６．プラグインDLL

６－１．概要

景観シミュレータにはいくつかの枝分かれしたバージョンが存在する[1)]。これらの機能を統合し、全ての機能を使用することができる統合バージョンを作ると、不必要に大きなシステムとなることから、以下の方針を採った。

（１）多くの利用分野に共通する一般的な機能を有する、完成度の高い基幹部分を作成する。この基幹部分には、選択的に利用する専門的な機能を、プラグインとして必要となった時点でメニュー一覧から選択し、動的に追加できるインターフェースを用意する。

（２）枝分かれバージョンのそれぞれが有する、専門性の高い機能を、プラグインDLLとして分離し、必要となった時点で基幹部分に付加できる形とする。

（３）プラグインDLLからは、基幹部分における編集対象となっている全てのデータ（最も重要なものは、ジオメトリのルートなど）へのアクセスを可能とする。

（４）プラグインDLLのコーディングのために、基幹部分が有するスタティックなライブラリ関数の使用が可能とする。

（５）プラグインDLLは、ユーザーが終了を選択し、処理が終了した時点でアンロードする（占有していたメモリを解放する)。

（６）基幹部分の側から、開いているプラグインを閉じるためのプロトコルを用意する。

６－２．基幹部分のライブラリ関数の提供

基幹部分がビルドにおいてリンクしている、スタティック・リンク・ライブラリの諸関数を、DLL側にエクスポートされる関数として再定義した。このために、各ライブラリのヘッダー(d3dml.h, g3drl.h 等)をそれぞれ二つのヘッダー・ファイルに分割し、関数の宣言を行っている部分を、d3symbol.h, g3symbol.h 等の名称の別ファイルに分離した上で、関数宣言についてコンパイル・スイッチにより、DLL側に対してエクスポートされるように宣言している。この宣言は、スタティック・リンク・ライブラリ（dml.lib, drl.lib 等）のビルド時、およびこのライブラリへのスタティック・リンクを含む基幹部分(sim.exe)のビルド時に、二度使用する。基幹部分が、_declspec(dllexport) 宣言によりエクスポートされる関数を有する結果、sim.exe のビルドに際して、sim.lib というライブラリが生成される。この中に、エクスポートされる（即ちプラグインDLLの側から利用することができる）関数に関する情報が含まれている。従って、プラグインDLLをビルドする際に、スタティック・リンク・ライブラリをリンクする必要はない。また仮にそのようなビルドを行うと、ライブラリ中のスタティック変数やグローバル変数の動作が問題となる危険性がある。

なお、注意しなければならない点は、関数のDLLエクスポートを前提としていなかった過去のバージョンのビルドにおいては、ソースコードによっては、当該ソースコードで定

義された関数を宣言するヘッダー・ファイルを冒頭でインクルードしていないものがあった、という点である。ビルドを通すだけのためには、自分自身を宣言する必要は無いが、それらの関数を DLL エクスポートするためには、宣言が必要である。従って、エクスポート関数を含むソースには冒頭部分でエクスポートを宣言するヘッダーをインクルードしておく必要がある。この点が原因となるリンク・エラーは解決に時間を要する場合がある。

リスト６－１：DLL エクスポートするライブラリ関数のヘッダー例

[例]d3dml.h の関数宣言の部分を、d3symbol.h として分離した（末尾部分）

```
・・・・(前略)・・・・
};
struct _d3PickPath {
/*private:*/
/*public:*/
        d3TravInfo *trav;
        int num;
};

#include "d3symbol.h" /*050418従来は全てこのヘッダー内容を直接個別に宣言していた*/
#ifdef __cplusplus
}
#endif
#endif  /* !__D3DML_H__ */
```

[例]d3symbol.h における DLL エクスポートの関数宣言例

```
#ifdef SIMDLL
#define a __declspec(dllexport)
#else
#define a
#endif

/* functions in d3dml.c プリプロセッサでSIMDLLが定義されていれば、DLLエクスポートされる関数として宣言される。そうでなければ、従来通り*/
a double inpo(double A[], double B[]);
a void expo(double A[], double B[], double H[]);
a int ketemu(double A0[], double A1[], double B0[], double B1[]);
a int d3CalcBoundingBox(d3Group *g);
/* 注）別の目的でシンボル「a」を用いている箇所はコンパイル・エラーを起こすので修正する必要がある。例：
NG: void SetColor(float r, float g, float b, float a);
OK: void SetColor(float r, float g, float b, float aa);  */
```

一方、同じヘッダーを DLL 側で利用する際には、SIMDLL というプリプロセッサを定義しないでコンパイルを行い、シンボル部分を通常の関数として宣言すると共に、リンク・ライブラリに sim.lib を加えることにより、プラグイン DLL が起動した時点で sim.exe に含まれるこれらの関数がダイナミックにリンクされ、それらを使用することができる。

プラグイン DLL は、独立性の高い外部関数とは異なり、起動後は基幹部分の一部を構成

するダイアログと同等に基幹部分の側のデータにアクセスし、削除・修正を含む様々の処理を行うことができる。この処理のために基幹部分のライブラリ関数を利用することもできる。従って、様々な機能・形態のものが作成可能であると言える。

６－３．基幹部分でのプラグイン DLL の管理機能の作成

プラグイン DLL を管理する各種機能を整理するために、基幹部分側に CPlugin というクラスを新たに作成した。この中では以下の処理を行っている。

① ライブラリ関数の一覧を記載した plugin.tab（テキスト・ファイル）を読み込みメニューに追加する。

② ユーザーが、メニューからあるプラグインを選択した場合、当該プラグイン DLL をロードし、エントリー・ポイント関数を呼び出す。

③ ユーザーが、プラグイン DLL の側での操作の中で「終了」を選択した場合に、CPlugin の中の終了処理を要求する。この終了処理の中では、プラグイン DLL のアンロードのためのタイマー割り込みによるスレッドを立て、DLL の全てのダイアログが閉じたことを確認の上でアンロードを行う。

④ ユーザーが基幹部分において、DLL の終了を選択した場合に、DLL 側に対して終了処理を行うようにリクエストする。

なお、プラグイン DLL の完成度が低く、その中でアクセス違反等が生じた場合には、操作中であった基幹部分も含め、アプリケーションが異常終了することは避けられない。また、DLL の中でメモリー・リークが生じた場合には、基幹部分が終了するまで解消しない。この２点は、CreateProcess 関数により起動される外部関数とは異なっている。

この他に、CPlugin から各プラグイン DLL を起動する際には、display_dialog という共通のエントリー・ポイントを設けている。これにより、基幹部分側の重要な情報を引数として渡している。更に、終了時点での制御にもこの関数を用いている。

６－４．プラグイン DLL と基幹部分のインターフェースの詳細

プラグイン DLL は、ロードされた後に起動される display_dialog(CMainFrame*) という関数を用意していなければならない。この関数は、_declspec(dllexport)宣言により、基幹部分側に対してエクスポートされるように定義する。基幹部分は、プラグイン DLL をロードした後に、この関数に引数として、基幹部分側の pMainFrame ポインタを渡す。プラグイン DLL の側では、このポインタを手掛かりとして、基幹部分側のデータにアクセスする。

終了時点のアンロード処理は、起動時よりも条件がシビアとなる。基幹部分側からプラグイン DLL のアンロードを実行した時点で、プラグイン側が起点となるスレッドが何か動いていると、アクセス違反が生じ、エラー終了となる。そこで基幹部分側では、プラグイン DLL が既に終了していることを確認したうえで、アンロードしなければならない。アン

