエリアサイズ
返り値 成功時１　エラー時０

int g3CreateDrawarea();
新たな表示領域を作成する。
返り値 作成した表示エリアの通し番号

int g3sCreateDrawarea();
新たな補助表示領域を作成する。OpenGL 画面で、カラーやテクスチャ等の見出し画像を表示するための領域に用いる。
返り値 作成した表示エリアの通し番号

int g3CreateGroundPixel(double left, double right, double bottom, double top);
地面データのデプスピクセルを作成する。

left, right, bottom, top 地面データの範囲 (X,Y)
返り値 成功時１　エラー時０

int g3CreateGroundPixelByBBox();
バウンディングボックス内の地面データのデプスピクセルを保存する。
返り値 成功時１　エラー時０

int g3AplGroundDepth(g3Ground *grd);
グラウンドのデプスを作成する。
grd グラウンドのアドレス
返り値 成功時１　エラー時０

int g3AplGroundDepthNearFar(g3Ground *grd, double zmin, double zmax);
グラウンドのデプスを NearFar 内で作成する。
grd グラウンドのアドレス
zmin Near
zmax Far
返り値 成功時１　エラー時０

int g3AplEndDepth(g3Ground *grd);
高さを計測するためのデプスバッファに地面の属性のある地物を書き込む。
grd グラウンドのアドレス
返り値 成功時１　エラー時０

int g3AplStartDepth(g3Ground *grd, g3Ground *newgrd, int reverse);
高さを計測するためのデプスバッファを作成する。
grd 既存グラウンドのアドレス
newgrd 生成されているグラウンドのアドレス
reverse 上下の逆転フラグ　　0:高　　1:低
返り値 成功時１　エラー時０

void g3AplAddDepth(g3Face *f);
高さを計測するためのデプスバッファに面を追加記録する。
f 面のアドレス

void g3MakeRasterFont();

ラスターフォントを作成する。画面への寸法線書き込みに使用する。

int g3LoadMaterial(int id);
マテリアルを読み込む。
id マテリアル ID
返り値 成功時 1　エラー時 0

int g3LoadMaterialAll();
マテリアルを全て読み込む。
返り値 成功時 1　エラー時 0

int g3LoadTexture(int id);
テクスチャを読み込む。
id テクスチャ ID
返り値 成功時 1　エラー時 0

int g3LoadTextureAll();
テクスチャを全て読み込む。
返り値 成功時 1　エラー時 0

int g3PlaneMovePoint(d3Group *g, int type, double ext, double p[3],
  double moveP[3], double left, double right, double bottom, double top);
頂点を移動させる(2.09 では、land.dll に移管)。
g グループのアドレス
type 追従パターン
ext 波及範囲
p[3] 移動する点 (X,Y,Z)
moveP[3] 移動する量 (X,Y,Z)
返り値 成功時 1　エラー時 0

int g3HeightMovePoint(d3Group *g, double ext, double p[3], double h, double left,
  double right, double bottom, double top);
頂点を高さ移動させる(2.09 では、land.dll に移管)。
g グループのアドレス
ext 波及範囲
p[3] 移動する点 (X,Y,Z)

h 移動量(z)
left,right,bottom,top エリア
返り値 成功時１　エラー時０

```
int g3Road(
    int pointCount, double *points, double roadWidth,
    double cutHeight,double cutStepWidth, double cutSlope, int cutCount,
    double pileHeight,double pileStepWidth, double pileSlope, int pileCount,
    double cutC, double pileC, float roadColor[4], s3Image *texture,
    float cutColor[4], s3Image *cutTex,
    float cutStepColor[4], s3Image *cutStepTex,
    float pileColor[4], s3Image *pileTex,
    float pileStepColor[4], s3Image *pileStepTex,
    double *cutVolume, double *pileVolume) ;
```

道路法面を作成する(2.09 では、nori.dll に移管)。
pointCount 補間された道路軌跡点数
points 補間された道路軌跡座標(pointCount×3)
roadWidth 道路幅
cutHeight 切土高
cutStepWidth 切土小段幅
cutSlope 切土傾斜 (1:nan)
cutCount 切土小段数
pileHeight 盛土高
pileStepWidth 盛土高小段数
pileSlope 盛土高傾斜 (1:nan)
pileCount 盛土小段数
cutC 切土の土量変化率
pileC 盛土の土量変化率
roadColor[4] 道路色
texture 道路テクスチャ(NULL はテクスチャなし)
cutColor[4] 切土色
cutTex 切土テクスチャ(NULL はテクスチャなし)
cutStepColor[4] 切り土の小段色
cutStepTex 切土のテクスチャ(NULL はテクスチャなし)
pileColor[4] 盛土色
cutStepTex 盛土テクスチャ(NULL はテクスチャなし)

pileStepColor[4] 盛土テクスチャの小段色
pileStepTex 盛土テクスチャの小段テクスチャ
cutVolume 切土量（出力）
pileVolume 盛土量（出力）
返り値 成功時１　エラー時０
※道路軌跡線のループの場合は結果は保証されない

int g3Round(d3Group *g, double radius, int division, int ednum, double *points);
角の丸め処理をする
g グループのアドレス
radius 半径
division 分割数
ednum 頂点数（線データ）
points 角取り処理する辺（線データ）の頂点（X,Y,Z･･･の並び）
返り値 成功時１　エラー時０

void g3StencilPrepare();
ステンシル・バッファを使えるように準備する(2.09 では各関数で直接 gl 関数を使用するため、本関数は非使用)。

void g3StencilStart();
ステンシル・バッファの開始(2.09 では各関数で直接 gl 関数を使用するため、本関数は非使用)。

void g3StencilEnd();
ステンシルの終了(2.09 では各関数で直接 gl 関数を使用するため、本関数は非使用)。

int g3Tunnel(d3Group *g, int tCount,double *points, double *startP, double *endP);
トンネルを作成する(2.09 では、tunnel.dll に移管)。
g グループのアドレス
tCount 断面頂点数
points 断面頂点列（X,Y,Z･･･の並び）
startP 始点（X,Y,Z）
endP 終点（X,Y,Z）
返り値 成功時１　エラー時０

g3TsTop *g3LoadTsf(const char *filename);
テクスチャ自動貼り付け設定ファイルを読み込む。
filename ファイル名
返り値 g3TsTop のアドレス

（３）初期化関数

int g3Initialize();
表示領域登録用配列にメモリ・ブロックを割り当て、初期化する。
返り値 成功時１　エラー時０

int g3InitializeWindow();
表示エリアを初期化する。
返り値 成功時１　エラー時０

int g3sInitializeWindowColorSashArea();
マテリアルを一覧するウインドウを初期化する
返り値 成功時１　エラー時０

int g3sInitializeWindowLightArea();
光源を表示する色球を表示するウインドウを初期化する
返り値 成功時１　エラー時０

int g3sInitializeWindowMaterialArea();
マテリアル・カラーを表示するウインドウを初期化する
返り値 成功時１　エラー時０

int g3sInitializeWindowMaterialArea2();
グラフィックなマテリアル表示を行うウインドウを初期化する
返り値 成功時１　エラー時０

int g3sInitializeWindowPlainMaterialArea();
グラフィックなマテリアル表示を行うウインドウを初期化する
返り値 成功時１　エラー時０

int g3sInitializeWindowsSteelArea();
型鋼を表示するウインドウを初期化する

返り値 成功時１　エラー時０

int g3sInitializeWindowTextureArea();
グラフィックにテクスチャを編集するウインドウを初期化する
返り値 成功時１　エラー時０

（４）データ定義/更新関数

int g3SetDrawarea(int num);
表示エリアを初期化する。
num 表示エリアの通し番号
返り値 成功時１　エラー時０

void g3SetColor(int ix, float r, float g, float b, float a);
表示画面の様々な部分に色設定を行う。
ix 設定対象部分の指定
 G3_COL_CLEAR クリアカラー(0,0,0,1) 地物が無い部分の画面色
 G3_COL_GRID グリッドカラー(1,1,1,1) グリッド（格子）の線の色
 G3_COL_MEASURE メジャーカラー(1,1,1,1) 縮尺の色
 G3_COL_AXIS_X 座標軸ｘ(1,0,0,1) Ｘ軸の色
 G3_COL_AXIS_Y 座標軸ｙ(0,1,0,1) Ｙ軸の色
 G3_COL_AXIS_Z 座標軸ｚ(0,0,1,1) Ｚ軸の色
 G3_COL_ORBITPOINT 軌跡ｘ点(1,1,1,1) 移動経路のコントロールポイント
 G3_COL_ORBITLINE 軌跡線(1,1,1,1) 移動経路の折れ線・曲線
 G3_COL_HILIGHT 強調表示線の色(1,0,0,1) 選択したオブジェクトの識別に用いる
 G3_COL_CAMERAMARK カメラマーク(1,1,1,1) 視点設定におけるカメラのアイコン
（ ）内はデフォルト値
※各 ix で（ ）内はデフォルトカラー値でｒ,ｇ,ｂ,ａ 色の値で 0.0-1.0 の値で
　ある

void g3Enable(int ix);
表示における様々なモードを有効にする。
ix モードの設定対象を指定する
　G3_DRA_TEXTURE テクスチャ表示モード（Enable）
　G3_DRA_ANTIALIAS アンチエリアシング　（Disale）
　G3_DRA_GRID グリッド表示（Disable）、オルソだけ
　G3_DRA_AXIS 座標軸（Disable）

G3_DRA_LIGHT 光源 (Enable)
G3_DRA_BACKIMAGE 背景画像表示 (Enable)
G3_DRA_FRONTIMAGE 前景画像表示 (Enable)
G3_DRA_CAMERAMARK カメラマーク (Disable)、オルソだけ
G3_DRA_MEASURE
メジャー (Disable)、オルソだけ画面の左隅に横方向の縮尺を表示する。
画面の約 10%の切りのよい長さ(1.5,10,50･･･)を線と数字で表示する。
G3_DRA_ORBIT 軌跡 (Disable)、オルソだけ
G3_DRA_HILIGHT 強調表示 (Disable)
G3_DRA_RECT 矩形表示 (Disable)、オルソだけ
G3_DRA_WIRE 線画表示 (Disable=shading)
G3_DRA_VISUAL 可視範囲表示 (Disable)
オルソの平面の場合(G3_CAM_TOP)だけ有意
( ) 内はデフォルト値
※モードが有効になっていても、g3Set 関数で必要な情報が設定されていないと無効

void g3Disable(int ix);
表示モードを無効にする。
ix モードの設定対象の指定
G3_DRA_TEXTURE テクスチャ表示モード (Enable)
G3_DRA_ANTIALIAS アンチエリアシング (Disale)
G3_DRA_GRID グリッド表示 (Disable)、オルソだけ
G3_DRA_AXIS 座標軸 (Disable)
G3_DRA_LIGHT 光源 (Enable)
G3_DRA_BACKIMAGE 背景画像表示 (Enable)
G3_DRA_FRONTIMAGE 前景画像表示 (Enable)
G3_DRA_CAMERAMARK カメラマーク (Disable)、オルソだけ
G3_DRA_MEASURE
メジャー (Disable)、オルソだけ画面の左隅に横方向の縮尺を表示する。
画面の約 10%の切りのよい長さ(1.5,10,50･･･)を線と数字で表示する。
G3_DRA_ORBIT 軌跡 (Disable)、オルソだけ
G3_DRA_HILIGHT 強調表示 (Disable)
G3_DRA_RECT 矩形表示 (Disable)、オルソだけ
G3_DRA_WIRE 線画表示 (Disable=shading)
G3_DRA_VISUAL 可視範囲表示 (Disable)
オルソの平面(G3_CAM_TOP)だけ

