第３章　マルチウインドウ管理

ポイント：プレゼンターマルチウインドウの管理法を学ぶ。

１．マルチウインドウ管理法
２．プログラムの起動時と終了時の処理

3.1 はじめに

本章では、プレゼンターの基本構成とマルチウインドウ管理技法について学ぶ。SPACE システムは多くのサブシステムと多数のファイル群とで構成され、それらが有機的に結合することよって効果的な働きをする。ここでは、SPACE におけるプレゼンターの役割と、プレゼンターの起動時と終了時の処理内容について説明する。

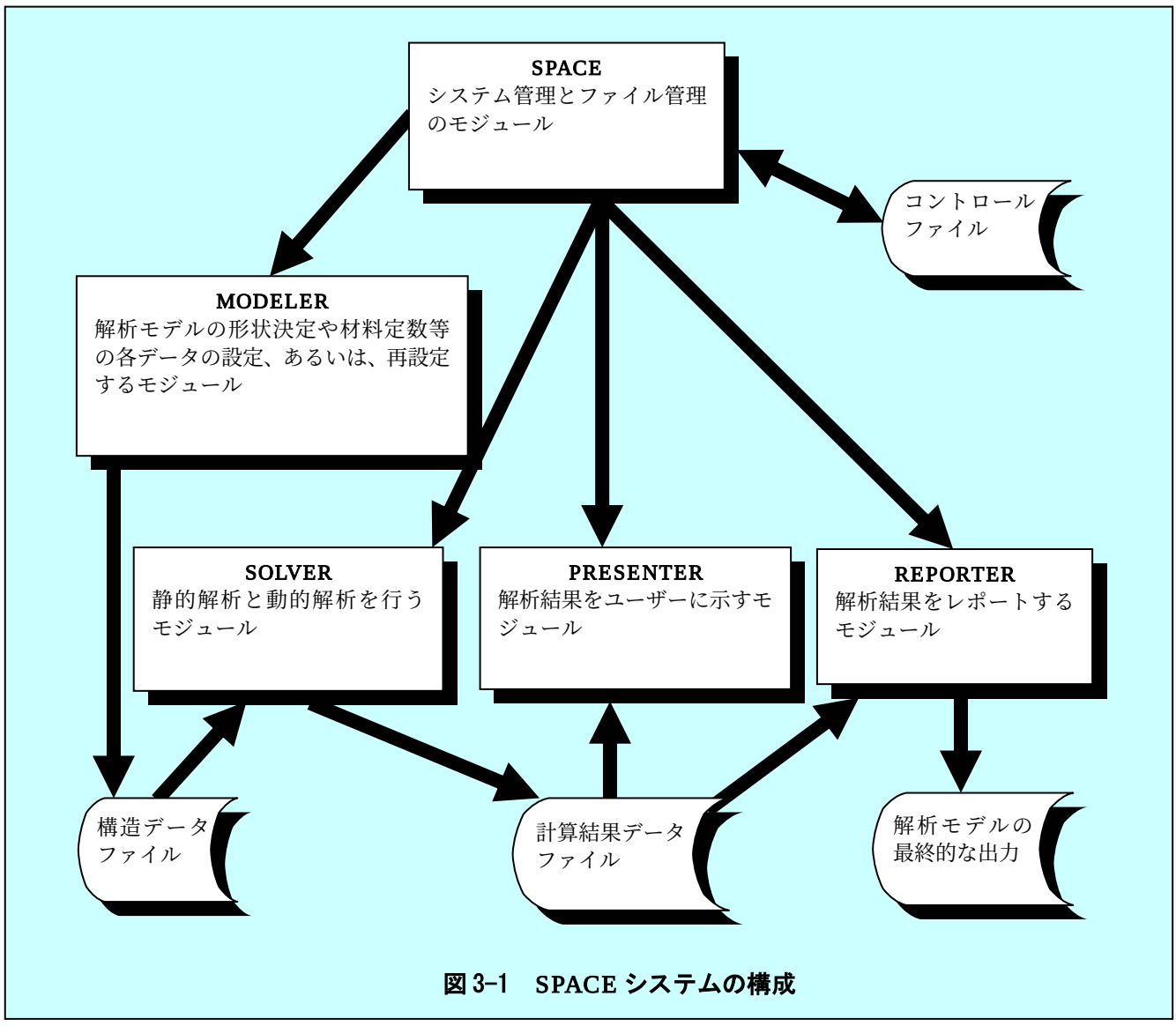

図 3-1　SPACE システムの構成

3.2 プレゼンターの起動と終了処理

SPACE システムは図 3-1 に示すように多くのサブシステムによって構成されている。同図を参照しながら、最初に、プレゼンターが起動されるまでの働きを見てみよう。

SPACE を使用する場合、まず、SPACE を利用してコントロールファイルを作成し、他のファイル群の名前を設定する。次にモデラーを用いて、構造データや他のデータ群を作成し、該当する各々のファイルに出力する。最後に、動的解析に必要となる制御用パラメータを、SPACE を用いて設定し、ファイルに出力する。これで準備が終わり、後は動的解析システムを起動して解析を実行し、その結果をファイルに出力する。動的解析を終了した後、プレゼンターを起動して、その結果を分析することになる。プレゼンターには、グラフや表、あるいはアニメーションなどの機能があり、これらを駆使して静的・動的解析結果を分析する。