ロードは基幹部分側の関数をスレッドの起点として実行するため、プラグイン DLL 側で基幹部分側のアンロードを実行する関数を起動するわけにはいかない。

そこで、本システムにおいては、タイマー割り込みを使用して基幹部分側に別スレッドを発生させ、そのコールバックの中でアンロードを実行することとした。

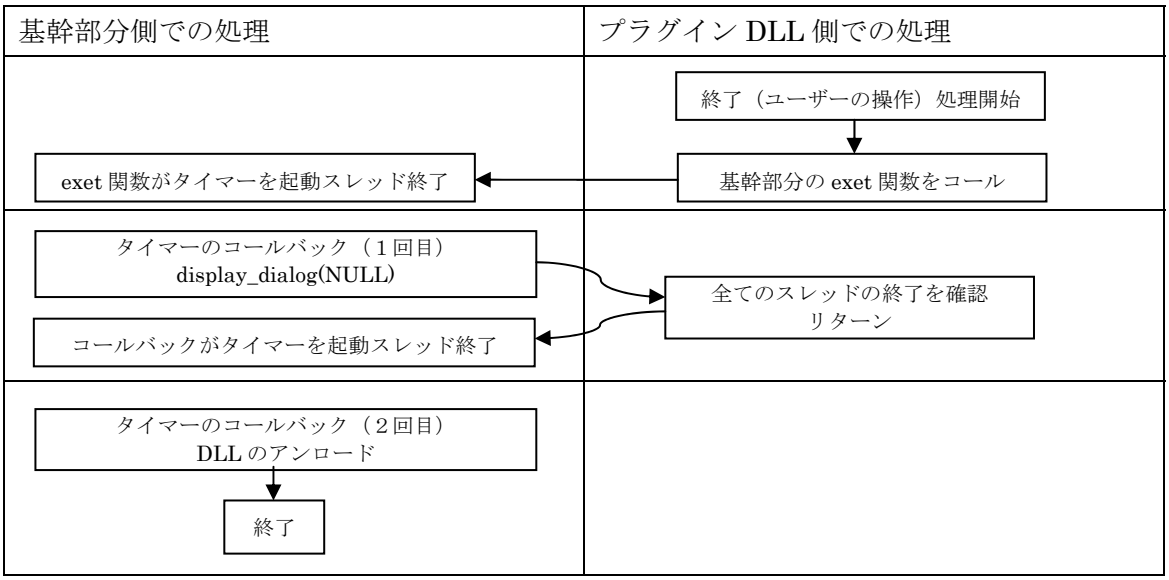


図６－１：プラグイン DLL 側からの終了要求に基づく、アンロードまでの処理

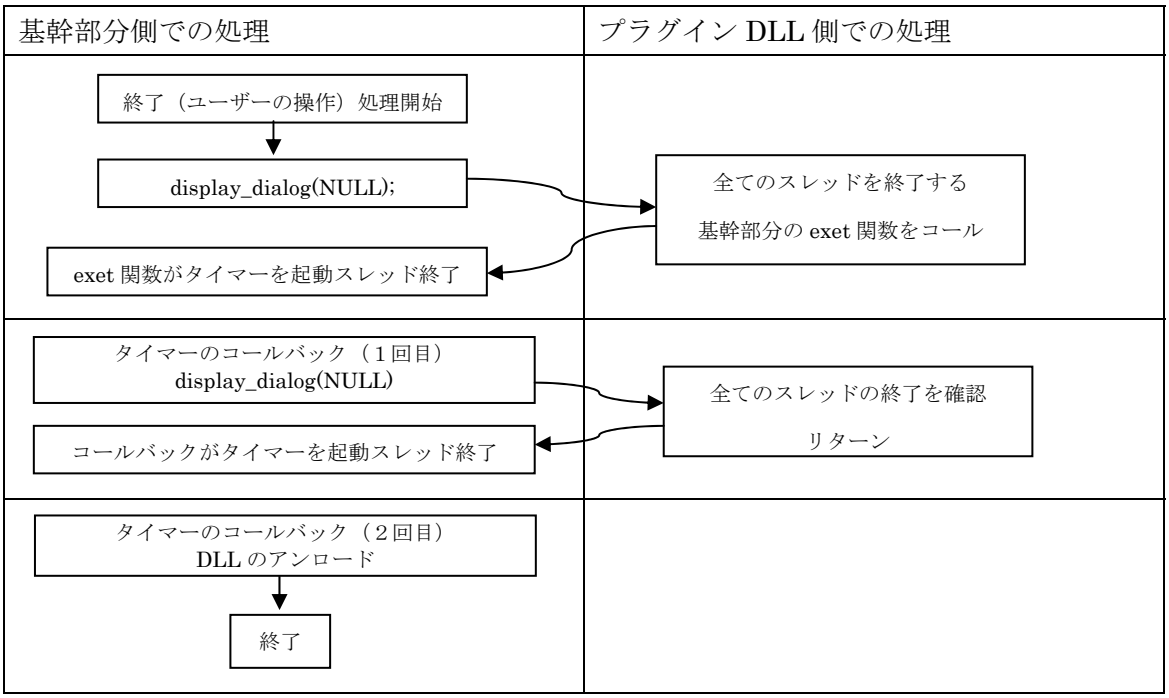


図６－２：基幹部分側からの終了要求に基づく、アンロードまでの処理

終了時点では、この引数が NULL だった場合には、基幹部分側からの終了リクエストと解釈して、終了処理に入る。その終了処理の中では、基幹部分側に対して、アンロード要求を出す。基幹部分側では、DLL からのアンロード要求を受けない限り、アンロードを実行しない。また、DLL 側から基幹部分に対してアンロード要求が出された場合においては、

基幹部分側でアンロードを行う直前に再確認のために display_dialog(NULL)が呼び出される。DLL 側では、全てのダイアログが閉じていることを確認の上、リターンする。

基幹部分側（CPlugin）では、アンロード要求を受けると、タイマーをセットする。実際のアンロードは別スレッドであるタイマーのコールバック関数の中で実行する。このコールバック関数においては、再度 DLL 側の display_dialog 関数を引数 NULL で呼び出し、全てのダイアログが閉じていることを確認した上で、アンロードを行う。

なお、CPlugin に関して、以下の２点を改良する余地がある。

①複数のプラグインが起動できるようにする

現在実装例としているプラグイン DLL はいずれも地形編集機能であるため、二つのプラグインが同時に起動して相互に干渉すると、不具合が生じる場合がありうる。そこで、プラグインを起動しようとした時点で実行されている別のプラグイン DLL が存在すれば、基幹部分の側から終了要求を出すようにしている。将来的には複数の性格が異なるプラグインが協調して動作するようなことも考えられるので、そのような場合には複数のプラグイン DLL の同時起動の仕組も検討する必要が生じるかも知れない。

②プラグインからの終了要求は、SendMessage 関数による CMainFrame への通知を用いている。

６－５．サンプル実装したプラグイン DLL

従来の枝分かれバージョンから、以下の機能を取り出し、プラグイン DLL の実装例とした。

（１）地形編集機能（land.dll）

起動した時点で、機能選択画面が開く。ユーザーはラジオボタンで、標高面作成、頂点移動、地形切断、地形の細分化と最適化、側面と底面の追加のいずれかの機能を選択する。各機能において終了することにより、プラグイン DLL 全体が終了する。

A. 機能選択画面

Land.dll は、地形編集に関するいくつかの機能をまとめた DLL である。プラグインを起動すると最初に機能選択画面を表示し、ここから個別の機能に分岐する。また、それぞれの機能を選択したうえで HELP により各機能の解説を読むことができる。

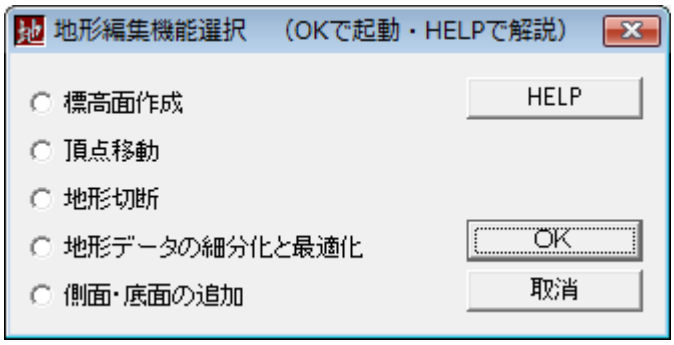


図６－３：地形編集機能選択画面

リソース：IDD_LAND_DLG_MENU ハンドラ：CLandMenu (LandMenu1.cpp)

ヘルプ：land.txt

B. 標高面作成

指定した高さの標高面を作成する。山頂の場合は、地面の下に形成され、窪地の場合は地面の上の水平面となる。処理に先立ってエリア指定を行うと、その範囲の標高のレンジを表示する。そのレンジの中での、面を作成する標高を指定し実行する。

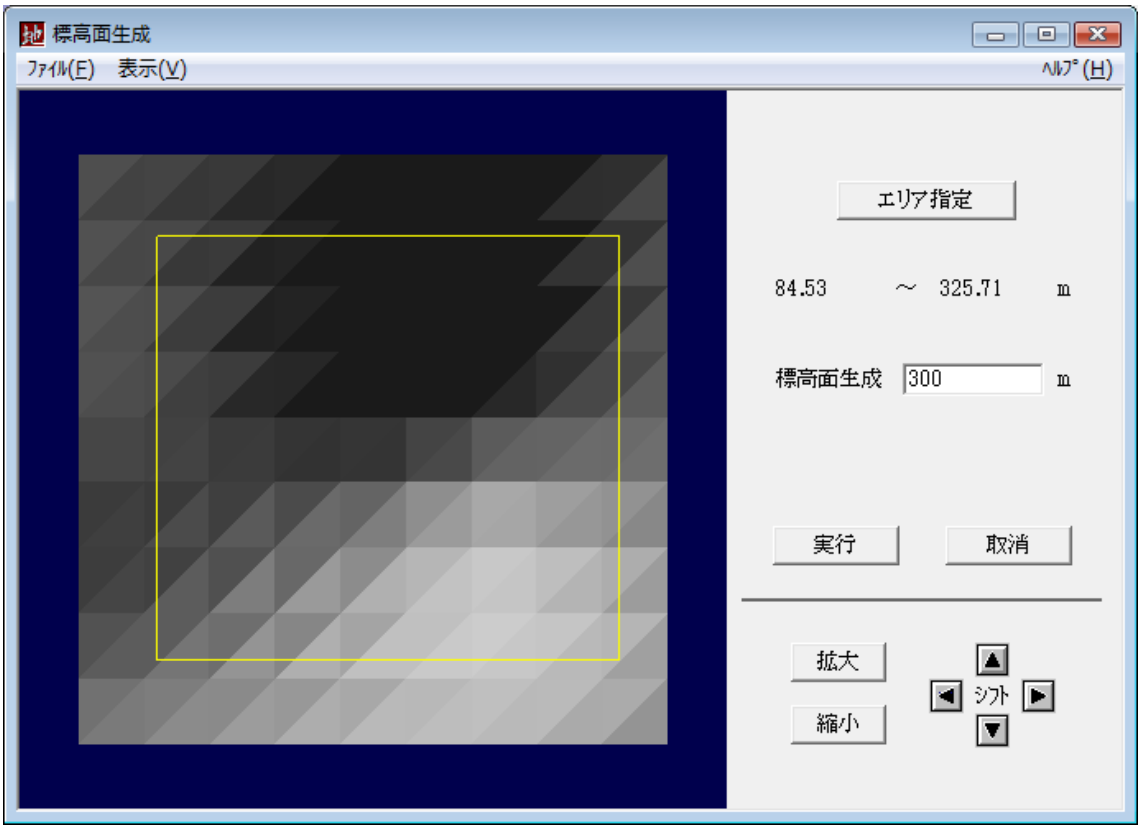


図６－４：標高面生成画面

①メニュー：IDR_MENU_LAND_HAICHI

メニュー構成 ［ファイル(F)］ ＋［閉じる］ダイアログを終了する ［表示(V)］ ＋［グリッド］５０m間隔のグリッドを表示する ＋［メジャー］縮尺を表示する ［ヘルプ］

②OpenGL 画面：CMyOrthoVW (myorthovw.cpp)

③右側ダイアログ：

リソース：IDD_LAND_DLG_HYOUKOU

ハンドラ：CHyoukouWnd (hyoukouwnd.cpp)