（ ）内はデフォルト値
※モードが有効になっていても、g3Set 関数で必要な情報が設定されていないと無効

void g3DrawImage2D(dbImage *img);
イメージを描画する。
img イメージ情報のアドレス

void g3SetCameraOrtho(int kind, g3Ortho *ortho);
オルソビューの視点情報をセットする。
kind
  G3_CAM_FRONT zx 平面：南立面図（正面図）
  G3_CAM_TOP zy 平面：平面図
  G3_CAM_SIDE yz 平面：東立面図（側面図）
  G3_CAM_UTARA -xz 平面：北立面図
  G3_CAM_BARAT -yz 平面：西立面図
ortho オルソ構造体のアドレス
ortho はコピー（メモリ・ブロック）を作成した上で表示画面の属性として管理する。pers と ortho は別個に管理される。

void g3SetCameraPers(s3Camera *pers);
パースビュー（透視図）の視点情報をセットする。
pers パース構造体のアドレス
pers は、コピー（メモリ・ブロック）を作成した上で表示画面の属性として管理する。pers と ortho は別個に管理される。

void g3SetCameraOptimize();
現在のデータの最大最小を調べて最適視点を設定する。

void g3SetCameraChangeOptimize(int kind);
表示画法（パース、各種オルソ）を変更した上で、最適化した視点を設定する。
視点パラメータを計算するのが面倒な場合に使える。
kind
  G3_CAM_FRONT zx 平面：南立面図（正面図）
  G3_CAM_TOP zy 平面：平面図
  G3_CAM_SIDE yz 平面：東立面図（側面図）
  G3_CAM_UTARA -xz 平面：北立面図

G3_CAM_BARAT -yz 平面：西立面図

void g3SetCameraZoom(double rate);
視点をズームさせる。
rate ズーム率（1.0 はズームなし）
透視図の場合は、視点と注視点を結ぶ直線に沿って視点を接近または後退させる。
オルソの場合は、left,right,top,bottom を変更し、表示の範囲と縮尺を変える。

void g3SetCameraHorizontalRound(double rate);
透視図における視点を横方向回転(注視点を中心として視点を回転移動)する。
rate 回転角度（360.0 は１回転して元に戻る）
rate＞0 は右回りを示す

void g3SetCameraVerticalRound(double rate);
透視図における視点の縦方向回転（注視点を中心として視点を回転移動）する。
rate 回転角度（360.0 は１回転して元に戻る）
rate＞0 は右回りを示す

void g3SetCameraHorizontalPutar(double rate);
透視図の注視点を、視点を中心に横方向回転する（パン)。
rate 回転角度（360.0 は１回転して元に戻る）
rate＞0 は右回りを示す

void g3SetCameraVerticalPutar(double rate);
透視図の注視点を、視点を中心に縦方向回転する。
rate 回転角度（360.0 は１回転して元に戻る）
rate＞0 は右回りを示す

void g3SetCameraHorizontalShift(double rate);
視点を横にシフト（平行移動）する。
rate シフト率
=0 はシフトなし
＞0 は右または上シフトである
※透視図の場合は視点、注視点を横方向に、視点と注視点の距離を 1.0 としたときの rate 分シフトする。
※オルソの場合は、left,right 間の距離を 1.0 としたときの rate を指定する 。

void g3SetCameraVerticalShift(double rate);
視点を縦シフトする。
rate シフト率
=0 はシフトなし
＞0 は右または上シフトである
※パースの場合は視点、注視点をアップベクトル方向（縦)、視点と注視点の距離を 1.0 としたときの rate 分シフトする。
※オルソの場合は、top,bottom 間の距離を 1.0 としたときの rate を指定する。

void g3SetLightGroup(s3LightGroup *lg);
光源グループを表示画面に対してセットする。
lg 光源グループのアドレス
※ 光源グループ lg 及びその配下の光源ユニットをコピーしたメモリ･ブロックを作成し、セットする。lg が、既に存在する場合には、これを解放した上でコピーをセットする。但し、既存の lg が、新たにセットしようとする lg と同じものであった場合には、解放せずにコピーをセットする。

void g3SetBackImage(s3Image *img);
背景イメージをセットする。
img イメージ情報のアドレス

void g3SetFrontImage(s3Image *img);
前景イメージをセットする。
img イメージ情報のアドレス

void g3SetRootGroup(int count, d3Group **root);
ルートグループをセットする。
count ルートグループ数
root グループ配列のアドレス

int g3CalcBoundingBox();
登録されているルートグループ全体のバウンディングボックス、即ち地物全体の存在範囲（３軸最小最大値）を計算する。結果は選択されている表示画面の属性(da[cd].bbox)として保存される。
返り値 成功時 1　エラー時 0

※ルートグループ以下が変更された場合（形状変更、配置変更、削除など）は本関数を呼ぶこと。
※バウンディングボックスにより、最適視点設定が可能になる。

void g3SetBoundingBox(double min[3], double max[3]);
バウンディングボックスをセットする。
min[3] 最小値(X,Y,Z)
max[3] 最大値(X,Y,Z)

void g3SetCameramark(s3Camera *cam);
カメラ情報からカメラマークに必要な情報がセットされる
cam カメラ情報のアドレス
※カメラマークは、編集画面上において視点位置を示すための一種のアイコンである。

void g3SetCameramarkFromPers();
パース視点情報からカメラマークに必要な情報がセットされる

int g3SetTime(float tim);
時間をセットする。
tim 日数
返り値 成功時１　エラー時０

void g3SetScene(s3Scene *scn);
指定したシーン構造体から必要な情報を、現在選択されている表示画面にセットする。
scn シーンのアドレス（通常は、シーン配列から選択された１要素）

void g3Light();
現在選択されている表示画面に設定されている光源グループを、OpenGL の関数として表示条件として出力する。光源グループが存在しなければデフォルトの光源をセットする(2.09では非使用)。

int g3SelPoint(float sx, float sy, int seltype);
スクリーン座標上の位置と指定選択方法に基づき、全地物を検査して、結果を格納するg3SelPath 構造体を構築し、表示画面の属性(da[ce].pathList)に格納する。
sx,sy スクリーン座標
seltype　選択方法

G3_SET_GROUP　グループを選択する
G3_SET_FACE　面を選択する
G3_SEL_SAMECOLFACE_IN_GROUP　同色面を選択する
返り値 ヒット数（同じ平面位置に重なり合っている三次元オブジェクト数）
※この関数により作成された情報は、g3GetSelpath 関数により取り出され、利用される。

int g3UpSelPath(g3SelPath *spath);
選択対象のグループのレベルを一つ上(親グループ)に上げる。
spath 選択パス情報のアドレス
返り値　成功時１　エラー時０

int g3DownSelPath(g3SelPath *spath);
選択対象のグループのレベルを一つ下（子グループ）に下げる。
spath 選択パス情報のアドレス
返り値 成功時１　エラー時０

int g3SetSingleHilight(g3SelPath *spath);
１つのオブジェクトをハイライト表示するようにセットする。
spath 選択パス情報のアドレス
返り値 成功時１　エラー時０

int g3AddHilight(g3SelPath *spath);
ハイライト表示するオブジェクトを追加する。
spath 選択パス情報のアドレス
返り値 成功時１　エラー時０

void g3SetSize(int width, int height);
表示エリアのサイズを、表示画面の属性(da[cd].width, height)にセットし、OpenGL の描画条件として設定する。
width 幅
height 高さ
※ユーザーが画面サイズを変更した場合に、OpenGL の表示条件をこれに合わせることにより、適切な表示を行うことができる。

void g3SetGridsize(double interval);
格子間隔をセットする。

interval 格子間隔 (0.0以上、デフォルト1.0

void g3SetTexMatrix(d3Matrix texm);
テクスチャマトリクスをセットする。
texm マトリクス

void g3SetOrbitPointHilight(int ix);
軌跡点のハイライトをセットする。画像視点抽出において、計算誤差の大きかった参照点を表示する場合などに使用する。
ix ハイライト表示する軌跡点の通し番号

void g3SetOrbitPoints(int count, double *points, double pointSize);
軌跡点をセットする。×印で表示される。
count 点数
points 点の座標 (X,Y,Z･･･の並び)
pointSize 軌跡点 (×印) の描画サイズ
pointsはポインタだけが保存される

void g3SetOrbitLine(int count, double *points);
軌跡線をセットする。
count 点数
points 点の座標 (X,Y,Z･･･の並び)
pointsはポインタだけが保存される
※曲線補間がされていること

void g3SetVisualArea(double left, double right, double bottom, double top, s3Image * texture);
可視範囲をセットする。
left,right,bottom,top 可視範囲のワールド座標(x,y)
texture 可視分布のテクスチャデータ
※テクスチャファイル名は不要
※アルファを1.0未満にすると半透明になる
※NULLの場合は、可視範囲が線画で表示される

void g3SetAntiAliasCount(int count);
アンチエリアシングの精度をセットする。

count 精度

int g3CalcGroupBoundingBox(d3Group *g);
グループのバウンディングボックスを計算する。
g グループのアドレス
返り値 成功時１　エラー時０

double g3CalcBoundingSphere(d3Group *group);
バウンディングスフィアーを計算する。
group グループのアドレス
返り値 半径

void g3PrintString(char *s);
文字（ラスターフォント）を出力する。縮尺の表示などに用いる。
s 表示する文字

void g3NormalizeTexture4(int *width, int *height, unsigned long **pixels);
テクスチャをノーマライズする。
width 幅の格納先
height 高さの格納先
pixels 処理結果のピクセル配列の格納先アドレス
※幅と高さを２のべき乗の値にする。

int g3AutoTextureFace(d3Face *f, g3TsDetail *detail, d3Matrix pathMat, double uvorigin[3], double udir[3], double uScale, double vScale);
面にテクスチャを自動貼り付けする。
f 面のアドレス
detail g3TsDetail のアドレス
pathMat パスマトリクス
uvorigin[3] uv 原点 (X,Y,Z)
udir[3] uv ベクトル (X,Y,Z)
uScale u スケール
vScale v スケール
返り値 成功時１　エラー時０

int g3AutoTextureGroup(d3Group *g, g3TsDetail *detail, d3Matrix pathMat,

double uvorigin[3], double udir[3], double uScale, double vScale);
グループにテクスチャを自動貼り付けする。
g グループのアドレス
detail g3TsDetail のアドレス
pathMat パスマトリクス
uvorigin[3] uv 原点 (X,Y,Z)
udir[3] uv ベクトル (X,Y,Z)
uScale u スケール
vScale v スケール
返り値 成功時 1　エラー時 0

void g3SetShadowLength(double len)
影計算において作成する影立体の長さの設定値を設定する。
len 影立体の長さ

void g3SetShadowMode(int mode);
影計算における原因物体設定モードを設定する。
mode 設定モード

```
G3_SHADOW_ALL              1  /* 全ての物体*/
G3_SHADOW_OTHER_GROUND     2  /* 地面以外の物体*/
G3_SHADOW_ONE              3  /* 選択した1つの物体*/
```

void g3SetShadowLightMode(int mode);
影計算における光源の設定モードを設定する。
mode 設定モード

```
G3_SHADOW_OTHERLIGHT_OFF   10  LIGHT0以外の光源をOFF
G3_SHADOW_OTHERLIGHT_ON    11  LIGHT0以外の光源をON
```