各ファイルの詳細は、SPACE のユーザーマニュアルあるいはリファレンスマニュアルを参照されたい。

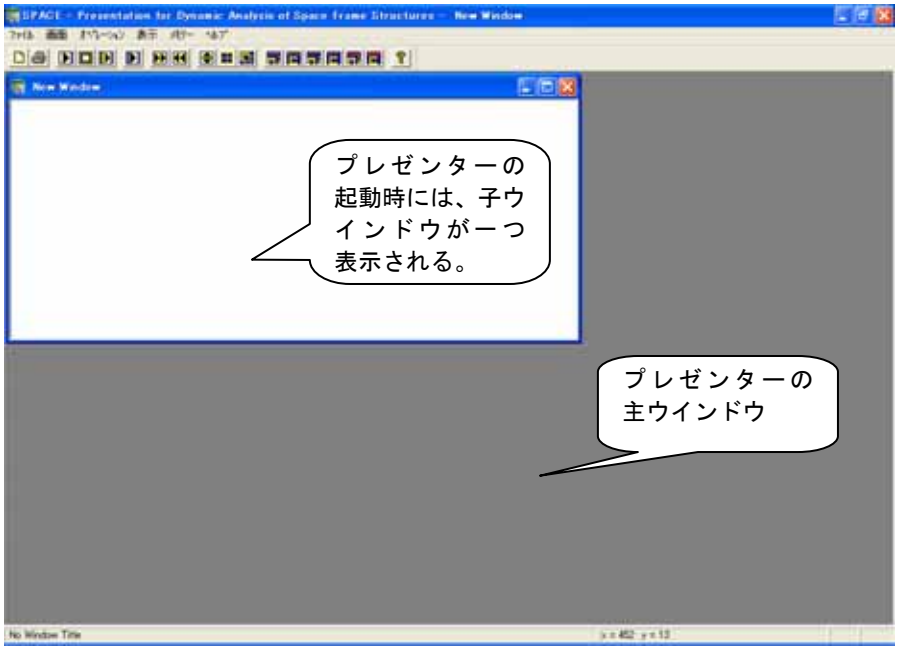

図 3-2 プレゼンター起動画面

プレゼンターの起動は SPACE によって行われ、情報の伝達は TEMP 領域に書き出される spacesys.xxx ファイルで行われる。このファイルの仕様は以下のようであり、SPACE からプレゼンターに解析種別とコントロールファイル名を受け渡す。ファイルの内容は 2 行で構成されており、一行目は解析種別を表し、2 行目はコントロールファイルの絶対パス名を表す。

```
10
D:¥space-frame¥DISK2¥動的解析編¥アーチの動的安定¥両端ピン支持アーチ¥Arch_弾性分岐動座屈.ctl
```

プレゼンターに起動がかかると、まず、一般的な手続きを行った後、Mainfrm.cpp 内の Mainframe コンストラクタでシステム内の初期設定を行う。ここでの処理の重要な部分を以下に示し、その処理内容の説明を行う。プログラム言語は VC++で、オブジェクト指向型である。このプログラムは、Microsoft 社の Developer Studio を用いて、基本部分を作成したものである。

プレゼンターが起動して、主ウインドウが描画される前に、主ウインドウを管理する mainframe クラスからひとつのオブジェクトが構築される。クラスが構築されるときと消滅するとき、必ず次に示す関数が呼ばれる。これをそのクラスのコンストラクタ及びデストラクタと呼ぶ。このクラスのコンストラクタ CMainFrame::CMainFrame()では、動的領域の確保などの初期設定が行われる。その内容については、後にプログラムを参照しながら説明する。主ウインドウが消滅するとき、つまり、解析が終了するとき、デストラクタ CMainFrame::~CMainFrame()が呼ばれる。この関数では、主にコンストラクタで動的領域確保した配列領域を解放している。

少し、MainFrame クラスの内容を見てみよう。BEGIN_MESSAGE_MAP と END_MESSAGE_MAP のキィワードに挟まれて記述されている関数に特別な意味がある。例えば、この２つのキィワードの間にある ON_WM_CREATE()と ON_WM_CLOSE()の存在は、他からのメッセージによって該当する関数が呼び出されることを意味する。つまり、ウインドウが構築されるとき、関数 CMainFrame::OnCreate()が起動され、同じくウインドウが閉じられるとき、関数 CMainFrame::OnClose()が実行される。この２つのキィワードの中に含まれる関数は、他のクラスやユーザーのマウス操作からのメッセージによって駆動されるわけである。このような記述は他のクラスにも見られるので、出現するときは注意して見られたい。