④ヘルプ：hyoukou.txt

C. 頂点移動

自由曲面の編集を指向した機能。変形の中心や、影響範囲を指定することにより、連続的に地形の一部を盛り上げたり、水平移動したりすることができる。

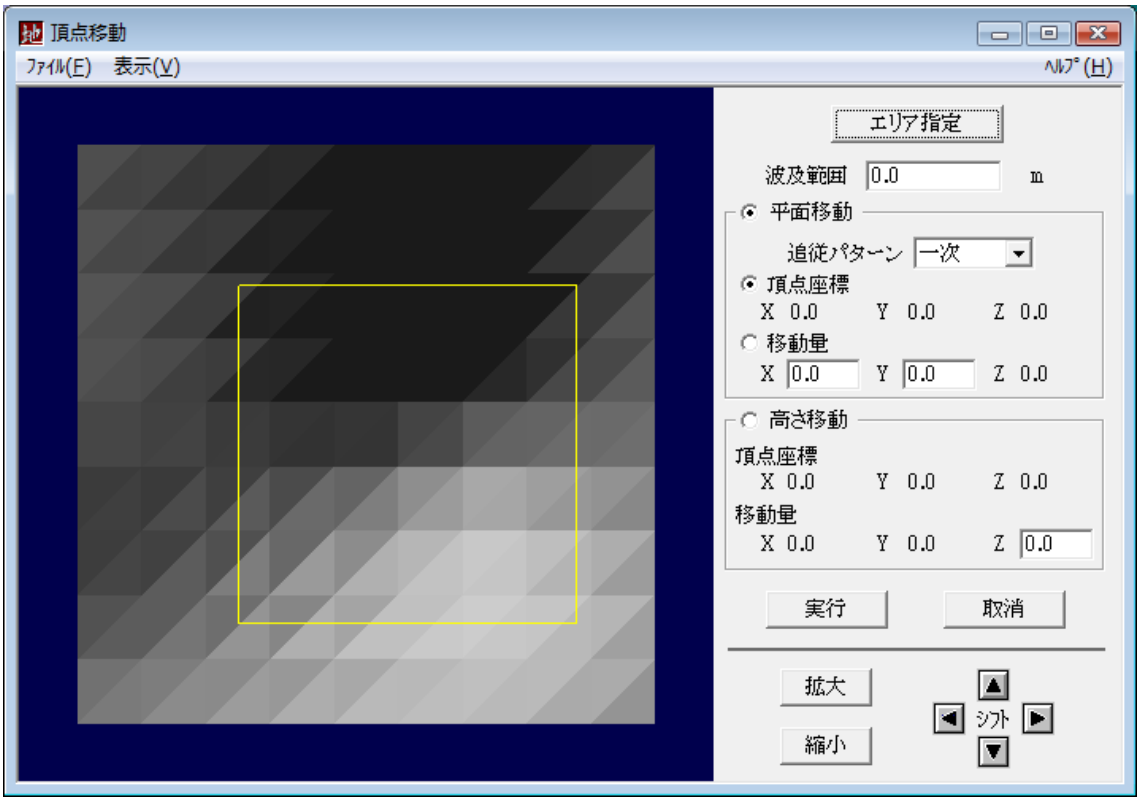


図６－５：頂点移動画面

① メニュー：IDR_MENU_LAND_HAICHI

ハンドラ：CPointMoveWnd (pmovewnd.cpp)

メニュー構成 ［ファイル(F)］ ＋［閉じる］ダイアログを終了する ［表示(V)］ ＋［グリッド］５０m間隔のグリッドを表示する ＋［メジャー］縮尺を表示する

② OpenGL 画面：CMyOrthoVW::m_orthoView (MyOrthovw.cpp)

③ 右側ダイアログ

リソース：IDD_LAND_POINT_MOVE　CDialogBar::m_Prm_Bar

ハンドラ：CPointMoveWnd (pmovewnd.cpp)

⑤　ヘルプ：pmove.txt

D. 地形切断

画面操作で多角形を指定し、その範囲内の地形を切り抜く。

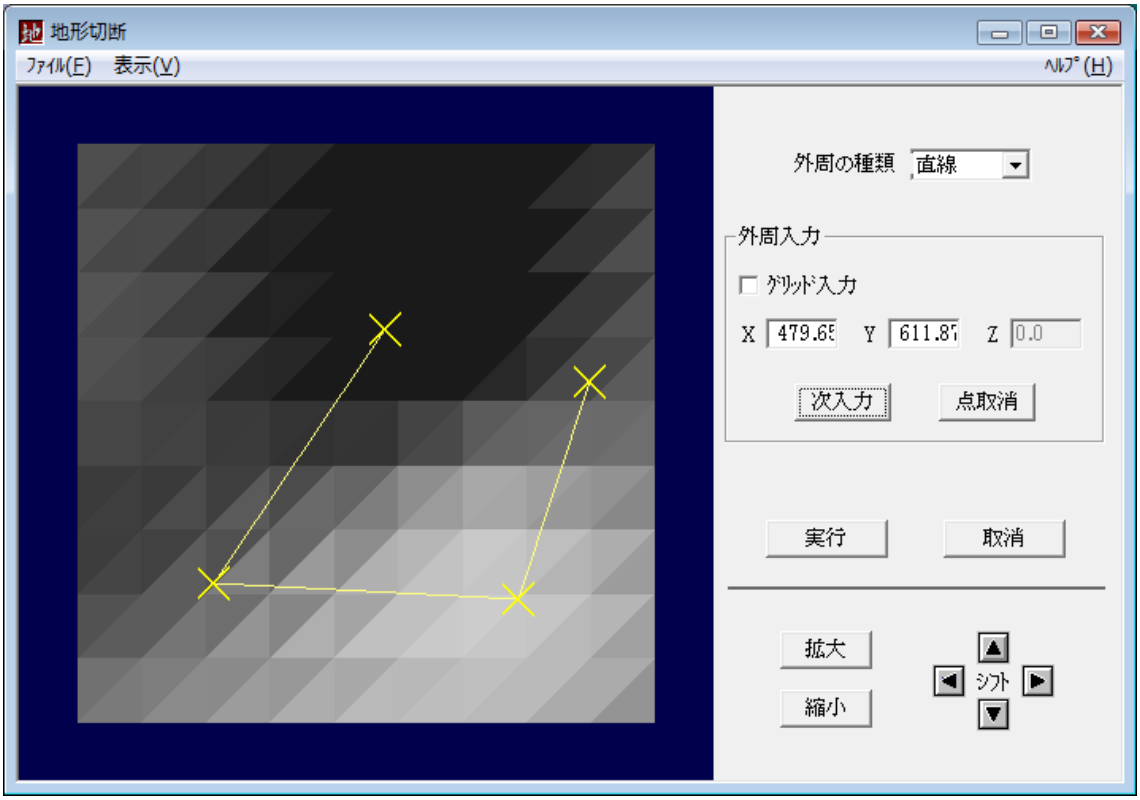


図６－６：地形切断画面

①メニュー：IDR_MENU_LAND_HAICHI

ハンドラ：CTopoCutWnd (topocutwnd.cpp)

メニュー構成
［ファイル(F)］
＋［閉じる］ダイアログを終了する
［表示(V)］
＋［グリッド］５０ｍ間隔のグリッドを表示する
＋［メジャー］縮尺を表示する

②OpenGL 画面：CMyOrthoVW::m_orthoView (myorthovw.cpp)

③右側ダイアログ：

リソース：IDD_LAND_DLG_TOPO_CUT

ハンドラ：CDialogBar CPointMoveWnd::m_Prm_Bar (topocutwnd.cpp)

④ヘルプ：topocut.txt

E. 地形データの細分化と最適化

地形を構成する三角形分割に対して、統計的・大局的な加工を行い、細分割したり、平坦に近い領域で面の統合を行ったりする。

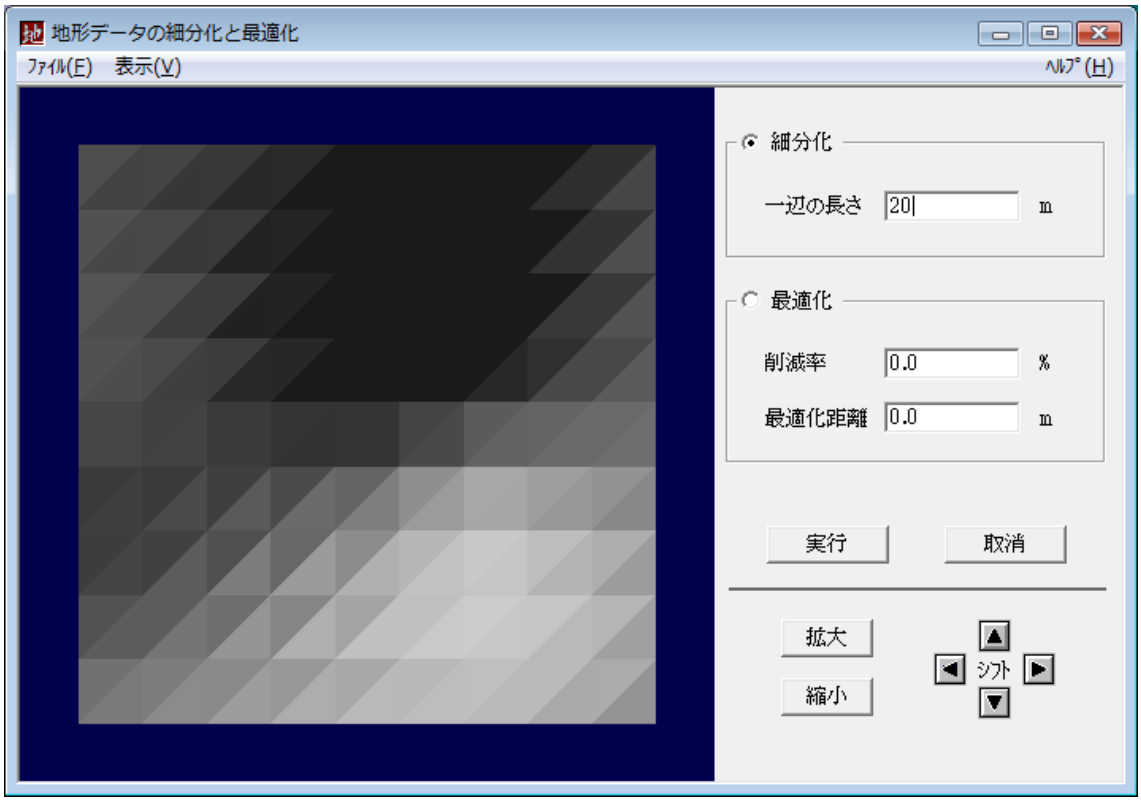


図６－７：地形データの細分化と最適化画面

① メニュー：IDR_MENU_LAND_HAICHI

メニュー構成 ［ファイル(F)］ ＋［閉じる］ダイアログを終了する ［表示(V)］ ＋［グリッド］５０m間隔のグリッドを表示する ＋［メジャー］縮尺を表示する

② OpenGL 画面：CMyOrthoVW*m_orthoView (orthovw.cpp)

③ 右側ダイアログ：リソース：IDD_LAND_TOPO_EDIT

ハンドラ：CTopoEditWnd (topoeditwnd.cpp)

メンバ変数 CDialogBar m_Prm_Bar

④ ヘルプ：topoedit.txt

F. 側面と底面の追加／削除

地表面だけから成る地形データに、側面や底面を付加して、BOOL 演算が行いやすい閉じた閉多面体を作る。合わせて、地形の体積や表面積などの諸元素を計算し表示する。

図６－８：地形諸元／側面底面追加画面

①リソース：IDD_LAND_DLG_SOLIDANAL

②ハンドラ：CSolidanal (kabe.cpp)

③ヘルプ：kabe.txt

（２）園路生成機能（parkroad.dll）

地物を平面図表示した画面表示の上で、地形の上に園路を構築する。この処理は、図形演算を用いて地形をカットする処理を含んでいる。機能的には高度であるが、ダイアログとしては一つだけのシンプルな構成である。