（５）データ取得関数

int g3GetDrawable(int ix);
指定した表示モードの有効/無効を取得する。
Ix　取得する表示モード
G3_DRA_TEXTURE テクスチャ表示モード (Enable)
G3_DRA_ANTIALIAS アンチエリアシング　(Disale)
G3_DRA_GRID グリッド表示 (Disable)、オルソだけ
G3_DRA_AXIS 座標軸 (Disable)

G3_DRA_LIGHT 光源 (Enable)
G3_DRA_BACKIMAGE 背景画像表示 (Enable)
G3_DRA_FRONTIMAGE 前景画像表示 (Enable)
G3_DRA_CAMERAMARK カメラマーク (Disable)、オルソだけ
G3_DRA_MEASURE
メジャー (Disable)、オルソだけ画面の左隅に横方向の縮尺を表示する。
画面の約 10%の切りのよい長さ(1.5,10,50･･･)を線と数字で表示する。
G3_DRA_ORBIT 軌跡 (Disable)、オルソだけ
G3_DRA_HILIGHT 強調表示 (Disable)
G3_DRA_RECT 矩形表示 (Disable)、オルソだけ
G3_DRA_WIRE 線画表示 (Disable=shading)
G3_DRA_VISUAL 可視範囲表示 (Disable)
オルソの平面(G3_CAM_TOP)だけ
( ) 内はデフォルト値
モードが有効になっていても、g3Set 関数で必要な情報が設定されていない場合は無視される
返り値 有効 1　無効 0

int g3GetCameraOrtho(int *kind, g3Ortho *ortho);
オルソビューの視点情報を取得する。
kind
G3_CAM_FRONT zx 平面：南立面図（正面図）
G3_CAM_TOP zy 平面：平面図
G3_CAM_SIDE yz 平面：東立面図（側面図）
G3_CAM_UTARA -xz 平面：北立面図
G3_CAM_BARAT -yz 平面：西立面図
ortho オルソのアドレス
返り値 成功時 1　エラー時 0
保存領域をコピーして返すので、ポインタを示すフィールドを直接変更した場合は、Ｓｅｔ関数を呼ぶ必要がある。

int g3GetCameraPers(s3Camera *pers);
透視図画面のカメラ情報を取得する。
pers カメラ情報の格納先
返り値 成功時 1　エラー時 0
構造体の内容を格納先にコピーするので、内容を変更した結果を表示に反映させるために

は、g3SetCameraPers 関数を用いる必要がある。

int g3GetBackImage(s3Image **img);
背景イメージを取得する。
img イメージ情報のアドレスを格納するアドレス
返り値 成功時 1　エラー時 0

int g3GetFrontImage(s3Image **img);
前景イメージを取得する。
img イメージ情報のアドレスを格納するアドレス
返り値 成功時 1　エラー時 0

int g3GetRootGroup(int *count, d3Group ***root);
ルートグループを取得する。
count ルートグループ数の格納先
root グループ配列のアドレスの格納先
返り値 成功時 1　エラー時 0

int g3GetBoundingBox(double min[3], double max[3]);
グループの属性として記録されているバウンディングボックスを取得する。
min[3] 最小値(X,Y,Z)
max[3] 最大値(X,Y,Z)
返り値 成功時 1　エラー時 0

int g3GetGroupLocalCenter(d3Group *g, double center[3]);
グループのバウンディングボックスの中心（ローカル座標）を取得する。
g グループのアドレス
center[3] 中心(X,Y,Z)
返り値 成功時 1　エラー時 0

int g3GetGroupWorldCenter(g3SelPath *spath, double center[3]);
グループのバウンディングボックスの中心（ワールド座標）を取得する。
spath 選択パス情報のアドレス
center[3] 中心(X,Y,Z)
返り値 成功時 1　エラー時 0

int g3GetTime(float *tim);
時間を取得する。
tim 時間（日数を浮動小数で表現）を格納するアドレス
返り値 成功時１　エラー時０

int g3GetSelPath(int num, g3SelPath **spath);
選択パス情報を取得する。
num セレクト番号(g3SelPoint の返り値以内の数)を指定
spath 選択パス情報のアドレスの格納先
返り値 成功時１　エラー時０
※セレクト番号は、同じ位置に重なっている複数のオブジェクトのいずれかを選択する。

int g3GetHilight();
ハイライト表示されているオブジェクトの数を取得する。
返り値 ハイライト数(da[cd].hilightCount)

int g3GetSize(int *width, int *height);
表示エリアのサイズを取得する。
width 幅
height 高さ
返り値 成功時１　エラー時０

int g3GetGridsize(double *interval);
格子間隔を取得する。
interval 格子間隔（0.0 以上、デフォルト 1.0）のアドレス
返り値 成功時１　エラー時０

int g3GetTexMatrix(d3Matrix texm);
表示画面に定義されているテクスチャマトリクスを取得する。
texm マトリクス(長さ 16 の倍精度浮動小数の配列)
返り値 成功時１　エラー時０

int g3GetOrbitPoints(int *count, double **points, double *pointSize);
軌跡点のデータを取得する。
count 点の総数の格納先
points 点の座標（X,Y,Z･･･の並び）の配列の格納先

pointSize ×印のサイズの格納先
返り値 成功時１　エラー時０

int g3GetOrbitLine(int *count, double **points);
軌跡線のデータ取得を行う。曲線補間がされていること。
count 点の数の格納先
points 点の座標 (X,Y,Z･･･の並び) の配列の格納先
返り値 成功時１　エラー時０

int g3GetVisualArea(double *left, double *right, double *bottom, double *top,
  s3Image **texture);
可視範囲を取得する。
left, right,　 bottom, top 可視範囲の矩形ワールド座標(x,y)の格納先
texture 可視分布のテクスチャデータのアドレスの格納先
返り値 成功時１　エラー時０

int g3GetAntiAliasCount(int *count, int *maxCount);
アンチエリアシングの精度を取得する
count 精度の格納先
maxCount 最大精度の格納先
返り値 成功時１　エラー時０

double g3AplHeight(g3Ground *grd, double x, double y);
任意の場所の地面の高さを取得する。
grd 地面の高さを記述するメッシュデータのアドレス
x,y 高さを取得したい位置
返り値 高さ
※メッシュ格子点の高さから補間計算を行う

double g3GetGroundHeight(double x, double y);
設定されているメッシュデータを用いて任意の場所の地面の高さを取得する。
x,y 位置
返り値 地面の高さ

double g3GetVisualRate(double eye[3], double ref[3], g3SelPath *spath);
視点と、選択されたグループから可視率を取得する。

eye[3] 視点(X,Y,Z)
ref[3] 注視点(X,Y,Z)
spath 対象グループを特定する選択パスのアドレス
返り値 可視率
※視点・注視点を用いて、可視率計算用画面に、選択対象物のみを描いた場合と、全地物を描いた場合の、対象物に係るピクセル数をカウントし、対象物以外の地物で隠されない部分の比率を計算する。

void g3ReadPixelDrawarea(int sx, int sy, int width, int height, int format, int type, void *pixels);
指定した長方形領域のピクセルを配列に取得する。
sx,sy スタート座標
width 幅
height 高さ
format フォーマット（取得したい色情報の形式：GL_BGRA 等）
type タイプ（読み取ったデータを保存する配列の型：GL_UNSIGNED_BYTE 等）
pixels ピクセルの格納先アドレス
glRead Pixels()に準ずる

void g3GetVisualHeight(d3Group *g, double left, double right, double bottom, double top, double *min_z, double *max_z);
高さ方向の可視範囲を取得する（2.09 では廃止)。
g グループのアドレス t
left, right, bottom, top エリア
min_z 高さの最小値
max_z 高さの最大値

int g3GetPlanePoints(d3Group *g, double top, double bottom, double left, double right, double z, int *count, double ***points);
指定した高さの面の頂点を取得する(2.09 では、land.dll に移管)。
g グループのアドレス t
left, right, bottom, top エリア
z 高さ
count 頂点数のアドレス
points 頂点（X,Y,Z･･･の並び）
返り値 成功時 1　エラー時 0

int g3GetDivision(d3Group *g, double v1[3],double v2[3], int num, int *division);
丸めの分割数を取得する(2.09 では非使用)。
g グループのアドレス
v1[3] 丸め始めの点 (X,Y,Z)
v2[3] 丸め終りの点 (X,Y,Z)
num 円の分割数
division 分割数のアドレス
返り値 成功時 1　エラー時 0

void g3GetLightGroup(s3LightGroup **lg);
表示画面に定義されている光源グループのアドレスを取得する。
lg 光源グループのアドレス格納先

double g3GetShadowLength();
影計算において作成する影立体の長さの設定値を取得する。
返り値 影立体の長さ

int g3GetShadowMode();
影計算における原因物体設定モードを取得する。
返り値 設定モード

G3_SHADOW_ALL	1	/* 全ての物体*/
G3_SHADOW_OTHER_GROUND	2	/* 地面以外の物体*/
G3_SHADOW_ONE	3	/* 選択した1つの物体*/

int g3GetShadowLightMode();
影計算における光源の設定モードを取得する。
返り値 設定モード

G3_SHADOW_OTHERLIGHT_OFF	10	LIGHT0以外の光源をOFF
G3_SHADOW_OTHERLIGHT_ON	11	LIGHT0以外の光源をON

int g3GetStereoPers(double eye, s3Camera *PersSave, s3Camera* leftPers, s3Camera *rightPers);
ステレオ表示のパラメータを取得する。
eye 左右間隔を指定する
PersSave 基準となるカメラ情報（入力）

leftPers 計算した左眼のカメラ情報（出力）
rightPers 計算した右眼のカメラ情報（出力）
返り値　成功時１　失敗時０

void g3ReadEyeParameter();
設定ファイル eye_param.set からステレオ表示に係るパラメータを取得する。

void g3sGetLightColor(float *c);
補助描画領域の光源色を取得する。
c　光源色を格納する配列のアドレス

void g3sGetLightPosition(float *pos);
補助描画領域の光源位置を取得する。
pos 光源位置座標を格納する配列のアドレス

dbMaterial* g3sGetMaterial(void);
補助描画領域のマテリアルを取得する。
返り値　マテリアルを記述する構造体のアドレス

void g3sGetSizeSteelArea(int *width, int *height);
型鋼の見出し画像を描画する領域の幅と高さを取得する。

（６）データ削除関数

void g3DeleteDrawarea(int num);
表示エリアを削除する。

void g3sDeleteDrawarea(int num);
補助表示エリアを削除する。

void g3Clear();
背景色でクリアする。デプスバッファもクリアする。

void g3ClearScene();
カメラ情報、光源、イメージ、ルートグループをクリアする。

void g3ClearHilight();