それでは、mainframe クラスの記述を検討しよう。

```
//    ――――――――――――――――――――――――――――――――――――――――
//    ●    CMainFrame クラスの動作の定義
//    ――――――――――――――――――――――――――――――――――――――――
//
#include "stdafx.h"
#include "sf31.h"
#include "sf31datex.h"
#include "MainFrm.h"
#include "fort.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
```

```
#endif
//    ――――――――――――――――――――――――――――――――――――――――――――
//    ●    CMainFrame クラス
//    ――――――――――――――――――――――――――――――――――――――――――――
//
IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)
//――――――――――――――――――――――――――――――――――――――――――――― ● メッセージマップ開始
BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)
        ON_WM_CREATE()
        ON_COMMAND(ID_WND_CHECK, OnWndCheck)
        ON_COMMAND(ID_VIEW_STATUS_BAR, OnViewStatusBar)
        ON_UPDATE_COMMAND_UI(ID_VIEW_STATUS_BAR, OnUpdateViewStatusBar)
        ON_UPDATE_COMMAND_UI(ID_INDICATOR_LEFT, OnUpdateLeft)
        ON_UPDATE_COMMAND_UI(ID_INDICATOR_RIGHT, OnUpdateRight)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
//――――――――――――――――――――――――――――――――――――――――――――― ● メッセージマップ終了
static UINT indicators[] =
{
        ID_SEPARATOR,           // ステータス ライン インジケータ
        ID_SEPARATOR,
        ID_INDICATOR_LEFT,
        ID_INDICATOR_RIGHT,
};
//    ――――――――――――――――――――――――――――――――――――――――――――
//    ●    CMainFrame クラスのコンストラクター
//    ――――――――――――――――――――――――――――――――――――――――――――
//
CMainFrame::CMainFrame()
{
//――――――――――――――――――――――――――――――――――――――――――――― ●  プレゼンターの初期設定
        int ihan,ndim,nnode;
        fno_yy=0;
        F_time_ii =0;
        F_Speed = 4;
        FI_Speed = 1;
        ihan=0;
        ndim=3;
        nnode=1;
        int xxfig[3];
        F_radio_color_pr =0;
//――――――――――――――――――――――――――――――――――――――――――――― ●  ペン用のデータをセット
        create_penx();
//――――――――――――――――――――――――――――――――――――――――――――― ●  システムデータのセット
        SYSNAM(&F_idfile[0],F_title);
//――――――――――――――――――――――――――――――― ● 透視図用データのセット
        PERSCT(&xxfig[0],&xxfig[1],&xxfig[2],&F_scrnps[0],
               &F_viewps[0],&F_scalps,&F_scalep[0],&F_mag[0]);
        FF_scalps = F_scalps;
//――――――――――――――――――――――――――――――― ● 動的用コントロールデータその１のセット
        DYCTL1(&ihan,&F_nindis,&F_gindis,&F_f1sec);
//――――――――――――――――――――――――――――――― ● 動的用コントロールデータその２のセット
        DYCTL2(&ihan,&F_nstep,&F_delt,&F_igra[0],&F_ibata,
```

```
                        &F_beta,&F_gamma,&F_xgal[0],&F_ntime,&F_dlamst);
//――――――――――――――――――――――――――――――――― ● 出力用データのセット
        DOUTCL(&ihan,&F_iwstp,&F_soutsc,&F_dmaxck);
//――――――――――――――――――――――――――――――――――――――――――― ● 配列データ設定用の値を計算
        F_mstep = F_f1sec/(F_delt*(float)F_iwstp);
        F_nstep=F_nstep/(float)F_iwstp+1;
        F_all_step = F_mstep+F_nstep-1;
        F_delt_cl = F_delt*(float)F_iwstp;
        F_all_time = (F_all_step-1)*F_delt_cl;
//――――――――――――――――――――――――――――――――――――――――――― ● 構造データの仮読み
        int m_axis;
        INPTFX(&node,&nelem,&mnbsb,&nrzero,&locod,&m_axis);
        int nomem;
        float SY;
        INPIST(&nelem,&nomem,&SY);
//――――――――――――――――――――――――――――――――――――――――――― ● 動的配列の確保
        ihan=1;
        nnode=node;
        posit = new float[nnode*ndim];
        if(posit == 0){
        MessageBox("Sorry! Memory allocation failure. You should stop this application.");
        }
        iconsb = new int[2*mnbsb];
        if(iconsb == 0){
        MessageBox("Sorry! Memory allocation failure. You should stop this application.");
        }
         ・
         ・
        F_iso =   new int[mnbsb];
        if(F_iso == 0){
        MessageBox("Sorry! Memory allocation failure. You should stop this application.");
        }
//――――――――――――――――――――――――――――――――――――――――――― ● 動的配列定義終了
//――――――――――――――――――――――――――――――――――――――――――― ● 構造データの入力
        GET_S_COMP_MODEL_X(i_comp_model);
        int nelemx = nelem - nomem;
        INPTFY(&node,&nelemx,&mnbsb,&nrzero,&locod,posit,&ndim,
                   iconsb,imeme,imeme_line,&nm_line,mtype,F_snp,F_smyp,F_smzp,
                   F_bound,&xxfig[0],Rrot,&F_nindis,&F_gindis,F_free,F_mytype,
                   F_al,F_alqq ,&m_axis,Fnm_type,F_iso,i_comp_model);
        if(nomem != 0) {
        INPISF(&nomem,&nelem,&mnbsb,imeme,mtype,F_snp,F_smyp,F_smzp,&SY,F_mytype,F_alqq);
        }
        F_mtype[0]=2;
        F_mtype[1]=4;
        F_mtype[2]=3;
        F_mtype[3]=5;
        F_read_demode =0;
        F_memory_mem_number =0;
        F_memory_nod_number =0;
}
//    ――――――――――――――――――――――――――――――――――――――――――
//    ●    CMainFrame のデストラクター
//    ――――――――――――――――――――――――――――――――――――――――――
```

```
//
CMainFrame::~CMainFrame()
{
//――――――――――――――――――――――――――――――――――――――――――――――― ● 動的配列の解放
         delete [] posit;    //ok
         delete [] iconsb;    //ok
         delete [] imeme;    //ok
         delete [] imeme_line;    //ok
         delete [] F_bz;    //ok
         .
         .
        if(FF_read_mass == 1){
         delete [] FF_mass;       //ok
        }
}
//     ―――――――――――――――――――――――――――――――――――――――――――――――
//     ●    OnCreate
//     ―――――――――――――――――――――――――――――――――――――――――――――――
//
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
//――――――――――――――――――――――――――――――――――――――――――――――― ●
         if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)
             return -1;
//――――――――――――――――――――――――――――――――――――――――――――――― ● 主ウインドウ用ツールバー
         if (!m_wndToolBar.Create(this) ||
                  !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
         {
                  TRACE0("Failed to create toolbar¥n");
                  return -1;      // 作成に失敗
         }
//――――――――――――――――――――――――――――――――――――――――――――――― ●  ステイタスバーの作成
         if (!m_wndStatusBar.Create(this,
                  WS_CHILD | WS_VISIBLE | CBRS_BOTTOM, ID_MY_STATUS_BAR) ||
                  !m_wndStatusBar.SetIndicators(indicators,
                    sizeof(indicators)/sizeof(UINT)))
         {
                  TRACE0("Failed to create status bar¥n");
                  return -1;      // 作成に失敗
         }
          m_wndStatusBar.SetPaneInfo(0,0,SBPS_STRETCH,0);
//――――――――――――――――――――――――――――――――――――――――――――――― ● ツール チップス
         m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
                  CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);
//――――――――――――――――――――――――――――――――――――――――――――――― ● ツール バーをドッキング可能
         m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
         EnableDocking(CBRS_ALIGN_ANY);
         DockControlBar(&m_wndToolBar);
         return 0;
}
//     ―――――――――――――――――――――――――――――――――――――――――――――――
//     ●    描画とプリンター用のペンとフォントの設定
//     ―――――――――――――――――――――――――――――――――――――――――――――――
//
```

```
void CMainFrame::create_penx()
{
   int i1;
   Brash_color = RGB(0,80,130);
   F_edit_line_width_m =3;
   F_edit_line_width_g =3;
   for(int i=0;i<50;i++){
    i1=5*(50-i);
    NewPenx_a[i].CreatePen(PS_SOLID,1,RGB(255,255-i1,255-i1));
    NewPenx_a[100-i].CreatePen(PS_SOLID,1,RGB(255-i1,255-i1,255));
    PrPenx_a[i].CreatePen(PS_SOLID,F_edit_line_width_m,RGB(255,255-i1,255-i1));
    PrPenx_a[100-i].CreatePen(PS_SOLID,F_edit_line_width_m,RGB(255-i1,255-i1,255));
   }
    NewPenx_a[50].CreatePen(PS_SOLID,1,RGB(255,255,255));
    PrPenx_a[50].CreatePen(PS_SOLID,F_edit_line_width_m,RGB(255,255,255));
    NewPenx_a[101].CreatePen(PS_SOLID,1,RGB(255,255,255));
    NewPenx_a[102].CreatePen(PS_SOLID,2,RGB(0,100,255));
    NewPenx_a[103].CreatePen(PS_SOLID,2,RGB(255,255,0));
     ・
     ・
    PrPenx_a[112].CreatePen(PS_SOLID,F_edit_line_width_g,RGB(0,0,255));
    PrPenx_a[113].CreatePen(PS_SOLID,F_edit_line_width_g,RGB(0,220,0));
//―――――――――――――――――――――――――――――――――― ● フォントのセット
      float xx= 4.;
      F_ipx=15*xx;
           F_ipy=15*xx;
           int point =12*xx;
         CClientDC dc(AfxGetMainWnd());
         prFont.CreateFont(point
                  ,0,0,0,FW_NORMAL,FALSE,FALSE,0,
                  SHIFTJIS_CHARSET,
                  OUT_DEFAULT_PRECIS,
                  CLIP_DEFAULT_PRECIS,
                  DEFAULT_QUALITY,
                  DEFAULT_PITCH | FF_MODERN,
                  "ＭＳ　Ｐゴシック");
            point =10*xx;
         prFontx.CreateFont(point,0,0,0,FW_NORMAL,FALSE,FALSE,0,
                  SHIFTJIS_CHARSET,
                  OUT_DEFAULT_PRECIS,
                  CLIP_DEFAULT_PRECIS,
                  DEFAULT_QUALITY,
                  DEFAULT_PITCH | FF_MODERN,
                  "ＭＳ　Ｐゴシック");
}
```

ファイル mainframe.cpp の最初で、多くのヘッダーファイルをインクルードしている。特に、fort.h には FORTRAN プログラムとのインターフェイス文が保持されており、また sf31datex.h では、プレゼンターの制御を行うための重要なグローバル変数やグローバル配列を定義している。ここでのグローバル変数やグローバル配列の定義は extern 宣言を