A. メイン画面

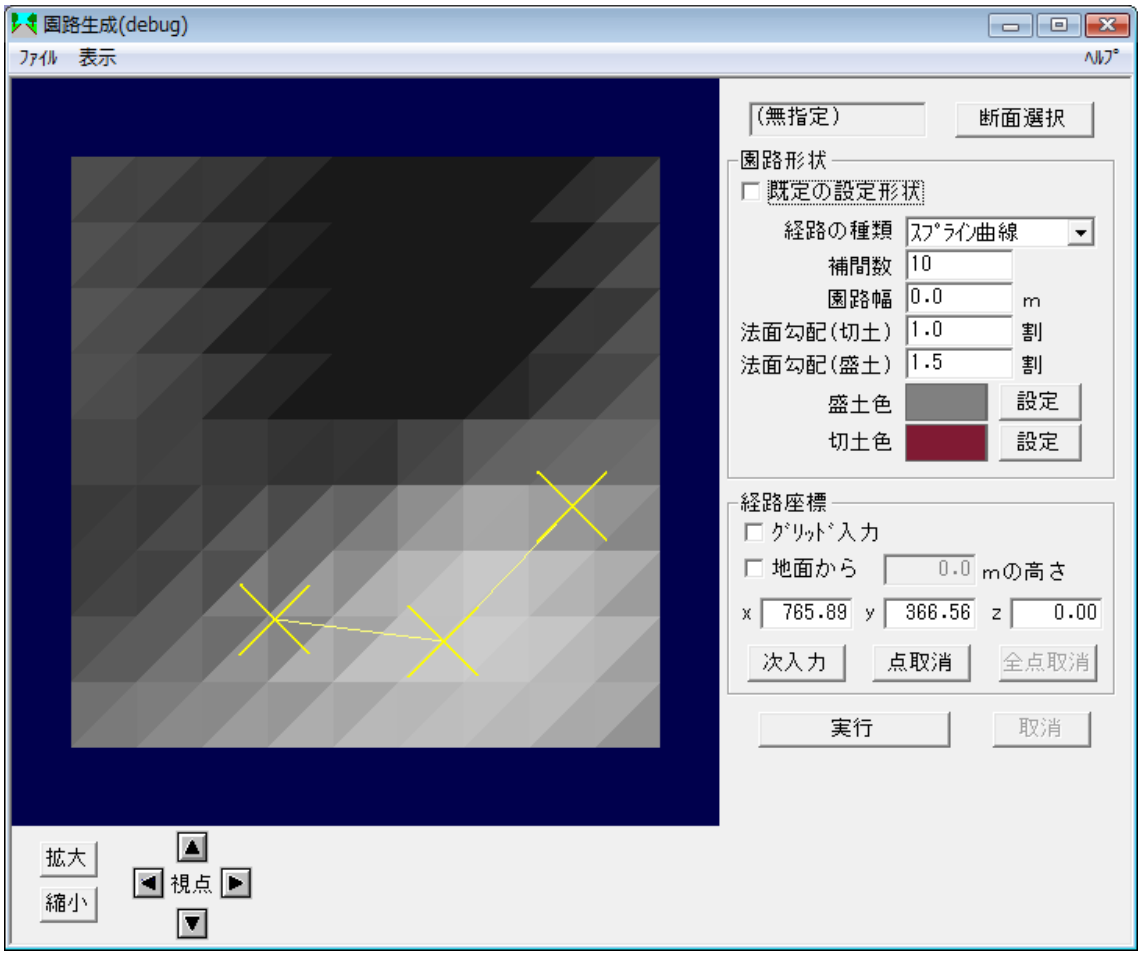


図６－９：園路生成画面

① メニュー：IDR_MENU_ENRO CParkRoadWnd (parkroadwnd.cpp)

② OpenGL 画面：CMyOrthoVW*m_orthoView(myorthovw.cpp)

③ 盛土色の小さな OpenGL 画面：CEnroNoriColWnd m_MoriColWnd (enronoricol.cpp)

④ 切土色の小さな OpenGL 画面：CEnroNoriColWnd m_KiriColWnd (enronoricol.cpp)

⑤ 右側ダイアログ：リソース IDD_DLG_ENRO_PARAM

ハンドラ：CDialogBar*m_ParamBar (parkroadwnd.cpp)

⑥ 下側ダイアログ：リソース IDD_DLG_ENRO_VIEW_CTL

ハンドラ：CDialogBar*m_VwCtrlBar (parkroadwnd.cpp)

⑦ ヘルプ：parkroad.txt

メニュー構成

［ファイル(F)］
＋［経路の新規作成］入力済みの経路を削除し新たな経路を作成する
＋［経路の読み込み］経路をファイルから読み込む
＋［経路を上書き保存］入力したファイル、先刻保存したファイルに上書き保存
＋［経路に名前を付けて保存］新しいファイルに経路を保存する
＋［園路生成の終了］ダイアログを終了する
［表示(V)］
＋［グリッド］５０m間隔のグリッドを表示する
＋［メジャー］縮尺を表示する
＋［全体視界］全体を表示する
＋［表示モード］
＋＋［テクスチャ表示］テクスチャを表示する
＋＋［シェーディング表示］テクスチャなしで面を表示
＋＋［ワイヤーフレーム表示］ワイヤーフレーム表示

B. 園路断面選択

Kdb/Geometry ディレクトリに置かれた ENRO_SEC.set という定義ファイルに登録されている LSS-G 形式（拡張子.geo）の断面定義ファイルの一覧を表示し、ユーザーの選択を求める。

図６－１０：園路断面ファイル選択画面

リソース:IDD_ENRO_DLG_POTONGAN　ハンドラ: CRoadPotongan (roaddan.cpp)

C. カラー、マテリアル、テクスチャ選択

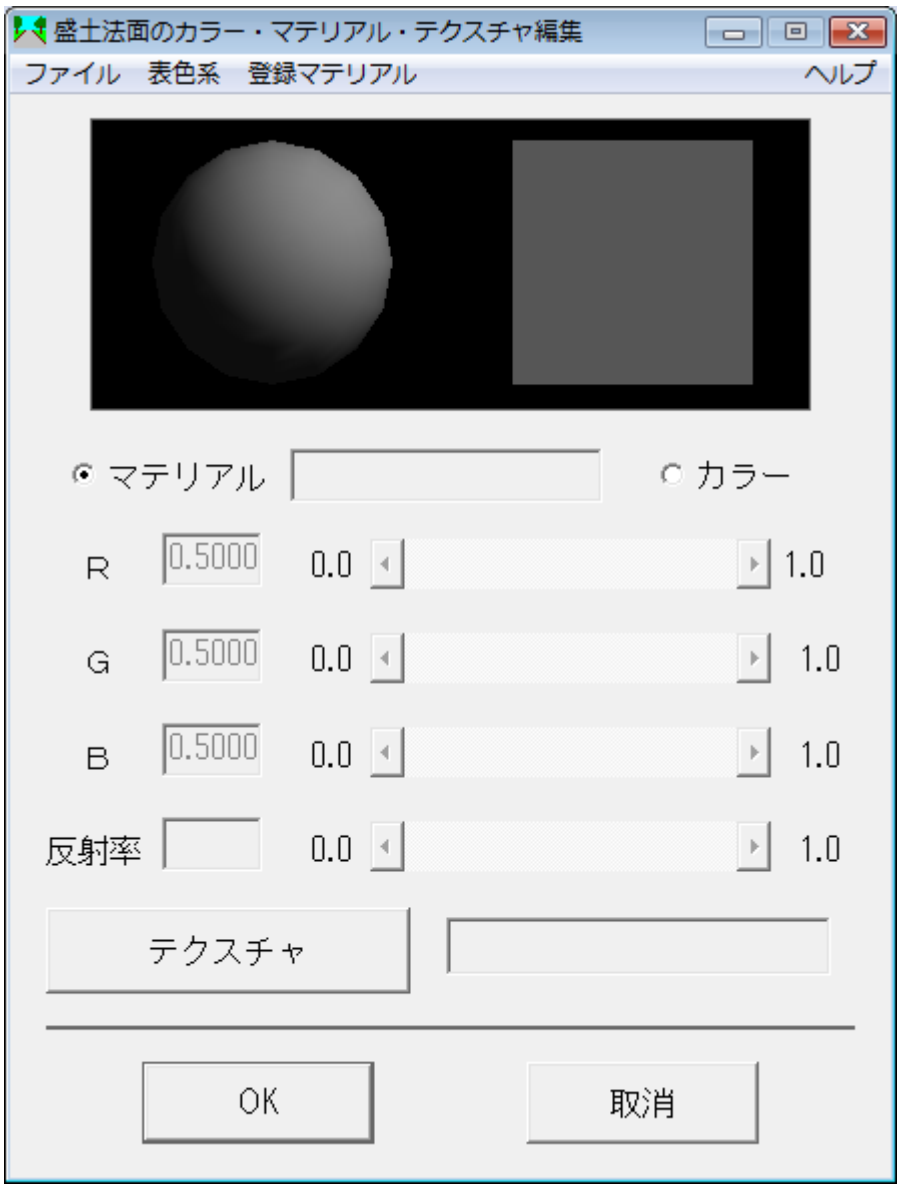


図６－１１：法面のカラー・マテリアル・テクスチャ設定画面

①メニュー：IDR_MENU_ENROMAT (enromat.cpp)

メニュー構成
［ファイル］
＋［閉じる］
［表色系］
［登録マテリアル］
＋［マテリアル選択画面］
＋［マテリアルファイル選択］
＋［選択済マテリアルファイル］

＋［指定マテリアル内容表示］

②OpenGL 画面：CNori2Dlg m_dlg (enromatlist.cpp)

③リソース：IDD_DIALOG_ENROMAT

ハンドラ：CEnroMat (enromat.cpp)

D. テクスチャ選択画面

［テクスチャ］ボタンで起動する。

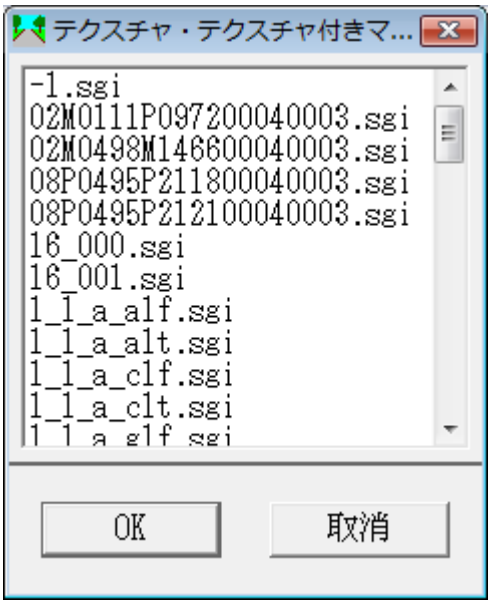


図６－１２：テクスチャ選択画面

リソース：IDD_DLG_TEXLIST ハンドラ：CEnroTexList (enrotexlist.cpp)