ハイライト数を 0 にする。
強調表示する場合は、g3Enable(G3_DRA_HILIGHT)を呼ぶこと。

void g3ClearOrbitPointHilight()
軌跡点のハイライトをクリアする。

void g3AplFreeDepth(g3Ground *grd);
地面の高さを求めるためのデプスバッファ（メッシュ情報）をクリアする
grd メッシュデータのアドレス

void g3ReleaseGroundPixel();
現在の地面の高さ情報をクリアする。

void g3Bebas();
表示領域の配列（メモリ・ブロック）を削除する。

void g3BebasPathList();
選択された表示領域の、オブジェクト選択状態を示すデータ（PathList）を削除する。

（7）表示関数

void g3Draw();
全地物を表示する。

void g3sColorSashareaDraw();
グラフィックなマテリアル編集の見出し画像領域を描画する。一つの長方形エリアの中に、全てのマテリアルを短冊状に表示する。

void g3sLightAreaDraw();
光源ユニットの色を表示する補助画面を描画する。

void g3sMaterialAreaDraw();
マテリアル編集画面（オリジナル）の設定カラーを表示する補助画面を描画する。

void g3sMaterialAreaDraw2();
グラフィックなマテリアル編集画面の設定カラーを表示する補助画面を描画する。

void g3sPlainMaterialAreaDraw();
グラフィックなマテリアル編集画面の選択候補マテリアル群を平面表示する補助画面を描画する。

void g3sSteelSectionDraw(int type);
型鋼の種類別パラメータの説明画面を描画する。
type 型鋼の種類
G3_HSTEEL 0
G3_CSTEEL 1
G3_TSTEEL 2
G3_LSTEEL 3

void g3Resize(int width, int height);
表示エリアのリサイズをする
width 幅
height 高さ

void g3ClearRectScreen();
矩形ライン表示をクリアする

void g3DrawRectScreen(float sx, float sy, float ex, float ey);
正規化されたスクリーン座標で矩形ラインを表示する
sx,sy スタート（表示エリアを 1.0 に正規化した座標）
ex,ey エンド（表示エリアを 1.0 に正規化した座標）
フロントバッファに直接描画するので、wg3Swapbuffers は呼んではいけない。
ラバーバンド表示用である

（8）ウインドウ操作関数

int wg3EntryDrawarea(void *w, void *c);
表示エリアの生成
w ウィジェットクラスのポインタ(Windows 系では、HDC を格納する変数のアドレス)
c コンテキスト(Windows 系では HGLRC を格納する変数のアドレス)
返り値 成功時 1　エラー時 0（意味は、制限個数(10)オーバーのエラー）

int wg3ReentryDrawarea(void *oldw, void *neww, void *newc);
表示エリアを作り直した場合、例えば singlebuffer と doublebuffer の切り替えなどで、

旧情報(g3Set 関数)をキープしたままウィジェットだけを交換する
以降は、新しいウィジェット(neww)でアクセス可能になる
oldw 古いウィジェットクラスのアドレス(Windows 系では､HDC を格納する変数のアドレス)
neww 新しいウィジェットクラスのアドレス(同上)
newc 新しいコンテキスト(Windows 系では HGLRC を格納する変数のアドレス)
返り値 成功時 1　エラー時 0（意味は、oldw が未登録のエラー）

void wg3DeleteDrawarea(void *w);
表示エリアを削除する。
w ウィジェットクラスのアドレス(Windows 系では、HDC を格納する変数のアドレス)

int wg3AssignDrawarea(void *w);
表示エリアを指定する。以後の処理はこの表示画面に対して行われる。
w ウィジェットクラスのアドレス(Windows 系では、HDC を格納する変数のアドレス)
返り値 成功時 1　エラー時 0

void wg3Swapbuffers();
バッファをスワップする。

void wg3SetRedrawFlag(int flag);
再表示フラグの ON/OFF をする。
flag フラグ (ON:1/OFF:0)
フラグが ON のとき通常通り再表示し、OFF のとき再表示しない

void g3SetCameraHorizontalPutar(double rate);
視点を水平回転（パン）する。
rate 回転角

int g3sSetDrawarea(int num);
処理対象とする補助描画画面を選択する。
num 対象とする補助描画画面の番号
返り値 成功時 TRUE　失敗時 FALSE

void g3sSetLightColor(float *c);
補助描画領域の光源色を設定する。
c　光源色の配列

void g3sSetLightPosition(float *pos);
補助描画領域の光源位置を設定する。
pos 光源位置座標の配列

void g3sSetMaterial(dbMaterial *m);
補助描画領域のマテリアルを設定する。

void g3sSetTexture(dbTexture *tex);
補助描画領域のテクスチャを設定する。

void g3sSetMaterialFile(char *name);
補助描画領域にマテリアルファイル名をコピーしたメモリ・ブロックを設定する。
name マテリアルファイル名の文字列

g3sSetSize(int width, int height)
選択されている補助描画領域のサイズを設定する。
width 幅
height 高さ

int g3sSetSizeSteelArea(int width, int height);
型鋼の見出し画像を描画する領域の幅と高さを設定する。
width 幅
height 高さ

（９）その他関数

void g3TransformScreen2World(int sx, int sy, double *x, double *y, double *z);
スクリーン座標からワールド座標へ変換する。
sx, sy スクリーン座標（左上を(0,0)、右下を(width-1,height-1）とした整数値
x, y, z ワールド座標のアドレス（奥行き方向の値は 0.0 で返す）
※オルソのみ

Ｂ－５．インタプリタ（ＩＰ）

（１）データ構築関数

d3Group *i3LoadLssg(const char *gname, const char *file);

ジオメトリファイル(LSS-G 形式)を読み込む。
gname グループ名称
file ファイル名
返り値 グループのアドレス

d3Group *i3LoadTmpGroup(const char *file);
一時的ファイルを読み込む。ファイル中の最上位のグループを返り値とする。
file ファイル名
返り値 グループのアドレス
※最上位グループは、上方リンクが存在しないことをもって識別している

void i3CallCube(char *name, double x, double y, double z, double lx, double ly, double lz);
外部コマンド（直方体）を実行する(2.09 では非使用)。
name グループ名称
x, y, z 物体原点(X、Y、Z)
lx, ly, lz 幅、奥行き、高さ

void i3CallSphere(char *name, double x, double y, double z, double r);
外部コマンド（球）を実行する(2.09 では非使用)。
name グループ名称
x , y, z 物体原点
r 半径

void i3CallCylinder(char *name, double x1, double y1, double z1, double x2, double y2,
  double z2, double r);
外部コマンド（円柱）を実行する(2.09 では非使用)。
name グループ名称
x1, y1, z1 下円中心
x2, y2, z2 上円中心
r 半径

void i3CallCone(char *name, double x1, double y1, double z1, double x2, double y2,
  double z2, double r1, double r2);
外部コマンド（円錐、円錐台）を実行する(2.09 では非使用)。

name グループ名称
x1, y1, z1 下円中心
x2, y2, z2 上円中心
r1, r2 下円半径、上円半径

void i3CallFlatCylinder(char *name, double x1, double y1, double z1, double x2,
  double y2, double z2, double r, int n);
外部コマンド（角柱）を実行する(2.09 では非使用)。
name グループ名称
x1, y1, z1 下面中心
x2, y2, z2 上面中心
r 中心から頂点の長さ
n 角数

void i3CallFlatCone(char *name, double x1, double y1, double z1, double x2, double
y2,
  double z2, double r1, double r2, int n);
外部コマンド（角錐、角錐台）を実行する(2.09 では非使用)。
name グループ名称
x1, y1, z1 下面中心
x2, y2, z2 上面中心
r1, r2 下面から頂点の長さ、上面中心から頂点までの長さ
n 角数

void i3CallCSteel(char *name, double a, double b, double c, double d, double h);
外部コマンド（溝形鋼）を実行する(2.09 では非使用)。
name グループ名称
a, b, c, d, h 高さ , 幅、厚み、縦板の厚み、長さ

void i3CallTSteel(char *name, double a, double b, double c, double d, double h);
外部コマンド（T型鋼）を実行する(2.09 では非使用)。
name グループ名称
a, b, c, d, h 高さ , 幅、厚み、縦板の厚み、長さ

void i3CallHSteel(char *name, double a, double b, double c, double d, double h);
外部コマンド（H型鋼）を実行する(2.09 では非使用)。

name グループ名称
a, b, c, d, h 高さ , 幅、厚み、縦板の厚み、長さ

void i3CallLSteel(char *name, double a, double b, double c, double d, double h, double x, double y);
外部コマンド（Ｌ型鋼）を実行する(2.09 では非使用)。
name グループ名称
a, b, c, d, h 高さ , 幅、厚み、縦板の厚み、長さ
x, y 原点からの離れ

void i3CallSweep1 (char *name, char *face, char *line);
掃引体１面を生成する(2.09 では非使用)。
name グループ名
face 断面ファイル名
line 中心線軌跡ファイル名

void i3CallSweep2 (char *name, char *face1, char *face2);
掃引体２面を生成する(2.09 では非使用)。
name グループ名
face1 断面ファイル名
face2 断面ファイル名
※Ver.2.09 においては、外部関数の起動は、原始図形とユーザー定義のパラメトリック部品を共通化したため、以上の各原始図形専用の形状生成関数は使用しない。

int IP_interpret(char *command);
command を解析し、DML ライブラリを用いてデータを構築・取得する。
command コマンド文字列
※インタプリタ（ＩＰ）の処理ループは、データファイルの一行である。一方、コマンドの実行単位は、セミコロン「;」を区切りとしている。従って、本関数の呼び出しは、複数行にまたがるコマンドの一部である場合（例１）も、また複数のコマンドを含む１行である場合（例２）もある。

例１：（１回目）G001=
（２回目）face(V01,V02,V03,
（３回目）V04,V05,V06);
この場合、１回目と２回目は、コマンドをスタックする。