行っており、実際の宣言は sf31dat.h に記述されている。これは CSf31View.cpp ファイルにインクルードされており、ここでグローバル変数などが割り付けられることになる。また、これらのグローバル変数やグローバル配列は図形表示やアニメーションの制御などに用いられる。各変数や配列の意味は定義文の後にコメントされているので参照されたい。また、これらの変数等は計算に関係しないため、実数は全て単精度で定義している。以下にその内容を示す。

```
//    ―――――――――――――――――――――――――――――――――――――――
//      ●　グローバル変数の定義
//    ―――――――――――――――――――――――――――――――――――――――
   extern  int    View_C_panel;    // 制御パネル表示コード　０：非表示　１：表示
   extern  HWND    F_hwnd;         // ウインドウ用ハンドル
   extern  double T_analysis;      // 解析全体の時間（sec）
   extern  double dt_analysis;     // 解析増分時間（sec）
   extern  int    thread_on;       // 計算実行中かどうかを示すパラメータ ０：実行中でない １：実行中
   extern  int    icontrol;        // 計算コード      0:予備計算  1:動的解析 99:終了処理
   extern  int    iend_code;       // 計算終了コード 0=継続  1:終了
   extern  int    ierr_dat;        // エラーコード
   extern  int    n_step,ns_step;  // 今回の解析ステップ数及び解析開始ステップ
   extern  int    id_max_v;        // 時刻ｔにおける最大変位を示す節点番号
   extern  double d_max_v;         // 時刻ｔにおける最大変位
   extern  int    F_calnum;        // 解析種別
   extern  int    F_holt;          // 計算実行が停止中かどうかを示すパラメータ　０：停止中でない
   extern  char   F_title[60];     // コントロールファイルに書かれたタイトル
   extern  int    F_idfile[100];   // 書き込み可能チェック用  1:可能  0:不可
   extern  char   Title[100];      // 解析タイトル
   extern  float* posit;           // スペースフレーム描画用節点座標
   extern  int*   iconsb;          // 部材両端の節点番号
   extern  int*   imeme;           // 部材に付されたが要素番号
   extern  int*   imeme_line;      // 部材グループ番号
   extern  int    nm_line;         // 部材グループ番号の最大値
   extern  int*   mtype;           // 部材内で弾塑性状態を表示する個数
   extern  float* F_bz;            // 透視図形描画用座標
   extern  CPen   NewPenx_a[131]; // 画面描画用ペンの種類
   extern  CPen   PrPenx_a[131];  // プリント用ペンの種類
   extern  CPen   NewPen_pr[10];  // プリント用ペンの種類その２
   extern  int*   F_mytype;        // 部材モデル番号
   extern  int*   Fnm_type;        // 履歴モデルのタイプ
   extern  int*   F_free;          // 各節点の自由度
   extern  float* F_snp;           // 降伏軸力
   extern  float* F_smyp;          // ｙ軸塑性モーメント
   extern  float* F_smzp;          // ｚ軸塑性モーメント１
   extern  int*   F_bound;         // 境界出力用データ
   extern  float* F_load_d1;       // ファイル１の静的荷重
   extern  float* F_load_d2;       // ファイル２の静的荷重
   extern  float* F_load_d3;       // ファイル３の静的荷重
   extern  float* F_al;            // 応力位置決定出力用データ
   extern  int*   F_alqq;          // 応力設定用オプション（１：モーメント、２：せん断力）
//                              ●　ウインドウ管理
   extern  int    F_time_ii;       // 解析過程のステップ数
```

```
    extern  int    F_read_mass;     // 節点質量動的確保オプション
    extern  int*   F_mass;          // 節点質量表示用データ
    extern  int    mem1;            // 画面出力ワーク領域
    extern  int    mem2;            // 画面出力ワーク領域
    extern  int    fno_xx;          // 画面コードワーク領域
    extern  int    fno_yy;          // 画面コードワーク領域
    extern  int    F_wind[100][5];  // ウインドウ管理配列
    extern  HWND   F_hwnd_timer;    // 解析を開始したウインドウのハンドル
    extern  HWND   F_hwind[100];    // ウインドウハンドル管理配列
//                              ●    透視図描画用データ
    extern  float F_scrnps[3],F_viewps[3],F_scalps,F_scalep[5],F_mag[6],FF_scalps;
//  透視画面の原点位置、視点位置、透視図スケール、
//  加速度などのスケール、矢印などの大きさ設定、透視図スケールの保存用
    extern  int  mnbsb,nrzero,locod,node,nelem;
// 部材数、境界節点数、局所座標数、節点数、要素数
    extern  int    F_pos[2][2];     // 画面の左上と右下の座標
    extern  int    F_pos_pr[2][2];  // プリンタ用紙の左上と右下の座標
    extern  CPoint F_point;         // マウス位置の座標を入れる入れ物
    extern  float  a_Zoom;          // マウス操作の拡大・縮小の大きさ
    extern  float  F_amx,F_amy;     // 図形移動量
    extern  int    F_hantei;        // マウス操作判定領域
    extern  float  F_Time;          // 解析過程における時間
    extern  int    F_ontimer;       // 解析状態を表すパラメータ
//                              ●    動的コントロールその１
    extern  int    F_nindis;        // 初期不整使用有無パラメータ
    extern  float  F_gindis,F_f1sec;// 初期不整大きさ、第一段階解析時間
//                              ●    動的コントロールその２
    extern  int    F_nstep,F_igra[3],F_ibata,F_ntime;
//解析ステップ数、地震波解析使用パラメータ、βの値を決めるパラメータ、反復最大回数
    extern  float  F_delt,F_beta,F_gamma,F_xgal[3],F_dlamst,F_delt_cl,F_all_time;
//解析用増分時間、ニューマークβ、ニューマークγ、地震最大加速度
//結果を出力する時間、解析時間
//                              ●    解析結果の出力コントロール
    extern  int    F_iwstp,F_mstep,F_all_step;
// 解析結果をファイルに出力する間隔、描画する間隔、解析全ステップ
    extern  float  F_soutsc,F_dmaxck;
// ＊＊結果を出力する最初の時刻、崩壊と見做す最大変位（現在使用されていない）
    extern  float  F_delugg;          // 地震波形増分時間
//                              ●    荷重、反力と不釣合い力
    extern  int    F_read_disp;       // ３次元座標の設定オプション
    extern  float* F_disp;            // 解析モデルの３次元座標
    extern  int    F_read_unbalanceF; // 不釣合力ベクトルの設定オプション
    extern  float* F_unbalanceF;      // 不釣合力ベクトル
    extern  float  F_unbalanceF_max;  // 不釣合力ベクトルの最大値
    extern  int    F_read_ndbalanceF; // 釣合力ベクトルの設定オプション
    extern  float* F_ndbalanceF;      // 釣合力ベクトル
//                              ●    節点速度と加速度
    extern  int    F_read_vel;        // 速度ベクトルの設定オプション
    extern  float* F_vel;             // 速度ベクトル
    extern  int    F_read_acc;        // 加速度ベクトルの設定オプション
    extern  float* F_acc;             // 加速度ベクトル
    extern  int    F_read_accab;      // 絶対加速度ベクトルの設定オプション
    extern  float* F_accab;           // 絶対加速度ベクトル
    extern  float* Rrot ;             // 座標変換行列
    extern  int    mm_opt[10];        // 部材描画オプションの入れ物
//                              ●    部材応力
```

```
   extern  int    F_read_spring;     // 部材応力ベクトルの設定オプション
   extern  float* F_fay;             // 降伏関数値ベクトル
   extern  float* F_n_spring;        // 軸力ベクトル
   extern  float* F_my_spring;       // ｙ軸曲げモーメントベクトル
   extern  float* F_mz_spring;       // ｚ軸曲げモーメントベクトル
   extern  int*   F_stat_spring;     // 部材の弾塑性状態
   extern  int    F_mtype[5];        // 塑性ヒンジ出力位置の個数を設定する入れ物
//                              ●   グローバル変数の定義
   extern  int    n_iterate;          // 反復回数
   extern  int    nm_iterate;         // 陰解法回数
   extern  int    numb_method;        // 解析法番号
```