E. マテリアル選択画面

メニューの［登録マテリアル］［マテリアル選択画面］で開く。

図６－１３：マテリアル選択画面

リソース：IDD_DLG_NORI_MAT_LIST　ハンドラ：CEnroMatList (EnroMatList.cpp)

（３）道路法面生成機能（nori.dll）

古いバージョンの sim.exe に含まれていた機能を分離独立させたものである。道路法面の形状計算に、OpenGL のデプス・バッファを用いているため、細部を接近して見ると粗い場合がある。

A. メイン画面

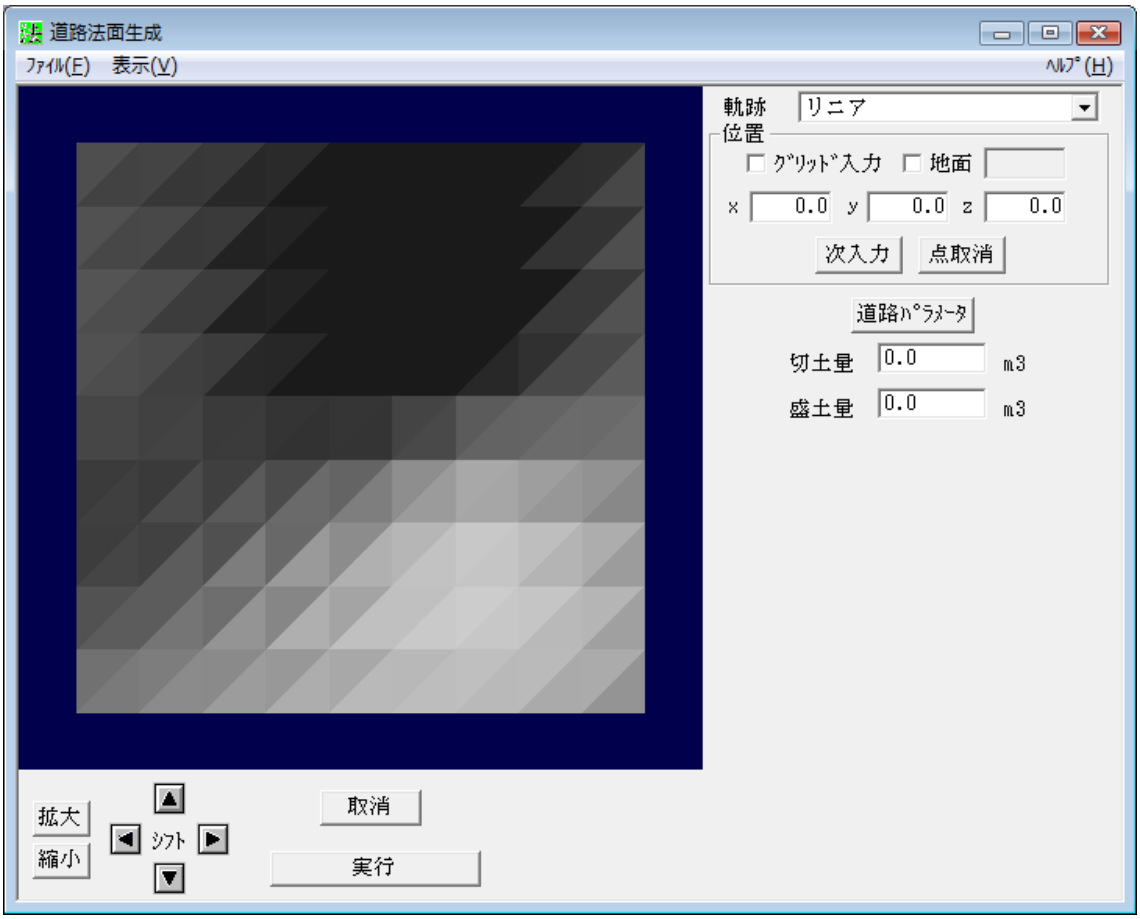


図６－１４：道路法面生成画面

① メニュー：IDR_MENU_HAICHI

メニュー構成
［ファイル(F)］
＋［閉じる］ダイアログを終了する
［表示(V)］
＋［グリッド］５０m間隔のグリッドを表示する
＋［メジャー］縮尺を表示する
［ヘルプ］ヘルプを表示する

② OpenGL 画面：CMyOrthoVW CNoriWnd::m_orthoView (noriwnd.cpp)

③ 右側ダイアログ：リソース:IDD_DLG_NORI_PRM
CDialogBar m_Prm_Bar (noriwnd.cpp)
④ 下側ダイアログ：リソース:IDD_DLG_NORI_CTL
CDialogBar m_Ctl_Bar (noriwnd.cpp)
⑤ ヘルプ：noriwnd.txt

B. 道路パラメータ設定画面

メイン画面の、［道路パラメータ設定］ボタンで開く(m_dlg)。

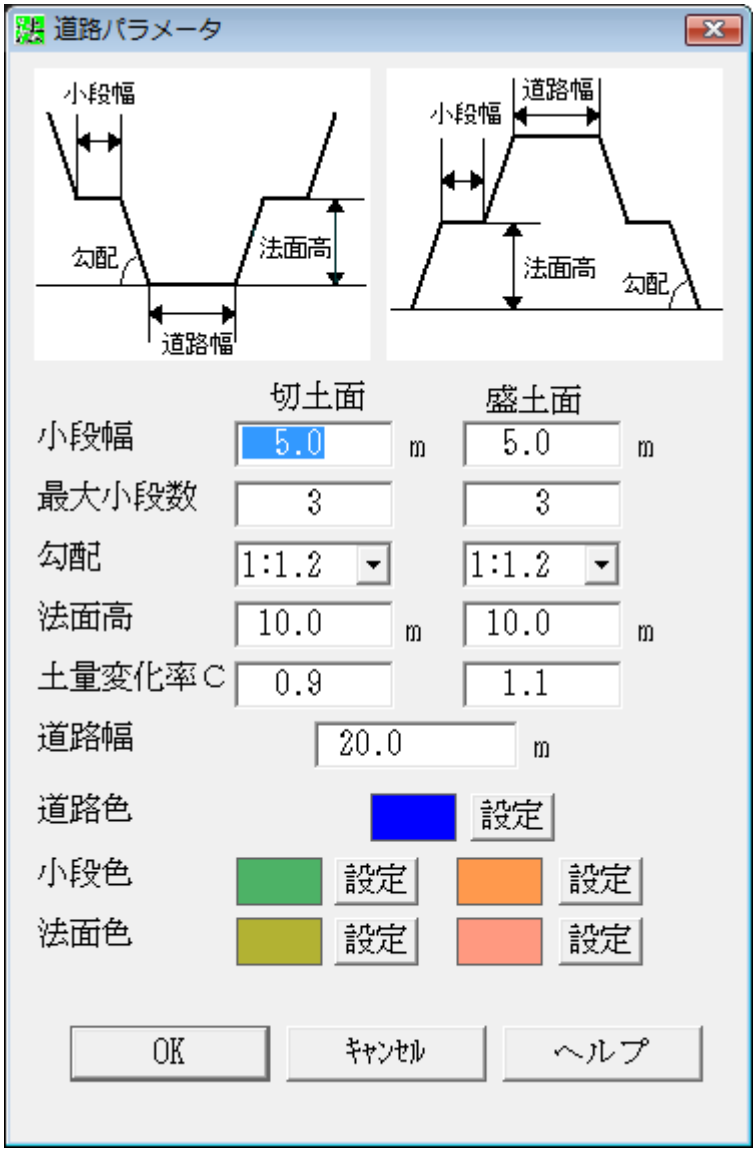


図６－１５：道路パラメータ設定画面

① リソース：IDD_DLG_ROAD_PARAM
② ハンドラ：CNoriDlg (noridlg.cpp)
③ 道路色 OpenGL 画面：CRoadColWnd m_RoadCol_Wnd (roadcol.cpp)

④ 切土小段色 OpenGL 画面：CKShoColWnd m_KshoCol_Wnd (kshocolw.cpp)
⑤ 切土法面色 OpenGL 画面：CKNoriColWnd m_KnoriCol_Wnd(knoricol.cpp)
⑥ 盛土小段色 OpenGL 画面：CMShoColWnd m_MshoCol_Wnd (mshocolw.cpp)
⑦ 盛土法面色 OpenGL 画面：CMNoriColWnd m_MnoriCol_Wnd(mnoricol.cpp)
⑧ ヘルプ：noridlg.txt

C. マテリアル選択画面

道路パラメータ設定画面の、各部分の［設定］ボタンで開く。それぞれの部分のマテリアルを設定する。

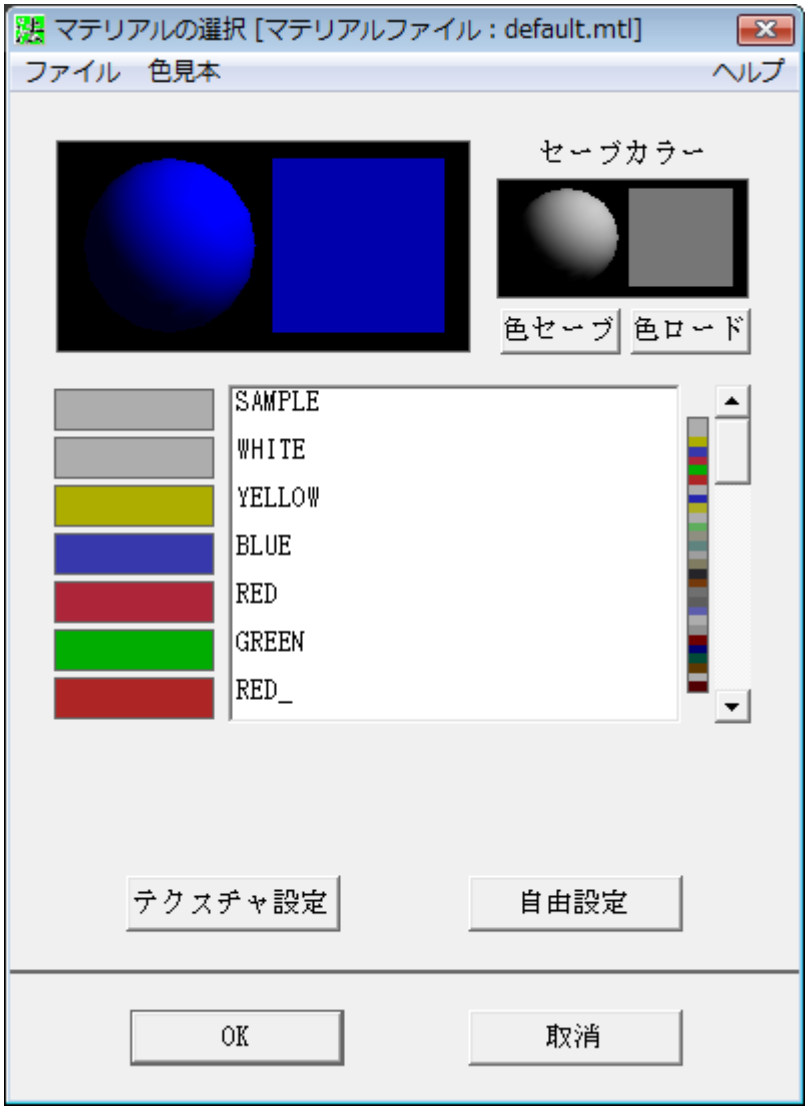


図６－１６：道路マテリアル設定画面

① メニュー：IDR_MENU_MATERIAL2

メニュー構成 ［ファイル］

＋［閉じる］ダイアログを終了する
［色見本］使用可能なマテリアルファイルの一覧をサブメニューとして表示する

② リソース：IDD_DIALOG_MATERIAL ハンドラ：CEditMaterial(editmat2.cpp)