３回目の';'で初めてコマンドを実行する。

例２：　V01=COORD(0, 0, 0,); V02=COORD(1, 0, 0);
内部ループで２つのコマンドを実行する。

FILE *i3_outputOpen(const char *filename);
出力ファイル(LSS-S または LSS-G)をオープンする。
filename ファイルのパス名
返り値 ファイルのアドレス(NULL は失敗)
※冒頭行にバージョンを表記するコメント行を出力する。
最適化保存が選択されている場合には、テンプレートとなる要素定義を出力する。

void i3_outputClose(FILE *fp);
出力ファイルをクローズする。
fp ファイルポインタ
※最適化保存を行った場合には、末尾にサマリーをコメント行として追加する。

（２）データ定義/更新関数：

void i3UpdateParentsAndMe(d3Group *g);
アップデートフラグを親グループと自グループに立てる。
g グループのアドレス
※ファイル参照により地物に追加されたオブジェクトに関して、形状、カラー、テクスチャ等が変更され、元のファイル（既存部品）とは異なるものになった場合にはこのフラグを立て、ファイル保存時点で、その内的構成まで出力する。

void IP_TraverseError(int (*func)(int mesnum, const char *basicmes, const char *var1, const char *var2));
インタプリタのエラー検出時の割り込み関数をデフォルトとは異なるものに設定する。
func 設定するエラー処理関数(NULL の場合は、デフォルトに戻す)。
(以下は、引数 func で設定する関数に渡される引数リストである)
mesnum メッセージ番号
basicmes メッセージ固定文字列
var1, var2 メッセージ可変文字列(NULL は無しを示す)
※１つのコマンド解析でエラーが複数発生する場合がある。処理関数の戻りを TRUE にすると、できる限りコマンド解析を続け、別のエラーも検出するが、FALSE にすると現在のコマンド解析処理を中止する（エラー処理を、逐次メッセージ表示とする場合、エラーが２つ

以上あるとオペレータにとって煩わしさが増幅するため、通常 FALSE が良いと思われる)。
本関数を使わないときは、エラーメッセージ (basicmes) がコンソールに表示される。
現在サポートされているエラーとその可変部は次の通り。
I3_MSGNO_NotFoundCommandName: var1:commandline, var2:line#
I3_MSGNO_UndefCommandName: var1:command, var2:line#
I3_MSGNO_IllegalCommandLine: var1:commandline, var2:line#
I3_MSGNO_NotFoundName: var1:commandtype, var2:line#
I3_MSGNO_DupName: var1:name, var2:line#
I3_MSGNO_InsufParams: var1:name, var2:line#
I3_MSGNO_UndefName: var1:name, var2:line#
I3_MSGNO_CreateError: var1:name, var2:line#
I3_MSGNO_FileOpen: var1:name, var2:line#
I3_MSGNO_IllegalArgNum: var1:arg, var2:line#
I3_MSGNO_IllegalArgWord: var1:arg, var2:line#
I3_MSGNO_NotSupportedCommandName: var1:command, var2:line#

void i3SetAttribute(d3Group *g, const char *str);
グループに属性を追加する。
g グループのアドレス
str 属性文字列(例：“&GROUND”)

void i3SetDefaultEnv(int type);
参照先ディレクトリに係る環境を設定する。
type 環境のタイプ
I3_DEFENV_MASTER 通常編集時のディレクトリ構成
I3_DEFENV_YURYO 景観事例データベースのディレクトリ構成
I3_DEVENV_YOUSO 景観構成要素データベースのディレクトリ構成
I3_DEFENV_ZAIRYO 景観材料データベースのディレクトリ構成

void i3SummitStack(d3Group *group);
サミットグループを通知する (2.09 では廃止)
group サミットグループのアドレス

int i3_outputFile(const char *filename, d3Group *group);
グループを指定のファイルに出力する。
filename 出力ファイル名 (フルパスで指定する)

group グループのアドレス
返り値 成功時１　エラー時０

int i3_outputScenesDirect(const char *file);
シーン(LSS-S 形式)をセーブする。
file ファイル名（フルパスで指定する）
返り値 成功時１　エラー時０

int i3_outputGroup(FILE *fp, d3Group *group);
オープン済の出力ファイルにグループを追加出力する。
fp ファイルポインタ
group グループのアドレス
返り値 成功時１　エラー時０

int i3_outputGroupsDirect(const char *filename, d3Group **groups, int grpcount, int summitDel);
全てのグループをファイル(LSS-G)に保存する。
filename ファイル名
groups グループ配列のアドレス
grpcount ルートグループ数
summitDel 0:サミットグループを削除する　-1:サミットグループをしない
返り値 成功時１　エラー時０

int i3_outputScenes(i3 *ip, i3Parts *parts);
OUTPUT_SCENE コマンドを実行し、全てのシーンをセーブする。
ip 解析中のファイルに関する情報を格納した構造体
parts 引数リスト（この場合、保存ファイル名を指定）
返り値 成功時１　エラー時０

int i3_outputScene(FILE *fp, s3Scene *scn);
一つのシーンをファイル出力する。
fp ファイルポインタ
scn シーン構造体のアドレス
返り値 成功時１　エラー時０

char* i3_allocCom(i3* ip);

LSS ファイルを解析し、一つのコマンド単位を抽出する。コマンドは、「;」を区切りとして、余分な余白、タブコード等を除き、複数行に跨るものは連結して一つのコマンドとする。
ip 解析中のファイルに関する情報を格納した構造体
返り値 コマンド単位文字列を格納したメモリ・ブロックのアドレス

char* i3_allocCopyStr (const char *str,char delim);
コマンド文字列から、指定したデリミタまたは「¥0」までの部分文字列を抽出する。
str 元の文字列
delim デリミタ
返り値 抽出された文字列を格納した新たなメモリ・ブロックのアドレス

char* i3_allocGroupName(const char *name, const char *gprefix);
接頭辞を付加したグループ名称を作成する。
name グループの本文
gprefic 接頭辞
返り値 作成した名称を格納する新たなメモリブロックのアドレス

int i3_allocParts(i3 *ip,const char *com,i3Parts *parts);
コマンド文字列を、コマンドと引数に分解する。
ip 解析対象のファイルに関する情報
com コマンド文字列（入力）
parts 結果を格納する文字列のアドレス配列のアドレス
返り値 成功時：TRUE 失敗時：FALSE

int i3_camera(i3 *ip,i3Parts *parts);
カメラ情報（パース表示のためのパラメータ）を解析し、S3 構造体を構築した上で、ip に追加する。
ip 解析対象のファイルに関する情報
com コマンド文字列（入力）
返り値 成功時：TRUE 失敗時：FALSE

i3Parts* i3_copyParts(i3Parts *parts);
コマンドと引数をコピーする。
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
返り値　複製された文字列アドレスの配列のアドレス

int i3_delete(i3 *ip, i3Parts *parts);
DELETE コマンドを実行する。
ip 解析対象のファイルに関する情報
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

int i3_effect(i3 *ip, i3Parts *parts);
EFFECT コマンドを実行する。
ip 解析対象のファイルに関する情報
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

int i3_effectGroup(i3 *ip, i3Parts *parts);
EFFECTGROUP コマンドを実行する。
ip 解析対象のファイルに関する情報
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

int i3_execCom(i3 *ip, const char *com, const char *gprefix);
コマンドを識別し、それぞれのコマンド処理に分岐・実行する。
ip 解析対象のファイルに関する情報
com コマンド文字列（入力）
gprefix 接頭辞（新たなグループを生成する場合、グループ名称を修飾する）
返り値　成功時：TRUE、失敗時：それぞれの処理の実行結果

int i3_file(i3 *ip, i3Parts *parts, const char gprefix)
FILE コマンドを実行し、外部ファイルまたは外部関数を使用する。
ip 解析対象のファイルに関する情報
com コマンド文字列（入力）
gprefix 接頭辞（新たなグループを生成する場合、グループ名称を修飾する）
返り値　成功時：TRUE、失敗時：それぞれの処理の実行結果

void i3_freeParts(i3Parts *parts);
コマンド解析の中間データを解放する。
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス

void i3_freeUserInfo(d3Group *g);
グループに定義された、属性などのユーザー情報を解放する。
g 処理対象とするグループ構造体のアドレス

i3Record* i3_getEntryRecord(i3 *ip, int typ, const char *name);
名称(面、グループ、リンク等)を参照するコマンドを実行し、名称をタイプ別に辞書にから検索する。未登録の場合には新たに登録する。
ip 解析対象のファイルに関する情報
typ データの種類
name 名称
返り値　辞書への登録内容

i3ExtCommand *i3_getExtCommand(const char *name);
外部関数登録ファイル(ext.tab)を検索し、登録内容（引数リスト）を取得する。
name 外部関数名
返り値　登録内容を記述した構造体のアドレス

i3ExtTable *i3_getExtTable();
外部関数登録ファイル(ext.tab)を開き、内容をメモリ上にロードする。
返り値　関数リスト（配列）と登録数を内容とする構造体のアドレス

int i3_getoptim()
LSS-G ファイルの保存に際して適用する最適化のレベルを取得する。
返り値　最適化レベル

int i3_group(i3 *ip, i3Parts *parts, const char *gprefix);
GOURP コマンドを実行する。
ip 解析対象の LSS-G ファイルに関する情報
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
gprefix 接頭辞（新たなグループを生成する場合、グループ名称を修飾する）
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

int i3_image(i3 *ip, i3Parts *parts);
IMAGE コマンドを実行する。
ip 解析対象のファイルに関する情報
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス

返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

int i3_initip(i3 *ip);
解析情報を初期化する。
ip 解析対象のファイルに関する情報
返り値　成功時：TRUE、失敗時：FALSE(ip が NULL の場合)

int i3_interpret(i3 *ip, const char *com_line, const char *gprefix);
コマンド行を解析し実行する。
ip 解析対象のファイルに関する情報
com_line コマンド行
gprefix 接頭辞（新たなグループを生成する場合、グループ名称を修飾する）
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

void i3_jounal(i3 *ip, const char *com);
解析経過をコンソール出力する（2.09 では何もしない）
ip 解析対象のファイルに関する情報
com コマンド

int i3_lf(i3 *ip, i3Parts *parts, int typ);
FACE コマンドから面の頂点リストを構築する。
ip 解析対象のファイルに関する情報
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

int i3_lfAttr(i3 *ip, i3Parts *parts, int typ int atyp);
FACE_COLOR、FACE_NORMAL、FACE_MATERIAL、FACE_TEXTURE、LINE_COLOR、LINE_NORMAL、LINE_MATERIAL および LINE_TEXTURE の各コマンドを実行し、カラー、法線、マテリアル及びテクスチャの属性を、面または線に定義する。
ip 解析対象のファイルに関する情報
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
typ 属性を付ける対象の種類(面または線)
atyp 属性の種類(カラー、法線、マテリアル及びテクスチャ)
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

int i3_lfGroup(i3 *ip, i3Parts *parts, int typ, const char *gprefix, int concave);