この中で注目すべき項目は、動的領域の宣言である。例えば、先に示したsf31datex.hの

```
extern int F_read_disp;
extern float* F_disp;
```

では、変数 F_disp に記号＊が付いている。これは、この変数が動的領域であることを意味する。そのためこの領域を使用するためには、次のようにプログラムの中で動的領域確保を行うコードが必要となる。

```
//   ―――――――――――――――――――――――――――――
//     ●  動的領域の確保
//   ―――――――――――――――――――――――――――――
  if(F_read_disp == 0){
     F_disp = new float[3*node];
      if(F_disp == 0){
        MessageBox("Sorry! Memory allocation failure.");
      return;
  }}
```

ただし、ユーザーがそのコードを実際に実行し、その変数の動的領域確保がなされたかどうかは、プログラムを記述した段階では不明である。そのため、ここでは動的領域確保を行った場合は、その上の変数F_read_dispを１に設定する。動的領域確保を行ったかどうかを知る必要が生じたとき、この変数を利用する。例えば、デストラクタでは、以下に示すように、このF_read_dispをチェックして動的領域を解放するか否かを決定している。

```
if(F_read_disp == 1){
    delete [] F_disp;
}
```

記号＊が付いた他のグローバル変数も同様な使い方をしており、これら

の変数の全てが実行時に大きさが決定する配列として用いられる。

次に、先に示したクラスコンストラクタ CMainFrame::CMainFrame()を見てみよう。ここでは３つの処理を行う。一つ目はグローバル変数の初期化であり、二つ目は図形用データや節点変位などの動的領域を確保すること、三つ目は動的解析用パラメータや動的構造データを入力することである。

コンストラクタの最初の部分は、グローバル変数の初期値セットであり、図形処理用やウインドウ管理用など各種のグローバル変数が初期化されている。次に、描画ペンの設定を行う関数 create_penx( )をコールする。この関数では、図形用カラーペンの太さや色を設定している。この関数 create_penx( )については後で説明する。

次に、動的解析を制御したパラメータをファイルより入力する。これらは、動的解析した状況をプレゼンターでも知る必要があるため、これらのコントロール用ファイルから解析用パラメータを入力する。最初に、関数 SYSNAM( )によって、SPACE からの伝達情報とコントロールファイルを読み込み、プレゼンターで必要となる各ファイル名をセットする。SPACE システムでは大文字で記述されている関数は SYSNAM( )のように FORTRAN のサブルーチンを表し、他のサブルーチンも全てこの仕様で書かれているので記憶されたい。

次は、透視図用ファイル、動的解析用制御ファイルその１、動的解析用制御ファイルその２、動的解析出力制御ファイルを読み込む。それらの関数は、

```
１．PERSCT( )　２．DYCTL1( )　３．DYCTL2( )　４．DOUTCL( )
```

であり、これらの関数は全て FORTRAN で記述されている。詳細は、他のマニュアルを参照されたい。これらの関数から得られる情報から、描画やアニメーションに必要なデータを取り出し、グローバル変数にセットする。

このコンストラクタの第 2 段階は、構造データを予備入力し、描画に必要な配列を動的確保することである。構造用ファイルの予備入力は、関数 INPTFX()で行う。この関数で、構造用ファイルの制御データを読み込み、その後、構造用制御パラメータを用いて描画用配列を動的に確保する。例えば、配列 posit は節点の座標を表し、以下のように new コマンドで動的領域を確保する。次の if 文で posit に値がゼロの場合は、動的確保に失敗したことを意味し、エラーメッセージを表示する。

```
posit = new float[nnode*ndim];
if(posit == 0){
  MessageBox("Sorry! Memory allocation failure. You should stop this application.");
            }
```

最後の第３段階では、実際に構造データファイルを読み込む。ただし、描画に不必要なデータは読み飛ばすことになる。この構造データファイルを読むプログラムは、関数 INPTFY()であり、FORTRAN で記述されている。構造データに関する２つの入力プログラムの詳細は、他のマニュアルを参照されたい。

続いて、デストラクタについて説明する。デストラクタ~CMainFrame()における主な仕事は、グローバル動的領域を解放することである。特に、変数 F_read_spring 入力パラメータが１となっている場合、ユーザーが応力などの表示を行うよう指示したために動的領域を確保したことを意味し、デストラクタでは、これらの動的領域を解放する。同様に、F_read_ndbalanceF、F_read_mass は不釣合力と質量に関連する入力パラメータである。上記の動的領域を解放するコードを以下に示す。

```
if(F_read_spring == 1){
        delete [] F_fay;
        delete [] F_n_spring;
        delete [] F_my_spring;
        delete [] F_mz_spring;
        delete [] F_stat_spring;
 }
```

関数 create_penx()は、クラス CmainFrame のメンバー関数であり、コンストラクタの中で呼ばれている。ここでは、ペンの太さと色を設定し、また、フォントの設定も行っている。これらのフォントやペンのデータは、グローバル変数にセットされており、他のクラスの図形描画の中で使用されることになる。

クラス CmainFrame の中の他のメンバー関数として、OnCreate()関数がある。この関数は、主ウインドウが表示された後、このウインドウのツールバーやステイタスバーを作成する。これらの詳細は、VC++のマニュアルあるいは参考書を参照されたい。

3.3 子ウインドウ開始と終了処理

本節では、プレゼンターシステム内の子ウインドウが表示される際、及びウインドウが閉じられる際にどのような処理が行われるかについて解説する。プレゼンターに起動がかかると、一般的な手続きを行われた後、Mainfrm.cpp 内の Mainframe コンストラクタで初期設定が行われ