③ 左色球 OpenGL 画面：CSphereDlg m_KyuDlg (spheredlg.cpp)

④ 右セーブカラー色球 OpenGL 画面：CSphereDlg m_saveKyuDlg(spheredlg.cpp)

⑤ 左側 7 箱 OpenGL 画面：CMatFrm m_colFrm[7] (matfrm.cpp)

⑥ 右側櫛の歯 OpenGL 画面：CColorSashDlg m_obiDlg (colsashdlg.cpp)

⑦ ヘルプ：editmate.txt

このダイアログは、基幹部分にも存在するが、これを兼用することにより編集結果の適用先をオプショナルなダイアログに設定することを避け、同じダイアログのコピーを道路法面生成にも独自に用意した。

D. カラーの自由設定

マテリアル編集画面の［自由設定］ボタンで開く

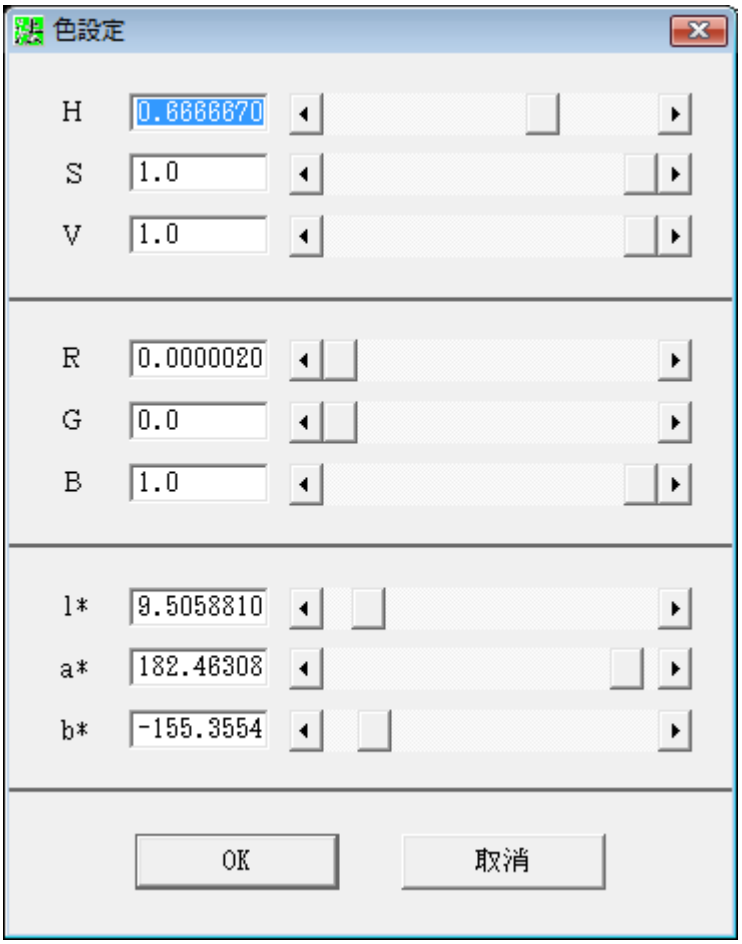


図６－１７：道路カラー設定画面

リソース：IDD_DIALOG_COLOR_SET ハンドラ：CColorSet(colorset.cpp)

E. テクスチャ設定画面

マテリアル編集画面の［テクスチャ］ボタンから開く。

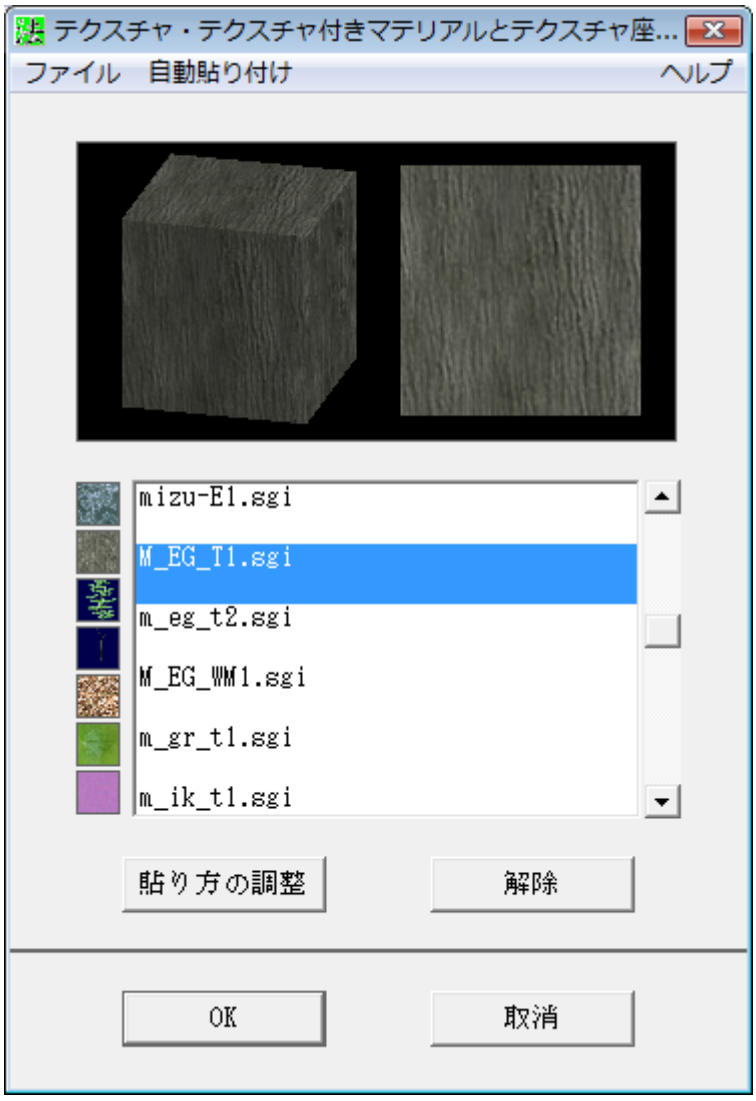


図６－１８：道路テクスチャ設定画面

① メニュー：IDR_MENU_TEXTURE2

メニュー構成 ［ファイル］ ＋［閉じる］ ［自動貼り付け］autotex.set ファイルで定義したメニューを表示する

② 上の OpenGL 画面：CCubeDlg m_boxDlg (cubedlg.cpp)

③ 左の 7 箱の OpenGL 画面 CTexFrm m_texFrm[7] (texfrm.cpp)

④ ダイアログ

リソース：IDD_DIALOG_TEXTURE ハンドラ:CEditTexture (edittex2.cpp)

⑤ ヘルプ：edittex2.txt

F. 貼り方の調整

テクスチャ編集画面の［貼り方の調整］ボタンで開く。

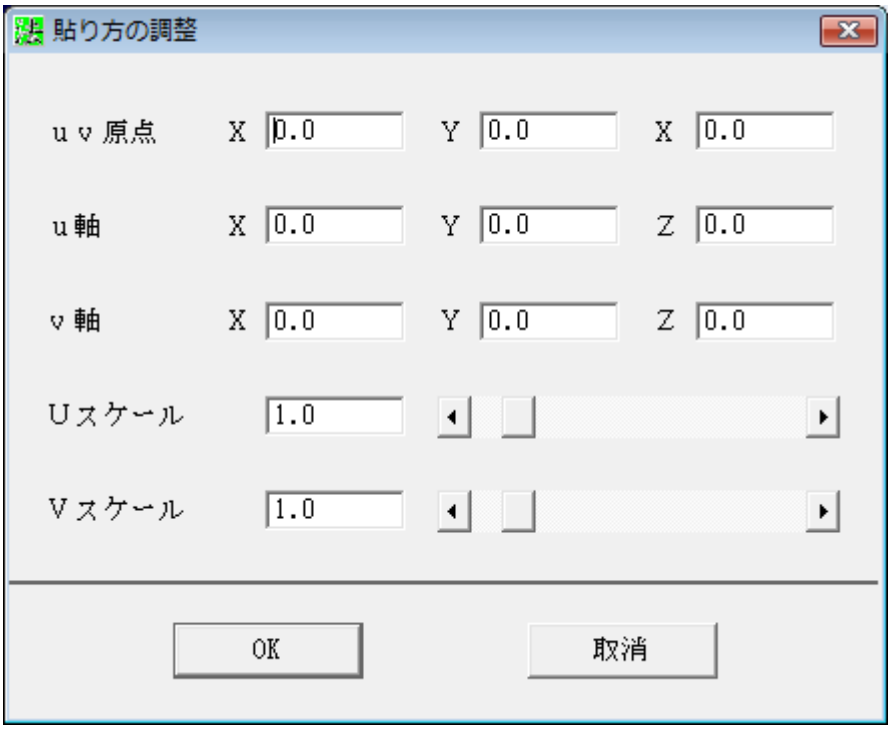


図６－１９：道路テクスチャの貼り方の調整画面

ダイアログ：IDD_DIALOG_TEXTURE_MAP ハンドラ：CTextureMap (textmap.cpp)

（４）トンネル生成機能(tunnel.dll)

古いオープンソースの形状計算ライブラリを用いて、地形にトンネルを通す。地形のメッシュが粗く、メッシュを構成する三角形の中をトンネルが完全に貫通する（地形の辺とトンネル形状の面が干渉しない）場合に、エラーを起こす。