GROUP_FACE コマンドを実行しグループに面群を定義する。
ip 解析対象のファイルに関する情報
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
typ 名称の種類（この場合、面）
gprefix 接頭辞（新たなグループを生成する場合、グループ名称を修飾する）
concave 凹ポリゴンであることを示すフラグ
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

int i3_light(i3 *ip, i3Parts *parts);
LIGHT コマンドを実行し、光源ユニットを構築する。
ip 解析対象のファイルに関する情報
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

int i3_lightGroup(i3 *ip, i3Parts *parts);
LIGHTGROUP コマンドを実行する（光源グループ）。
ip 解析対象のファイルに関する情報
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

int i3_link(i3 *ip, i3Parts *parts, const char *gprefix);
LINK コマンドを実行する。
ip 解析対象のファイルに関する情報
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
gprefix 接頭辞（新たなグループを生成する場合、グループ名称を修飾する）
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

int i3_linkxform(i3 *ip, i3Parts *parts);
LINK_XFORM コマンドを実行する。
ip 解析対象のファイルに関する情報
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

int i3_model(i3 *ip, i3Parts *parts);
MODEL コマンド(シーンへのモデルの読み込み)を実行する。
ip 解析対象のファイルに関する情報

parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

int i3_mt(i3 *ip, i3Parts *parts, int typ);
MATERIAL 及び TEXTURE コマンドを実行する。
ip 解析対象のファイルに関する情報
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
typ マテリアルかテクスチャかの区別
 I3_TYP_MATERIAL 100
 I3_TYP_TEXTURE 110
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

int i3_mtGroup(i3 *ip, i3Parts *parts, int typ, const char *gprefix);
GROUP_MATERIAL 及び GROUP_TEXTURE コマンドを実行する。
ip 解析対象のファイルに関する情報
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
typ マテリアルかテクスチャかの区別
 I3_TYP_MATERIAL 100
 I3_TYP_TEXTURE 110
gprefix 接頭辞（新たなグループを生成する場合、グループ名称を修飾する）
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

i3Record* i3_newList(i3 *ip, int typ, const char *name);
 インタプリタの辞書に新たなエントリーを作成する。
ip 解析対象のファイルに関する情報
typ エントリのタイプ
name エントリの名称
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

int i3_nextEntryRecord(i3 *ip, int typ, int startix);
インタプリタの辞書において指定した番号以降の、指定したタイプの最初のエントリを返す(ver.2.09 では非使用)。
ip 解析対象のファイルに関する情報
typ エントリのタイプ
startix 検索開始番号
返り値　該当するエントリ番号　または I3_NULL_INDEX（該当無き場合）

int i3_output(i3 *ip, i3Parts *parts, const char *gprefix);
OUTPUT コマンドを実行する。パーツの名称のファイルを作り、内容を出力する。
ip 解析対象のファイルに関する情報
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
gprefix 接頭辞（新たなグループを生成する場合、グループ名称を修飾する）
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

int i3_outputFace(FILE *fp, d3Face *f, int type);
一つの面をファイル出力する。条件設定により最適化出力を行う。
fp 出力先ファイル
f 出力する面
type 面(FACE)または線(LINE)の区別
 I3_EXT FACE 5
 I3_EXT_LINE 6
返り値　成功時：TRUE、失敗時：FALSE

int i3_outputGroupFace(FILE *fp, const char *gname, d3Face *f, int type);
何もしない
返り値：TRUE
※外部関数の引数がメモリ上にしか存在しない FACE 等となっていた場合に、ファイル保存に際して、この FACE を独立したファイルに出力する必要がある。このようなタイプの外部関数がまだ存在しないため、実装していない。

int i3_reset(i3 *ip);
インタープリタをリセットし、ＤＭＬメモリ空間を初期化する。
ip 解析対象のファイルに関する情報
返り値 d3Initialize の処理結果

void i3_resetIp(i3 *ip);
インタープリタの解析結果を解放する。
ip 解析対象のファイルに関する情報

int i3_scene(i3 *ip, i3Parts *parts);
SCENE コマンドを実行する。
ip 解析対象のファイルに関する情報

parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

int i3_searchEntryRecord(i3 *ip, int typ, const char *name);
インタプリタの解析結果から、所定のラベル名、タイプのエントリを検索する。
ip 解析対象のファイルに関する情報
typ エントリのタイプ（グループ、面、頂点、・・・）
name エントリの名称
返り値 エントリの番号または I3_NULL_INDEX(存在しない場合：-1)

int i3_setErrMsg(i3 *ip, int mesnum, const char *var);
インタープリタのエラーメッセージを保存する。
ip 解析対象のファイルに関する情報
mesnum メッセージ番号
var 補足情報（パラメータ等）
返り値 TRUE（デフォルトの出力先の場合）または指定出力関数（IP_TraverseError 関数により出力先の変更が行われた場合）の処理結果

int i3_setFace(i3 *ip, int ix);
インタプリタが解析した面に関する情報から、d3Face 構造体を生成する。
ip 解析対象のファイルに関する情報
ix インタープリタの解析結果における処理対象面の登録番号
返り値 成功時 TRUE　失敗時 FALSE

void i3_setoptim(int i);
LSS-G ファイル保存における最適化のレベルを設定する。
int i 最適化のレベル(ビット単位のスイッチとして表現する)
 I3_OPT_3　1　三角形の連続出力
 I3_OPT_4　2　四角形の連続出力
 I3_OPT_C　4　カラーのソート
 I3_OPT_M　8　マテリアルのソート
 I3_OPT_N 16　法線のソート
 I3_OPT_T 32　テクスチャのソート

void i3_setPath(char *pathname, e3Param *envdir, const char *filename);
環境設定ファイルの指定ディレクトリと、ファイル名からフルパスのファイル名文字列を

合成する。
pathname 結果を格納する文字列
envdir 環境設定ファイルで指定されたディレクトリ名称
filename ファイル名

void i3_stackCom(i3 *ip, const char *com_line);
入力ファイルから取得した１行分の文字列をコマンド列に追加する。コメント行は無視する。
ip 解析対象のファイルに関する情報
com_line ファイルから入力した１行分のテキスト

int i3_time(i3 *ip, i3Parts *parts);
TIME コマンドを実行する。
ip 解析対象のファイルに関する情報
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

int i3_vertex(i3 *ip, i3Parts *parts);
VERTEX コマンドを実行する。
ip 解析対象のファイルに関する情報
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
返り値　成功時：TRUE、失敗時：i3_setErrMsg 関数の返り値

int i3_vntc(i3 *ip, i3Parts *parts, int typ);
COORD, NORMAL, TCOORD, COLOR のコマンドを実行する。
ip 解析対象のファイルに関する情報
parts コマンドを構成要素に分解した文字列アドレスの配列のアドレス
typ コマンドのタイプ
 I3_TYP_V 10 頂点座標
 I3_TYP_N 20 法線ベクトル
 I3_TYP_T 30 テクスチャ座標
 I3_TYP_C 40 カラー値(R,G,B,A)
返り値 TRUE

（３）データ取得関数

char *i3GetAttribute(d3Group *g, int num);

グループに定義されている属性を取り出す。
g グループのアドレス
num 属性の番号(0,1,2,・・・)
返り値 成功時 属性を記述する文字列のアドレス　エラー時 NULL

d3Face *IP_getface(const char *lssgfile);
ジオメトリファイル(LSS-G 形式)の最初に出てくる面を取得する。
lssgfile ファイル名
返り値 成功時 面のアドレス　失敗時 NULL
※この時、目的とする面を擁するグループも生成するが、2.09 では、別プロセスである外部関数の引数の解析に使用しているため、処理終了後は全て解放され、メイン画面の表示に現れることはない。

char* i3GetGroupNamePtr(d3Group *group);
グループ名を取得する(2.09 では廃止)。
group グループのアドレス
返り値 グループ名のグループのアドレス

int i3GetDefaultEnv();
参照先ディレクトリに係る環境設定を取得する。
返り値
 I3_DEFENV_MASTER 通常編集時のディレクトリ構成
 I3_DEFENV_YURYO 景観事例データベースのディレクトリ構成
 I3_DEVENV_YOUSO 景観構成要素データベースのディレクトリ構成
 I3_DEFENV_ZAIRYO 景観材料データベースのディレクトリ構成

int i3GetGeoDefaultEnv();
現在の環境設定下における LSS-G ファイルの参照先ディレクトリを取得する。
返り値：
 E3_FILE_PATH_MASTER_GEOMETRY 通常編集
 E3_FILE_PATH_JIREI_GEOMETRY 景観事例
 E3_FILE_PATH_YOUSO_GEOMETRY 景観構成要素
 E3_FILE_PATH_ZAIRYO_GEOMETRY 景観材料

int i3GetSceneDefaultEnv
現在の環境設定下における LSS-S ファイルの参照先ディレクトリを取得する。

返り値：

E3_FILE_PATH_MASTER_SCENE 通常編集

E3_FILE_PATH_JIREI_SCENE 景観事例

E3_FILE_PATH_YOUSO_SCENE 景観構成要素

E3_FILE_PATH_ZAIRYO_SCENE 景観材料

（４）問い合わせ関数

int i3IsAttribute(d3Group *g, const char *keyword);

グループに指定の属性があるかチェックする。

g グループのアドレス

keyword 属性文字列(例：“&GROUND”)

返り値 ある時１　ない時０

int i3IsGeometry(const char *file);

ジオメトリファイルかチェックする(2.09 では廃止)。

file ファイル名

返り値 成功時１　エラー時０

int i3_checkRootGroup(d3Group *g);

グループがルートグループかどうかをチェックする。

g グループのアドレス

返り値 ルートである時１　ルートではない時０

※上方リンクの有無を調べる。

Ｂ－６．環境設定ライブラリ（ＥＮＶ）

（１）データ構築関数

void e3SetDataPath(int type);

データベースのパスをセットする(2.09 では廃止)。

type データベースタイプ

void e3ReadSettingFile(char *filename);

環境設定ファイル(デフォルト名：kdbms.set)を読み込む。

filename ファイル名

（２）初期化関数

int e3Initialize();
ENV を初期化する(2.09 では廃止)。
返り値 成功時１　エラー時０

（３）データ定義/更新関数

void e3SetKeywordDefault();
デフォルトパラメータで環境設定を行う。
※環境設定ファイル(kdbms.set)の読み込みに失敗した場合や、定義が欠如に項目に備える。

void e3SetKeyword(char *buf);
文字列（環境設定ファイルの１行に相当）を解析し、当該項目にパラメータをセットする。
buf 文字列

void e3SetPath(int index);
ディレクトリ指定項目をセットする。
index 環境設定の番号
※指定した項目が、ホーム・ディレクトリからの相対アドレスで指定されていた場合に、これをフルパスの記述に変換する。

int e3SaveFile(char *filename);
環境ファイルを保存する。
filename ファイル名
返り値 成功時１　エラー時０
※コメント行などは、上書き保存で削除されるので注意を要する。

void e3SimpangKerja(char *tempat);
作業用ディレクトリを登録する。
tempat 作業用ディレクトリ

void e3GantiTitle(int i, char* t);
環境変数の表示タイトルを変更する。表示言語を切り替える際に使用する。
i 環境変数の ID 番号
t 新たなラベル名