る。その後、ひとつの子ウインドウに関するオブジェクトを構築し、子ウインドウを描画する。

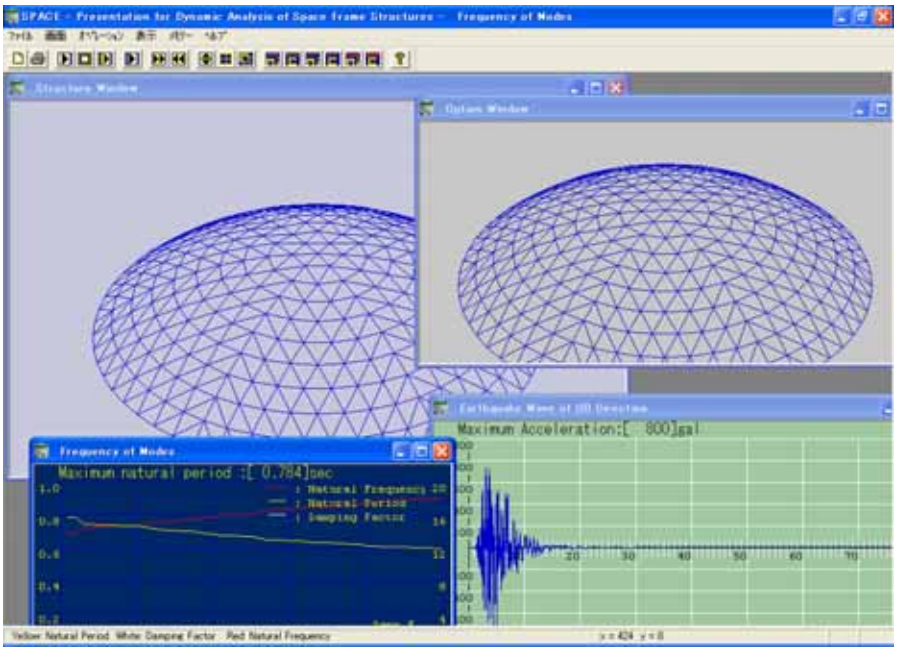

図 3-3 マルチウインドウ表示

まず、この CSf31View クラスのコンストラクタとデストラクタ及びメッセージマップについて説明しよう。該当する部分の VC++コードの一部を以下に示す。

```
//     ――――――――――――――――――――――――――――――――――――――――
//     ●    sf31View.cpp : CSf31View クラスの動作の定義
//     ――――――――――――――――――――――――――――――――――――――――
//     ●     マルチウインドウシステム定義
//     ――――――――――――――――――――――――――――――――――――――――
#include "malloc.h"
#include "stdafx.h"
#include "sf31.h"
#include "sf31dat.h"
#include "sf31Doc.h"
#include "sf31View.h"
#include "fort.h"
#include "ProgressDlg.h"
#include "Dlgstress_sp.h"
#include "Dig_prop_mem.h"
#include "Digme_spring.h"
#include "MainFrm.h"
#include "winuser.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
```

```
#endif
//    ―――――――――――――――――――――――――――――――――――――――――――――
//    ●    ウインドウメッセージマップ
//    ―――――――――――――――――――――――――――――――――――――――――――――
IMPLEMENT_DYNCREATE(CSf31View, CView)
BEGIN_MESSAGE_MAP(CSf31View, CView)
        //{{AFX_MSG_MAP(CSf31View)
        ON_WM_LBUTTONDOWN()
        ON_WM_MOUSEMOVE()
        ON_WM_LBUTTONUP()
        ON_WM_RBUTTONDOWN()
        ON_WM_RBUTTONUP()
        ON_WM_TIMER()
        ON_COMMAND(ID_SWND_ST, OnSwndSt)
        ・
        ・
        ON_COMMAND(ID_FILE_PRINT_COMENT, OnFilePrintComent)
        ON_COMMAND(ID_SWND_SECTION, OnSwndSection)
        ON_COMMAND(ID_SEC_MEM, OnSecMem)
        ON_COMMAND(ID_MEMO_SEC_CR, OnMemoSecCr)
        ON_COMMAND(ID_SWND_ISO, OnSwndIso)
        ON_WM_LBUTTONDBLCLK()
        //}}AFX_MSG_MAP
        // 標準印刷コマンド
        ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
        ON_MESSAGE(IDS_MESSAGE_WIND, OnMessageWind)
END_MESSAGE_MAP()
//    ―――――――――――――――――――――――――――――――――――――――――――――
//    ●    CSf31View クラスの構築
//    ―――――――――――――――――――――――――――――――――――――――――――――
//
CSf31View::CSf31View()
{
//    ●    ウインドウ管理データの初期設定
      m_dat_printx       = 0;
      m_dat_struct       = 0;
      m_dat_mode_number  = 1;
      m_pst_disp         = 0;
         ・
         ・
      Section.m_stress   = F_mag[4];
      Section.m_strain   = F_mag[5];
}
//    ―――――――――――――――――――――――――――――――――――――――――――――
//    ●    CSf31View クラスの消滅
//    ―――――――――――――――――――――――――――――――――――――――――――――
//
CSf31View::~CSf31View()
{

}
//    ―――――――――――――――――――――――――――――――――――――――――――――
```

```
//      ●     CSf31View クラスの破壊
//      ―――――――――――――――――――――――――――――――――――――――
//
void CSf31View::OnDestroy()
{
          HWND hwnd = this->GetSafeHwnd();
          CView::OnDestroy();
          setwindcheck_x(hwnd );
          setwindcheck( );
}
```

このクラスは、マルチウインドウの管理と図形描画処理を主に行っており、そのため多数のメンバー関数が存在する。プログラムを見ると、まず、多くのヘッダーファイルがインクルードされていることが目に付く。これら多くのヘッダーファイルの中で、このクラスに付属するsf31View.h の内容中、重要な部分を以下に示す。

最初は、他のヘッダーファイルのインクルード文が続く。これは、sf31View.h ファイルの中で、これらのクラスを参照しているからである。次に、define 文があるが、ここでは、各種のメッセージの番号を変数に割り付けている。例えば、IDS_MESSAGE_WIND は、アニメーションで他のウインドウオブジェクトにメッセージを送るときのメッセージ番号であり、実際の値は WM_USER+5 が設定されている。次の enum{}文は、FORTRAN の data 文と同様に、変数に値を割り付けている。例えば、STRUCT は１に、WAVE は２などである。