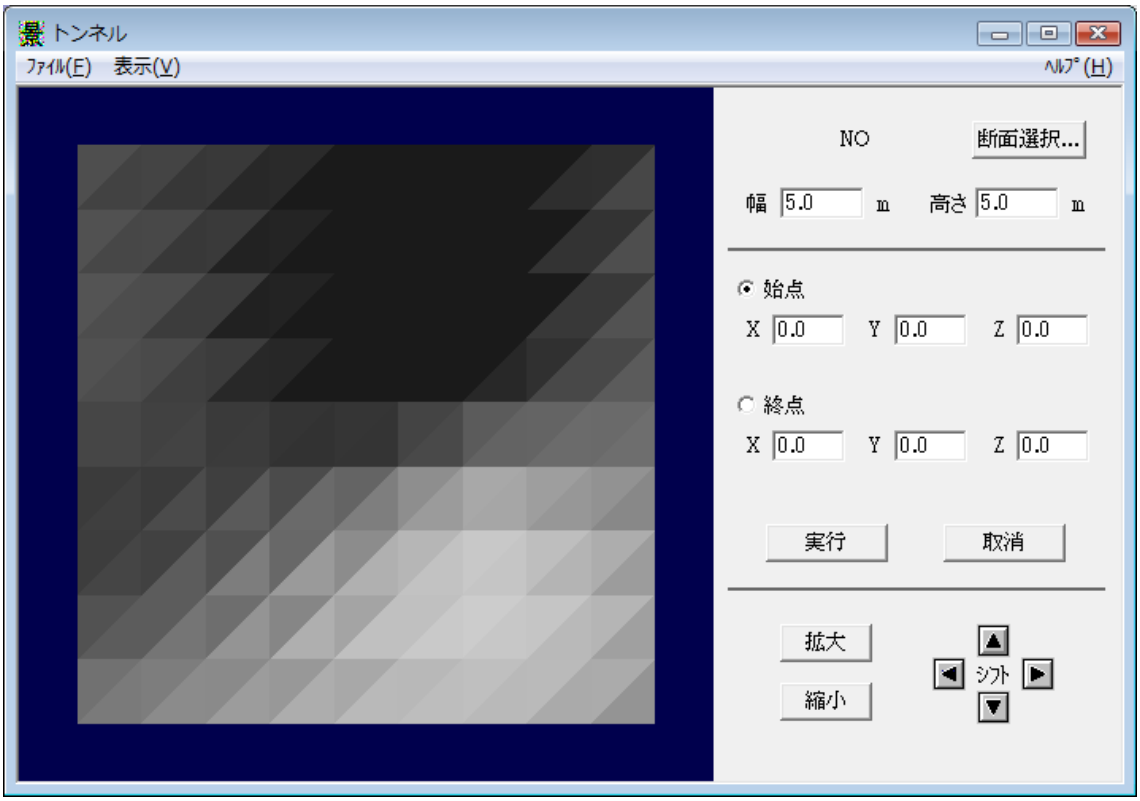


図６－２０：トンネル画面

①メニュー：IDR_MENU_LAND_HAICHI

メニュー構成
［ファイル］
＋［軌跡読込］
＋［軌跡保存］
＋［上書保存］
＋［終了］ダイアログを終了する
［表示］
＋［グリッド］
＋［メジャー］
＋［全体視界］
＋［上下範囲］
＋［縦横範囲］
＋［表示モード］

＋＋［テクスチャ表示］
＋＋［シェーディング表示］
＋＋［ワイヤーフレーム表示］
＋＋［オプション設定］
＋＋＋［地面のみ表示］
＋＋＋［地面テクスチャ表示］
＋＋［オプション解除］

② ハンドラ：CTunnelWnd (tunnelwnd.cpp)

③ OpenGL 画面：CMyOrthoVW m_orthoView (myorthovw.cpp)

④ 右側のダイアログ：

リソース:IDD_DLG_TUNNEL ハンドラ：CDialogBar m_Prm_Bar (tunnelwnd.cpp)

⑤ ヘルプ：tunnel.txt

６－６．プラグインＤＬＬと、三次元図形演算機能 (2011 年 3 月追記)

本章に掲載したプラグインＤＬＬの例は、三次元図形演算機能の発展過程を示している。

(1)道路法面生成機能(nori.dll)

1996 年に開発し、Ver.2.05 から標準機能に搭載した機能であり、地形の上に道路の線形を指定すると、地形と道路法面の図形演算を行う。

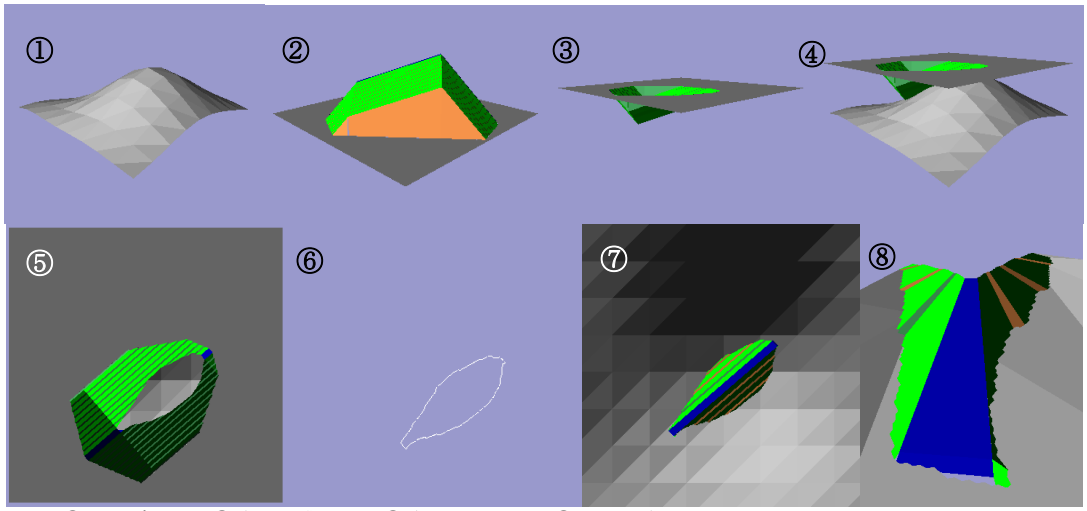
法面には、切土と盛土がある。計画された道路が地面よりも低い場合には、地形の一部を切り取り、その下に斜めの切土の法面を作成する必要がある。また、道路が地面よりも高い場合には、地形の上に斜めの盛土の法面を作成する必要がある。この場合、地形は残されていても、表示においては隠されるが、データ量や視点移動等の処理速度を考えると、不要となった部分を削除する方が無駄がない。更に、道路が斜面の等高線に沿って通るような場合には、山側に切土、谷側に盛土の法面を作成する必要がある。

道路法面生成(nori.dll)においては、この処理を、OpenGL のデプス・バッファの機能を用いて実現していた。処理過程を図６－２１に示す：１．道路面から仮定される、十分に大きなサイズの切土と盛土の法面をそれぞれ生成する②③。２．それぞれの法面と地形を重ねて④平面図表示を行う⑤。３．OpenGL のステンシル・バッファの機能を用い、地形が上となる平面領域と、法面が上となる平面領域を、ビットマップの形で取得する。４．地形と法面の境界の二次元形状を、折れ線として取得する⑥。５．この折れ線を用いて、地面の内、切土よりも上となる部分、及び盛土よりも下となる部分を削除する⑦。６．同じ折れ線を用いて、切土の内、地面よりも上となる部分、及び盛土の内、地面よりも下となる部分を削除する。７．最後に、地面と道路と切土面と盛土面の残された部分を合成する⑧。

この方法を用いた場合、たとえ非常に複雑な地形や道路であっても、計算処理のロジックは単純である。しかしながら、境界線の形状の計算精度は、デプス・バッファの解像度

（計算のために生成する画面の縦横ドット数）に依存する。しかし、デプス・バッファの解像度を高くしても、境界線の斜め直線区間は階段状となるため、境界線の折れ線の頂点数は非常に大きなものとなる。このため、二次元図形どうしの図形演算は時間のかかるものとなり、二次元の内外判定でエラーを発生する率が高くなる。

また、図形演算に平面図を用いていることから、垂直よりも大きな勾配（下向き）の面を有する図形に関する演算処理を行うことが原理的に不可能である。



①地形データ　②路面＋盛土面　③路面＋切土面　④地形＋路面＋切土面
（路面・法面は、断面形と中心線軌跡からパラメトリックに自動生成する）
⑤地形＋法面の相貫を平面表示画面で解析　⑥交差線（二次元）を求める
⑦これを用いて地形と法面を加工　⑧加工部分付近（デプス・バッファに起因する複雑な交差線）

図６－２１：道路法面の生成過程

この演算処理は、当初 u3 ライブラリの関数として実装された。Ver.2.09 への整理統合に伴い nori.dll として分離した際には、この三角形を対象とする図形演算処理機能を、このプラグインを構成する n3xxx.c のソースコードに移管した上でデバッグを加えた。

(2)トンネル生成機能(tunnel.dll)

1997 年以降に開発した、斜面に穴をあけ、トンネルの坑口を作成する機能である。トンネルの断面形状は、外部ファイルにより、自由に指定できる。幾何学的には、任意の断面を有する掃引体と、任意の形状を有する地形の間の演算処理の問題となる。

地面の内、トンネルの坑口付近に関して、トンネルの内部となる領域を切除する。一方、トンネルは、坑口部分で軸線に垂直な面で切断する。

この処理に際しては、デプス・バッファを使用せずに、図形どうしの相貫を直接座標計算する方法を用いた。地面の内、トンネルの内部となる部分を切除するためには、地面を構成する各面と、トンネルの側面を構成する各面との相貫を求め、不要となる部分を切除する、という処理を繰り返す。この処理においては、まず地形とトンネル側面の全てをまず三角形に分割した上で、三角形どうしの相貫関係を計算する処理とした。図形演算を三角形と三角形の相貫に還元することにより、場合分けを単純化することができる。さらに、トンネルの場合には、切り取る側の図形（トンネル）が、直線状の掃引体であることから、