（４）データ取得関数

char *e3GetDataPath(int type);

データベースのパスを取得する(2.09 では非使用)。
type データベースタイプ
返り値 パスの文字列
※初期のバージョンにおけるセットアップのディレクトリに対応していた。

int e3GetType(int index);
キーワードのタイプを取得する
index 環境設定項目の番号
返り値 タイプ

E3_TYPE_HOME_PATH	0 ホームディレクトリ（フルパス）
E3_TYPE_FILE_PATH	1 ディレクトリ（フルパスまたはホームからの相対）
E3_TYPE_FILE_NAME	2 ファイル名
E3_TYPE_SELECT	3 強調表示タイプ
E3_TYPE_LONG_1	4 一つの整数
E3_TYPE_LONG_2	5 二つの整数
E3_TYPE_LONG_3	6 三つの整数
E3_TYPE_LONG_4	7 四つの整数
E3_TYPE_FLOAT_1	8 一つの浮動小数
E3_TYPE_FLOAT_2	9 二つの浮動小数
E3_TYPE_FLOAT_3	10 三つの浮動小数
E3_TYPE_FLOAT_4	11 四つの浮動小数
E3_TYPE_STRING	12 文字列

char *e3GetTitle(int index);
環境設定項目の表示タイトルを取得する。
index 項目の通し番号
返り値 表示タイトルの文字列のアドレス
※環境設定の編集ダイアログで表示する、各言語でのタイトル(例：ホームディレクトリ)

char *e3GetName(int index);
環境設定項目のエントリ名を取得する。
index 項目の通し番号
返り値 エントリ名称の文字列
※環境設定ファイルに用いるエントリ名称(例：“E3_HOME_PATH”)

char *e3GetSelectParam(int index);

環境設定項目の選択肢を取得する。
index　環境設定項目の通し番号
返り値 選択肢の文字列

int e3GetFile(FILE *fp, char *buf, int size);
ファイルの１行分の文字列を取得する。
fp ＦＩＬＥポインタ
buf 文字列のアドレス
size 取得するサイズ（バッファ長以下）
返り値 成功時１　エラー時０

e3Param *e3GetEnvParam(int index);
指定した 環境設定項目のパラメータ部を取得する。
index　環境設定項目の通し番号
返り値 パラメータ構造体のアドレス

char *e3GetItemString(int itemno);
パラメータを文字列で取得する。
itemno　環境設定項目の通し番号
返り値 文字列

const char e3AmbilKerja();
作業用ディレクトリを取得する。
返り値 文字列（メモリ・ブロックのアドレス）

（５）データ削除関数

void e3ResetEnvParam();
環境設定ファイル(kdbms.set)データが読み込み済みであることを示すフラグを降ろす(多重読み込み防止用)。

void e3Clear();
ＥＮＶライブラリに関連したメモリ・ブロックを解放する。

void e3BebaskanKerja();
作業用ディレクトリを解除する。

（６）データ比較関数÷

int e3CheckDirName(char *path, char *dname);
指定したパスに指定したディレクトリがあるかチェックする。
path パス
dname ディレクトリ名
返り値 ある:1　ない:0

Ｂ－７．ユーティリティライブラリ（Ｕ３）

（１）データ構築関数÷

void *u3SaveGroupFamily(d3Group *g);
グループ以下全てを一時的ファイルに保存（バックアップ）する。
g グループのアドレス
返り値 成功時:u3SaveGroup 構造体のアドレス　エラー時:NULL

int u3B3Spline(int loop, int pCount, double *points, int interp, double *outPoints);
３次の B スプライン補間をする。
loop True は面、False は線
pCount 軌跡点数
points 軌跡点列(X,Y,Z･･･の並び)
interp 補間点数（１以上）
outPoints 出力座標(X,Y,Z･･･の並び)
返り値 成功時１　エラー時０

d3Group *u3RestoreGroup Family(int load, void *sginfo);
保存してある一時的ファイル（バックアップ）からグループを復元する。
load load＞0 なら復元
load≦0 なら領域の解放のみ
sginfo u3SaveGroup 構造体のアドレス（u3SaveGroupFamily の返り値）
返り値 グループのアドレス

int u3PosAlongLine(int pCount, double *points,double interval, int random, double shiftUMax, double shiftVMax, double **outPoints);
線上の配置位置を求める。
pCount 入力線の点数
points 入力線の座標列(X,Y,Z･･･の並び)

interval 間隔
random 位置の揺らぎに関する乱数の適用方法を指定
  U3_RAND_NONE　　0　なし
  U3_RAND_FLAT　　1　一定乱数　(u3RandomOffset関数の第一引数)
  U3_RAND_LINEAR　2　リニア乱数（同上）
shiftUMax 線方向の最大ずれ
shiftVMax 線に XY 面垂直方向の最大ずれ
outPoints 出力点群の座標(alloc される)
返り値 出力点数

int u3PosInXYArea(int pCount, double *points, double origin[3], double uVec[3], double vVec[3], int random, double shiftUMax, double shiftVMax, double **Points);
多角形エリア内の配置位置を求める。
pCount 入力線の点数
points 入力線の座標列(X,Y,Z･･･の並び)
origin[3] 基準点座標
uVec[3] u 軸方向と間隔
vVec[3] v 軸方向と間隔
random 位置の揺らぎに関する乱数の適用方法を指定
  U3_RAND_NONE　　0　なし
  U3_RAND_FLAT　　1　一定乱数　(u3RandomOffset関数の第一引数)
  U3_RAND_LINEAR　2　リニア乱数（同上）
shiftUMax u 方向の最大ずれ(uVec を 1)
shiftVMax v 方向の最大ずれ(vVec を 1)
outPoints 出力点群の座標(alloc される)
返り値 出力点数
※多角形のエリア内に origin を基準とし、uVec と vVec 方向に格子配置する。エリア内のチェックは XY 投影図で行う。

double u3RandomOffset(int random, double shiftMax);
ランダムな位置ずれを求める。
random ランダムタイプ
  U3_RAND_NONE　　0　なし（常に0.0を返す）
  U3_RAND_FLAT　　1　一定乱数（0を中心に+-shiftMaxの範囲で一様）
  U3_RAND_LINEAR　2　リニア乱数 4(r-0.5)|r-0.5|shiftMax, 確率密度は$|x|^{-1}$に比例)
shiftMax 最大のずれ量

返り値 乱数値（平均0.0、最小-shiftMax、最大+shiftMax）

double u3RandomRate(int random, double rateMax);
ランダムなスケールを求める。
random ランダムタイプ
rateMax 最大スケール量
返り値 平均1.0、最小1/rateMax、最大rateMax

int u3RandomNumber(int count, double *rates);
ランダムな種類を求める。
count 種類数
rates 各種類の割合
返り値 0～countまでの値
※countの場合は、どれも選ばれなかったことを示す(ratesの合計が1.0に満たない場合)。

void u3Offsetline(int pCount, double *points, int whichSide, double offsetXY, double offsetZ, double *outPoints);
オフセットした線を求める。
pCount 基準線の点数
points 基準線の座標列（X,Y,Z･･･の並び）
whichSide 右か左か（進行方向）
offsetXY　XYのオフセット
offsetZ　Zのオフセット
outPoints オフセットされた線の座標列（X,Y,Z･･･の並び）

int u3SlantPolygon(int count, double *points, double **outPoints);
スプラインによりポリゴンのスムージングを行う。
count 点数
points 入力座標（X,Y,Z･･･の並び）
outPoints 出力座標（X,Y,Z･･･の並び）
返り値 点数

int u3ClipByPolygon(int pCount, double *points, int cpCount, double *cPoints, int inside, int *newCount, int **newVcounts, double **newPolygons);
ポリゴンの切断を行う。
pCount 軌跡点の点数

points 軌跡点の座標列 (X,Y,Z･･･の並び)
cpCount 被切断面の点数
cPoints 被切断面の座標列 (X,Y,Z･･･の並び)
inside TRUE:内側の切断 FALSE:外側の切断
newCount 切断後のポリゴン数
newVcounts 切断後のポリゴン頂点数
newPolygons 切断後のポリゴン座標列 (X,Y,Z･･･の並び)
返り値 TRUE:正常　FALSE:切断なし

void u3CalcTcoord(int opCount, double *oPoints, double *oTcoords,int pCount, double *points, double *tcoords);
テクスチャ座標を計算する。
opCount 元の面の頂点数
oPoints 元の面の頂点列 (X,Y,Z･･･の並び)
oTcoords 元の面のテクスチャ座標列
pCount 新しい面の頂点数
points 新しい頂点列 (X,Y,Z･･･の並び)
tcoords 新しい面のテクスチャ座標

int u3CrossCheck2d(double *sv, double *ev, double *csv, double *cev, double *opnt ,int *scross, int *ccross);
交わりの状態をチェックする。
sv 線分１の始点
ev 線分１の終点
csv 線分２の始点
cev 線分２の終点
opnt 交点
scross 線分１の交わり状態
ccross 線分２の交わり状態
返り値 交わりあり:1　交わりなし:0

int u3MakeUVMatrix(double uvOrg[3], double u[3], double v[3], d3Matrix m);
ＵＶマトリクスを作成する。
uvOrg[3] ｕｖ原点
u[3] ｕベクトル
v[3] ｖベクトル

m マトリクス
返り値 成功時１　エラー時０

int u3CrossPoint2d(double p1[2], double p2[2], double p3[2], double p4[2], double op[2], double *s, double *t);
線分と線分の交点を求める（２次元)。
p1[2] 線分１の始点
p2[2] 線分１の終点
p3[2] 線分２の始点
p4[2] 線分２の終点
op[2] 交点
s 線分１の長さに対する始点から交点までの距離の割合
t 線分２の長さに対する始点から交点までの距離の割合
返り値 交点あり:1　交点なし:0

int u3CrossPoint(double l1s[3], double l1e[3], double l2s[3], double l2e[3], double point1[3], double point2[3], double *rate1, double *rate2);
線分と線分の交点を求める（３次元)。
l1s[3] 線分１の始点
l13[3] 線分１の終点
l2s[3] 線分２の始点
l2e[3] 線分２の終点
point1[3] 線分１の交点
point2[3] 線分２の交点
rate1 線分１の長さに対する始点から交点までの距離の割合
rate2 線分２の長さに対する始点から交点までの距離の割合
返り値 交点あり:1　交点なし:0

int u3DecomposePolygonWithWin2d(u3Polygon2d *polygon, int win, u3Polygon2d *winPolygon[], int *tri, u3Poygon2d **triPolygons);
ポリゴンの分割
polygon 被分割ポリゴン
win 分割モード（内/外）
winPolygon[] 窓ポリゴン
tri 分割後のポリゴン数
triPolygons 分割後のポリゴン配列のアドレス