WM_USER は、VC++システムが開発者に許しているメッセージ番号であり、プログラム内の私用メッセージ番号は、これ以降の値を用いることになる。

クラスの中でメンバー関数とメンバー変数が定義されている。この中で、public:と protected:文の下で定義されているメンバー関数やメンバー変数は、各々使用方法が異なる。public:は以下で定義されている関数や変数が他のクラスでも使用可能であるが、protedted:は以下で定義されている関数や変数がこのクラスの中でしか使用できないことを宣言するものである。ここには多くの関数が含まれているが、図形処理に関連する関数はこの本の中で説明されることになる。

```
//      ―――――――――――――――――――――――――――――――――――――――
//      ●     sf31stView.h
//      ―――――――――――――――――――――――――――――――――――――――
//      ●     CSf31stView クラスの宣言およびインターフェイスの定義
//      ―――――――――――――――――――――――――――――――――――――――
//      ―――――――――――――――――――――――――――――――――――――――
//      ●     インクルードファイル
//      ―――――――――――――――――――――――――――――――――――――――
#include "sf31wave.h"
#include "sf31ftt.h"
#include "sf31load.h"
#include "sf31mode.h"
```

```
・
・
#include "Section.h"
#include "DIG_floor.h"
#include "sf31shear.h"
//    ――――――――――――――――――――――――――――――――――――――――
//    ●　インクルードファイル
//    ――――――――――――――――――――――――――――――――――――――――
#define IDS_MESSAGE_WIND   WM_USER + 5
  enum{STRUCT=1,WAVE=2,SLOAD=3,STRESS=4,MODEWD=5,DISMAX=6,DEMOD=7,SECTION=8
 };
class CSf31View : public CView
{
protected: // シリアライズ機能のみから作成します。
        CSf31View();
        DECLARE_DYNCREATE(CSf31View)
// アトリビュート
    public:
        CSf31Doc* GetDocument();
// オペレーション
//    ――――――――――――――――――――――――――――――――――――――――
//    ●　ダイアログなどのクラス定義
//    ――――――――――――――――――――――――――――――――――――――――
public:
      CSf31wave sf31wave;
      Csf31ftt sf31ftt;
      CSf31load sf31load;
       ・
       ・
      CDIG_floor  Dig_floor;
      CDig_sec    Dig_sec;
      CSection   Section;
      Csf31shear sf31shear;
//    ――――――――――――――――――――――――――――――――――――――――
//    ●　メンバー変数
//    ――――――――――――――――――――――――――――――――――――――――
      BOOL ButtonDown;
      float m_rr[4][4];
      float m_scrnpsx[3];
      float m_viewpsx[3];
      float m_scalepx[3];
      float m_scalpsx;
      int m_nstep;
      float m_base[3];
      int m_fno_xy;
      int m_dat_struct;
       ・
       ・
      int m_x2_pr;
      int m_y_pr;
      int m_han_pr;
      int m_dat_printx;  //  画面用
      float m_mlimit;
      CString m_header;
```

```
        CString m_footer;
        int F_mode_opt[21];
        int m_pst_mass;
//      ―――――――――――――――――――――――――――――――――――――――
//      ●　メンバー関数の定義
//      ―――――――――――――――――――――――――――――――――――――――

        void Pr_xy_circle(CDC* ,float);
        void Dtclear_obj() ;
        long setwindx(long,int,HWND);
        BOOL setwindcheck();
        BOOL setwindcheck_x(HWND);
        int getwindx(long);
        void disp_coment(CDC*);
        ・
        ・
        void  set_member_frame_pr(CDC*,HWND);
        void set_section_frame_pr(CDC*,HWND,int);
        int checkLG(char*,int);
        void  disp_color_bar_pr(CDC*);
        void  rectdrow(CDC*  ,int ,int ,int  ,int ,int );
        CString getheader(int);
        CString getfooter(int,int,int,int,float,float,float,CString);
// オーバーライド
// ClassWizard は仮想関数を生成しオーバーライドします。
//{{AFX_VIRTUAL(CSf31View)
  public:
    virtual void OnDraw(CDC* pDC);  // このビューを描画する際にオーバーライドされます。
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
  protected:
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnActivateView(BOOL bActivate, CView* pActivateView, CView* pDeactiveView);
    virtual void OnPrint(CDC* pDC, CPrintInfo* pInfo);
//}}AFX_VIRTUAL

// インプリメンテーション
    afx_msg LONG OnMessageWind(UINT wParam, LONG lParam);
public:
    virtual ~CSf31View();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:
// 生成されたメッセージ マップ関数
protected:
         //{{AFX_MSG(CSf31View)
         afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
         afx_msg void OnMouseMove(UINT nFlags, CPoint point);
         afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
         afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
```

```
        .
        .
         afx_msg void OnMemoSecCr();
         afx_msg void OnSwndIso();
         afx_msg void OnLButtonDblClk(UINT nFlags, CPoint point);
         //}}AFX_MSG
         DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // sf31View.cpp ファイルがデバッグ環境の時使用されます。
inline CSf31Doc* CSf31View::GetDocument()
   { return (CSf31Doc*)m_pDocument; }
#endif
```

次に、メッセージマップを見てみよう。このクラスのメッセージマップは、コードに見られるように多くの関数を含む。これらの関数は２つに分類される。ひとつは、ON_COMMAND()で始まるコードであり、第一引数はメッセージであり、第二引数はそのメッセージがこのクラスに届いたとき実行される関数である。無論、この関数はクラスのメンバー関数であり、その関数では各種の処理が行われる。また、メッセージは他のクラスを含めてプログラムの中で発せられる。

次は、ON_WM_RBUTTONDOWN()のように ON_WM で始まる関数であり、これらの関数は、ウインドウからのメッセージによって駆動される。例えば、ON_WM_RBUTTONDOWN()がメッセージで駆動されると、メンバー関数 OnRButtonDown(UINT nFlags, CPoint point)が実行される。これらのメッセージは、主に、ユーザーのマウス操作などによって発せられ、アクティブとなっている子ウインドウのオブジェクトに届くようなメカニズムとなっている。

次に、各メンバー関数について説明しよう。まず、関数 CSf31View()のコンストラクタとデストラクタについてであるが、コンストラクタ CSf31View::CSf31tView()では、このオブジェクトを制御するデータ群の初期設定を行う。特に、アニメーションの時刻や構造透視図のスケール、その他の図形処理に関する多くの初期設定が行われる。

子ウインドウが閉じられるとき、デストラクタがオジェクトの消去処理をする前に、関数 OnDestroy()がコールされ、その中でウインドウ管理に関する２つの関数が呼ばれる。

```
void CSf31View::setwindcheck()
BOOL CSf31View::setwindcheck_x(HWND hWnd)
```

子ウインドウを閉じるようユーザーからの指示があったとき、この関数

OnDestroy()が呼ばれ、ウインドウ管理のための処理を行う。まず、そのウインドウのハンドルを取得した後、Cview クラスのメンバー関数 OnDestroy()をコールする。次に、メンバー関数 setwindcheck_x()を用いて、先に取得したウインドウハンドルを頼りに子ウインドウを管理する配列からこのウインドウを抹消する。さらに、関数 setwindcheck()では、管理情報の中で、既にゾンビとなっているウインドウ情報をチェックし、抹消する。これらの関数については、次節で説明する。