地形の各面をこれと垂直な面に投影することにより、最終的には、二次元平面における三角形の切欠きの問題に還元することができる。

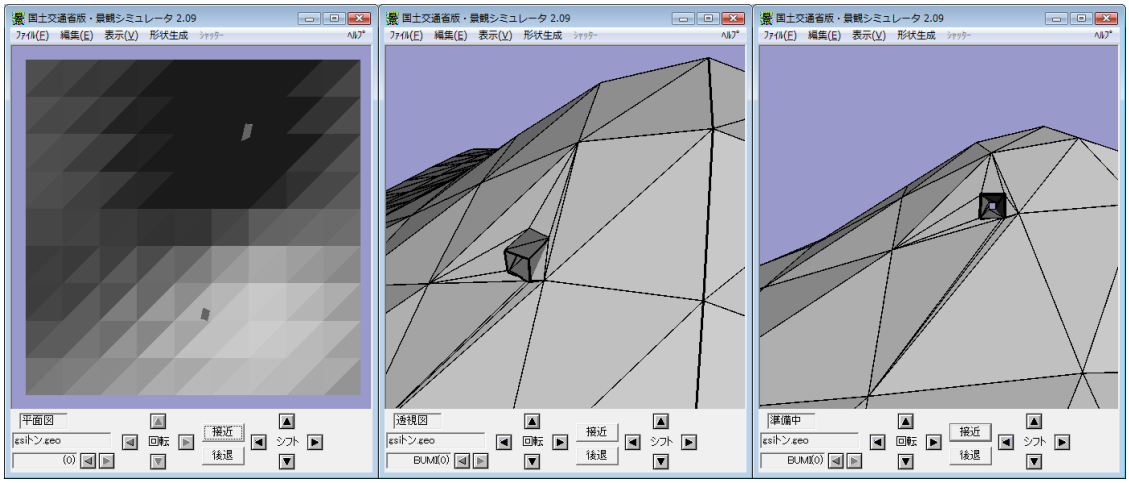


図６－２２：トンネルの生成過程

地面を構成する一つの三角形の切除処理においては、トンネルを構成する各三角形との相貫により、切れ目が生じた場合、切欠きの結果生じる図形を、次の処理に備えて更に三角形に分割する。また、一つの三角形が切断され、四角形と三角形に生じたような場合には、四角形を三角形に分割する処理を行う。このため、処理が終了した時点で元の一つの面が、多数の三角形に分割されたような状態となる（図６－２２）。

この処理を支援するために、別途地形処理機能(land.dll)を同時に用意した。この中には、地面を構成する面が４以上の辺を有していた場合に、全て三角形に分割する前処理機能や、後処理として、細分化された地形に関して、隣接する面であって向きの差が小さいものを再統合して、複雑さを減ずるポリゴン縮減機能を用意した（地形編集 land.dll から起動する「地形データの細分化と最適化」）。

この処理方法により、垂直よりも切り立った、下向きの面を持つ地形であっても、トンネルをあけることができるようになった。

三角形メッシュを用いた図形演算の機能は、tunnel.dll のビルドに含まれる u3cutmesh.c 及び u3fcom.c に移管しデバッグした。但し、地形を構成する面（三角形）に対してトンネルの断面が非常に小さく、三角形の辺に触れることなしに内部を貫通するような場合等に、演算に失敗する。

(3)園路生成機能

公園設計機能の開発を行った枝分かれバージョン(SimParkRoad.exe、非公開)においては、園路生成機能として、４以上の頂点を有する面から構成された地形に関しても、立体的に曲がりくねった園路とその法面と直接図形演算を行う必要が生じた。

この問題を解決するために、閉じた三次元図形どうしの演算を実行する関数を、基幹部分とは別にデバッグ・改良可能な DLL として改良することとし、本体側である開発中の実行形式 SimParkRoad.exe（非公開）とは分離して図形演算機能の改良を進めた。

Ver.2.09 への整理統合に際して、この開発用の実行形式から、園路生成のダイアログの部分を切り離し、プラグイン(ParkRoad.dll)に分離すると共に、改良過程にある図形演算機能も dll として外付けし、園路生成機能から必要に応じて起動できる構成とした。最終的に高い成功率が達成され実用的となったのは、2010 年末である。

この処理過程を、図６－２３に示す。１．ユーザーが指定した経路から８の字型の断面を持つ法面のテンプレートを作成する②。２．これを用いた図形演算で、地形①のうち不要となる部分を削除する③。３．経路から切土面⑤、盛土面④及び園路面を作成する。４．切土面・盛土面に関して地面の下（内部）となる領域を図形演算により切除する⑥。５．切除後残された地形、切土面、盛土面及び園路面を合成する⑦。

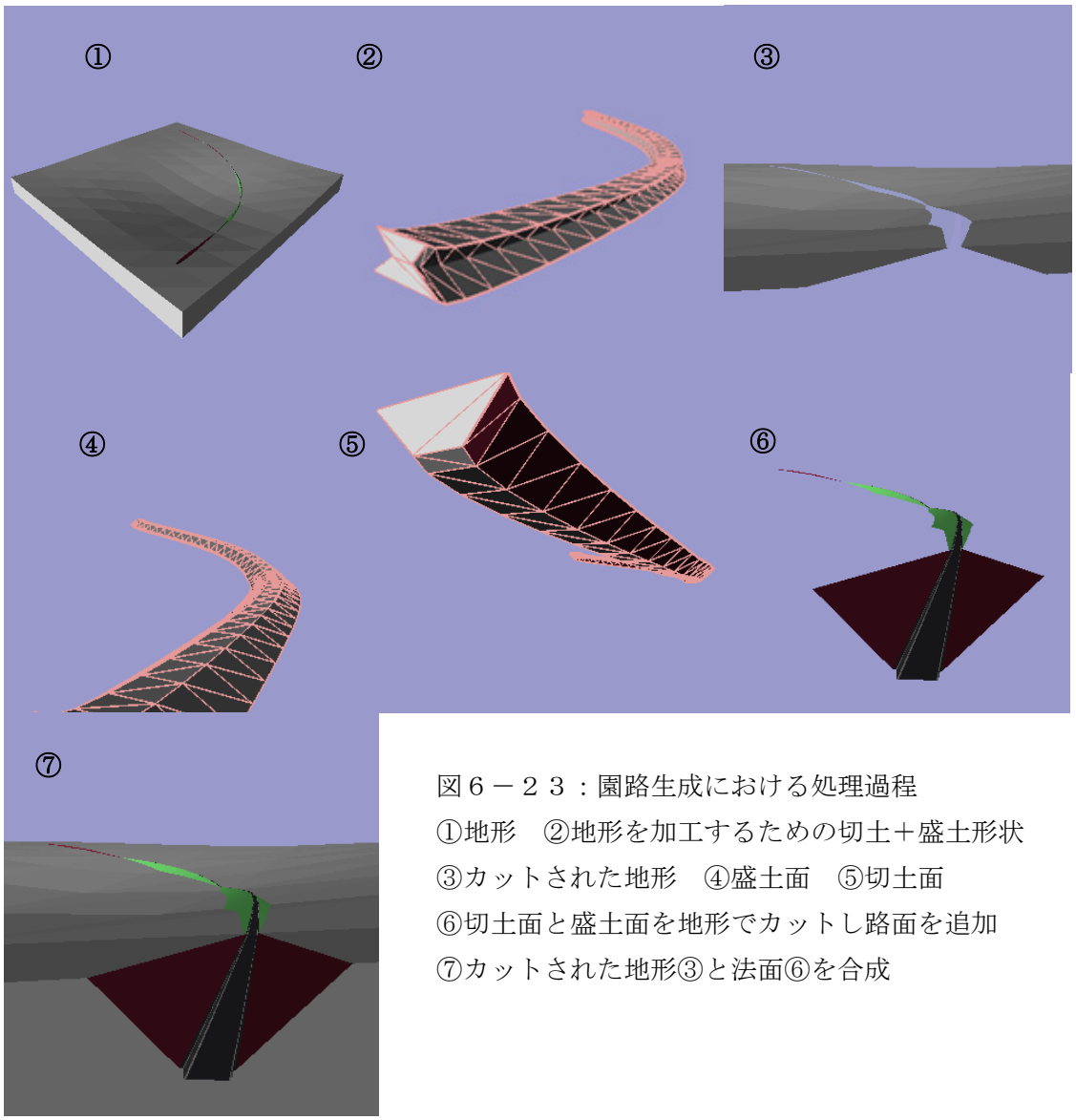


図６－２３：園路生成における処理過程
①地形　②地形を加工するための切土＋盛土形状
③カットされた地形　④盛土面　⑤切土面
⑥切土面と盛土面を地形でカットし路面を追加
⑦カットされた地形③と法面⑥を合成

なお、路面の断面は任意の形状をファイルで定義して指定することができるが、計算処理を単純化するために、法面の立体を作成する際には、園路断面は単純な線分として処理

している。最終的に合成する園路が複雑な断面を有する場合には、端面の一部である、園路面に対応する辺を、ユーザーが指定した形状の断面に差し替えている。この際に、捩じれた図形が生じる場合があるため、これを修正処理している。

図形演算機能は、今後開発するプラグイン DLL から利用することができる。その内部処理の詳細に関しては、別途資料の作成を予定している。

注釈：

1）統合化作業に着手した 2008 年 7 月時点で、以下の枝分かれバージョンが存在していた。

①Ver.2.07

2001 年に、まちづくり・コミュニケーション・システムのために改良した。インターネットを介してデータを取得する機能を付加したバージョンであり、Ver.2.09 に向けた統合作業開始時点で最も安定性が高い。Windows　XP に対応している。唯一の公開バージョンである。

②ステレオ版

2001 年に開発した。偏光メガネを用いて立体視するための画像を生成するためのバージョンである。2.07 と同時に開発したが、ソースコードが別となっている（同時公開）。

③Ver.4.0

1997 年頃から、ダム等の現場に対応するために、現場適用に伴い機能追加を行った、Ver.2.5、Ver.3.2 の延長上のバージョンである。影の表示機能、地形編集機能、トンネル生成、地形編集、動画合成機能、等が追加されているが、安定性に欠ける（Ver.3.2 実行形式を 1998 年 3 月に建設省土木研究所の WEB サイトから公開したが、Ver.4.0 は非公開）。

④ParkRoad

バグの少ない Ver.2.07 を出発点として 2004 年頃から、国営公園における景観検討に向けて機能追加を行った枝分かれバージョンである。GIS データから変換して作成した地形の上に、園路を作りこむ機能が追加されている（非公開）。

⑤SpeedUpSIM

2004 年頃、OpenGL のディスプレイ・リストの機能等を用いて、視点移動の高速化を試行したバージョンである（非公開）。

⑥MultiLang

2006 年頃、インドネシアでの住宅地の将来像を検討するために作成したもので、同じバージョンのままで、表示言語を切り換える機能を有している。言語に依存する部分は、ヘルプ、エラー・メッセージ、及びメニューやボタン等の表示である。このバージョンにおいては、それら全てがテキスト・ファイルとして外部化されているため、翻訳作業によりメモ帳で開くことのできるテキスト・ファイルを用意するだけで、新たな言語上で使用することが可能となった（非公開）。

この他に、1996 年頃に翻訳移植した、ハングル版、インドネシア語版などが存在するが、機能的には他のバージョンに含まれるもののみであり、言語切り替えに関しては、⑥の多言語機能を組み込んだ上で、プラグイン DLL や外部関数にも拡張した Ver.2.09 で対応可能である。