返り値 成功時１　エラー時０

int u3CutPolygon2d(u3Polygon2d *tobeCut, u3Polygon2d *cutPolygon, int win, int *tri,
u3Polygon2d **triPolygons);
ポリゴンの切断を行う。
tobeCut 被切断ポリゴン
cutPolygon 切断ポリゴン
win 切断モード（内/外）
tri 切断後のポリゴン
triPolygons 切断後のポリゴン配列のアドレス
返り値 成功時１　エラー時０

u3Polygon3d *u3PolygonAlloc();
u3Polygon3 構造体の領域を確保する。

int u3DivideTriangle3d(u3Polygon3d *polygon, double area, int *tri,
  u3Polygon3d **triPolygons);
ポリゴンを三角形分割する（2.09 では land.dll に移管）。
polygon 分割するエリア
area エリア
tri 分割後のポリゴン数
triPolygons 分割後のポリゴンのアドレスのアドレス
返り値 成功時１　エラー時０

int u3CombineTriangle3d(int count, u3Polygon3d **polygons, float d_value,
  float r_ratio, int *tri, u3Polygon3d **triPolygons);
ポリゴンを結合する（2.09 では land.dll に移管）。
count ポリゴン数
polygons 結合するポリゴン配列のアドレス
d_value 結合するレートを表す係数
r_ratio 結合共用範囲角度
tri 結合後のポリゴン数
triPolygons 結合後のポリゴン配列のアドレス
返り値 成功時１　エラー時０

int u3Round(int innum, u3Poygon3d **inpoly, double radius, int division_no,

int ednum, u3Edge3d *smoothing_edge, int *outnum, u3Polygon3d **outpoly);
丸め処理をする（2.09には含めない）。
innum 頂点数
inpoly ポリゴン配列のアドレス
radius 半径
division_no 分割数
ednum 頂点数（線データ）
smoothing_edge エッジのアドレス
outnum 点数
outpoly 出力ポリゴン配列のアドレス
返り値 成功時1　エラー時0

（2）データ定義/更新関数

int u3CheckPointInAreaXY(double point[3], int pCount, double *points);
点のエリア内をチェックする
point[3] 点
pCount エリア線の点数
points エリア線の座標列(X,Y,Z･･･の並び)
返り値 エリア内:1　エリア外0

void u3StartRandom(int restart);
ランダム計算を開始する。
restart 1:前回の乱数初期値と同　0:新規乱数（呼び出し時刻を用いてSEEDを更新）

int u3OptimizeQuadXY(double points[12], double texCoord[8]);
四角形の矛盾を除く。
points[12] 頂点
texCoord[8] テクスチャ座標
返り値 正常:4
ＸＹから見て右回り:-4
ＸＹから見て８の字なら左回りの部分を残した三角形にする:3
同一点、同線上の点がある場合は削除する:3～0

int u3TransGrid(int width, int height, double left, double rigth, double bottom,
double top, double zNear, double zFar, int count, int *ipoint2d, float *depth,
double *points);

二次元ポリゴンとデプスバッファを用いた地面の高さから三次元ポリゴンを生成する(nori.dll に移管)。
width バッファの横サイズ
height バッファの縦サイズ
left, rigth X 方向の min､max
bottom, top Y 方向の min､max
zNear, zFar Z 方向の min､max
count 点数
ipoint2d ２次元のポリゴンの座標列 (X,Y,Z･･･の並び)
depth Z 値列 (3×width×height)
points 出力座標
返り値 点数

int u3CrossCheck(int cnt1, ouble *pnt1, int ix1, int cnt2, double *pnt2, int ix2,
    double *opnt);
被切断面ポリゴンと切断面ポリゴンの交点チェックを行う。
cnt1 被切断ポリゴンの点数
pnt1 被切断ポリゴンの座標列 (X,Y,Z･･･の並び)
ix1 被切断ポリゴンの線分の通し番号
cnt2 切断ポリゴンの点数
pnt2 切断ポリゴンの座標列 (X,Y,Z･･･の並び)
ix2 切断ポリゴンの線分の通し番号
opnt 交点があった場合の座標列 (X,Y,Z)
返り値 TRUE:交点あり
FALSE:交点なし

int u3FaceCrossCheck(int out, int cnt1, double *pnt1, int ix1, int cnt2, double *pnt2,
  double **opnt);
被切断ポリゴンと切断ポリゴンの包含チェックを行う。
out TRUE:交点座標列を出力する
FALSE:出力しない
cnt1 被切断歩 ori ゴンの点数
pnt1 被切断ポリゴンの座標列 (X,Y,Z･･･の並び)
ix1 被切断ポリゴンの線分の通し番号
cnt2 切断ポリゴンの点数
pnt2 切断ポリゴンの座標列 (X,Y,Z･･･の並び)

opnt 交点があった場合の座標列 (X,Y,Z)
返り値 交点数

int u3CheckSamePoint(int count, double *inPoints, double ** outPoints);
同一点をチェックし、不要な頂点を削減する。
count 点数
inPoints 入力座標列 (X,Y,Z･･･の並び)
outPoints 出力座標列 (X,Y,Z･･･の並び)
返り値 出力点数

（3）データ取得関数

void u3GetQuadShape(int ix, double *leftPoints, double *rightPoints,
  double points[12]);
２本の点列のそれぞれの ix 番目と ix+1 番目をつないだ四角形を取り出す
ix 点の通し番号
leftPoints 左側の点列 (X,Y,Z･･･の並び)
rightPoints 右側の点列 (X,Y,Z･･･の並び)
points[12] 四角形の座標

void u3GetQuadTcoord(int ix, int uvType, double *centerPoints, double offsetXY,
  double uStart, double vStarts, double uScale, double vScale, double texCoord[8]);
２本の点列の ix 番目と ix+1 番目をつないだ四角形のテクスチャ座標を取り出す
ix 点の通し番号
uvType 縦／横
centerPoints 線 (X,Y,Z･･･の並び)
offsetXY オフセット値
uStart 開始点の u 座標
vStarts 開始点の v 座標
uScale u スケール
vScale v スケール
texCoord[8] テクスチャ座標

int u3Get2dGridPolygon(int width, int height, unsigned char *grids, int xStart,
  int yStart, int *ipoints2d);
ステンシルバッファから島状図形の輪郭線を抽出する。
width バッファの横サイズ

height バッファの格子の縦サイズ
grids バッファのメッシュの値(0/1)(width×height)
xStart, yStart 着目する島の内部点の位置を示す
ipoints2d ポリゴン（格子）座標（2×width×height）の格納先
返り値 ポリゴンの頂点数

int u3GetZLine(d3Face *f, double z, double s[3], double e[3]);
面の高さ Z の水平面との交線を調べる。
f 面のアドレス
z 高さ
s[3] 交線の始点
e[3] 交線の終点
返り値 交差がある 1　無い 0

Ｂ－８．メッセージライブラリ（Ｚ３ＥＲＲ）

（1）メッセージ定義ファイル

E3_BIN_PATH　で設定されているディレクトリ上にある“ERR_MSG.txt”というメッセージ定義ファイルにあるメッセージフォーマットを使用する。
その“ERR_MSG.txt”ファイルのメッセージのフォーマットを以下に示す。

＜ERR_MSG.txt＞

```
E  1001  エラー(%d)
E  1002  アプリケーションエラー_%s
  ・・・・・・・・・・・
W  3001  %s がありません
W  3002  %d の%s が見つかりません
  ・・・・・・・・・・・
I    5001  %s ができません
  ・・・・・・・・・・・
C  7001  %s の削除をしてもいいですか？
```

１カラム目：メッセージタイプを表すアルファベット大文字１文字：
　Ｅ：エラー(error) Ｗ：警告(warning) Ｉ：情報(info) Ｃ：確認(confirm)
　これら以外は無視される。
　基本的な仕分け：

E はシステムに起因するエラー：ユーザーに対する謝罪（バグ、リソース不足など）
W はユーザーに起因するエラー：ユーザーに対する訓示（禁じ手、操作ミスなど）
I はユーザーへの安心感の増進（操作成功の祝辞など、うるさくない範囲で）
C はユーザーに判断を求める場合（疑問手や重大な操作の事前再確認、後始末選択）

２カラム目：スペースを入れる。

３カラム目から４桁の数字：メッセージNo.

1000 番台：エラー　　3000 番台：警告

5000 番台：情報　　7000 番台：確認

※これは整理のためのものであって、表示方法はメッセージタイプが決める。

例えば、5000 番台のメッセージでも、タイプを W とすれば警告

７カラム目：スペース

８カラム目：メッセージ

%d、%f、%s に対応し、%.2f などのフォーマットも指定可能で個数の制限は特にない。

また、スペースを入れたい時は半角アンダーバー（_）で置き換えて入力すること。

制限事項：１行 512byte を越えてはならない。

（２）関数

void z3LoadMessage();

初版において、メッセージ定義ファイルをロードする。

一番最初に呼ぶ。

void z3LoadMessageML(char* filename);

多言語版において、メッセージ定義ファイルをロードする。多言語版において、起動時または言語切換時に呼ぶ。

filename 言語別のエラーメッセージファイル名をフルパスで指定

（例：″c:¥@keikan¥ksim¥Language¥ja¥ERR_MSG.ja.txt″）

void z3CloseMessage();

メモリー上のメッセージテーブルを解放する。プログラム終了時に呼ぶ。

void z3StoreMessage(int no, ...);

メッセージをバッファにセットする。

no メッセージ番号

... メッセージ可変引数部

※表示系（G3DRL）のエラーは、直ちに表示すると多重エラー無限ループに陥りがちである。

int z3FlushMessage(int display);
バッファにセットされているメッセージを表示(非表示)して、メッセージをクリアする 。
display TRUE ならセットしてあるメッセージを表示
FALSE なら非表示
返り値 TRUE ならＯＫ（はい）ボタンが押された
FALSE ならいいえボタンが押された
（メッセージタイプが確認モードのときのみ意味がある）

void z3Message(int no, ...);
no（エラー、警告、情報モード）のメッセージを（直ちに）表示する。
no メッセージ番号
... メッセージ可変引数部

int z3Confirmation(int no, ...);
no（確認モード）のメッセージを（直ちに）表示する。
no メッセージ番号
... メッセージ可変引数部
返り値 TRUE ならＯＫ（はい）ボタンが押された　FALSE ならいいえボタンが押された

void z3GetMessage(char *type, int *no, char *msg);
現在ヒットされているメッセージ情報を取得する。
type,msg 共に呼ぶ側で領域（type[2]、msg[512])を確保しておくこと
type E（エラー）かW（警告）かI（情報）かC（確認）
no メッセージ番号
msg （可変部が埋められている）メッセージ

int z3ShowWindow();
メッセージウインドウを表示する。
中身は、UNIX と WindowsNT とで異なる
ライブラリ以外から呼ばないこと