3.4 ウインドウ管理システム

SPACE のプレゼンターシステムでは､多くの子ウインドウが表示され､各種の図形が描画される。本節では、この子ウインドウを管理するためのシステムを説明しよう。

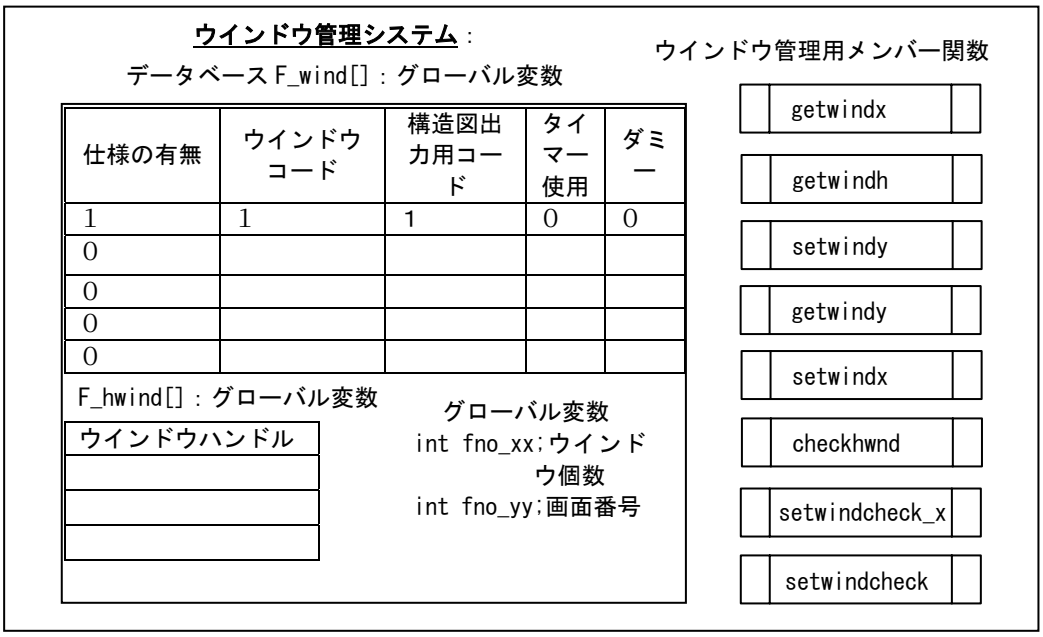

図 3-4 マルチウインドウの管理法

ウインドウ管理は、グローバル配列 F_wind[100][5]と F_hwind[100]を用いてクラス sf31View で行っている。この２つのグローバル配列は、sf31dat.h で定義しており、100 画面分用意されている。その仕様は、

```
//  ―――――――――――――――――――――――――――
//  ●  ウインドウコード
//  ―――――――――――――――――――――――――――
//      fno_yy :: 現在開いているウインドウ数
//      F_wind[iix-1][0]::0:空き、1:現在使用
//      F_wind[iix-1][1]::ウインドウコード
//      STRUCT=1,WAVE=2,SLOAD=3,STRESS=4,MODEWD=5,DISMAX=6,DEMOD=7,SECTION=8
//      F_wind[iix-1][2]= :構造図出力用コード（0-5）
//        0:オプション図形;1:
//      F_wind[iix-1][3]=0:ダミー
```

```
//        F_wind[iix-1][4]=0:ダミー
//        F_hwind[iix-1] = hWnd : ウインドウのハンドルを保存
//    ―――――――――――――――――――――――――――――
```

となっており、この仕様にしたがって関数が定義されている。また、子ウインドウ一つひとつに存在するハンドル内にひとつの long のデータ領域があり、この変数域を利用してウインドウを管理する。この変数領域へのデータセットと取り出しは、以下の関数を用いる。この中で、hwnd はウインドウハンドルであり、設定する fno_zz は画面番号である。

```
SetWindowLong(hwnd,GWL_USERDATA,fno_zz);
GetWindowLong(hwnd,GWL_USERDATA);
```

このクラスのメンバー関数の中で、ウインドウ管理に関連する関数を以下に示し、説明する。ウインドウ管理は以下の８つのメンバー関数から構築されている。

```
CSf31View::getwindx(long fno)
CSf1tView::getwindh(long fno)
CSf31View::setwindy(long fno,int ii)
CSf31View::getwindy(long fno)
CSf31View::setwindx(long fno,int ii,HWND hWnd)
CSf31View::checkhwnd(HWND hWnd,int jj)
CSf31View::setwindcheck_x(HWND hWnd)
CSf31View::setwindcheck()
```

上記の関数について、その機能を簡単に説明する。

１）関数 getwindx()では、画面番号を引数にして、対応するウインドウコードをリターン値とする。ウインドウコードがない場合は、-1 とする。

２）関数 getwindh()は、画面番号を引数にして、対応するウインドウのハンドルをリターン値とする。該当するウインドウがない場合は、NULL を返す。

３）関数 setwindy()は、画面番号と構造図出力用コードを引数にして、該当する配列 F_wind[]に構造図出力用コードをセットする。

４）関数 getwindy()は、画面番号を引数にして、該当する構造図出力用コードを取り出す。

５）関数 setwindx()は、画面番号とウインドウコードとそのウインドウのハンドルを引数にして、該当する配列 F_wind[]にウインドウコードとそのウインドウのハンドルをセットする。もし、該当する

ウインドウがない場合は、配列 F_wind[]内の空の領域か、もしくは、追加領域に各パラメータをセットする。

６）関数 checkhwnd()は、画面番号とハンドルを引数にして、該当するハンドルのウインドウが現在生きているかどうか、つまり、消去されていないかどうかを確かめ、もし、消去されているならば配列 F_wind[]と F_hwind[]の該当する位置のデータをクリアする。

７）関数 setwindcheck_x()は、引数としてハンドルを渡し、該当する領域の F_wind[]と F_hwind[]をクリアする。ここで、このハンドルに対応するウインドウがアニメーションを実行するウインドウであると、解析タイマーを F_ontimer = 0 として、動的処理を中止する。

８）関数 setwindcheck()は、全ウインドウについて、クローズしたウインドウがないかどうかチェックする。もし該当するウインドウがあれば、配列 F_wind[]と F_hwind[]中の該当するデータをクリアする。

これら８つのプログラムを以下に示す。その内容は簡単なので容易に理解できよう。

```
//    ―――――――――――――――――――――――――――――――――――――――
//      ●　ウインドウ管理システム
//    ―――――――――――――――――――――――――――――――――――――――
//      ●　ウインドウコードの取得
//    ―――――――――――――――――――――――――――――――――――――――
 int CSf31View::getwindx(long fno)
{
      int iix ;
      iix = (int) fno -1;
      if(iix >= 0 && iix < fno_yy) {
        if( F_wind[iix][0] == 1){
          iix = F_wind[iix][1];
          return iix;
      }}
      iix= -1;
      return iix;
}
//    ―――――――――――――――――――――――――――――――――――――――
//      ●　ウインドウのハンドル取得
//    ―――――――――――――――――――――――――――――――――――――――
 HWND CSf31View::getwindh(long fno)
{
        int iix ;
        iix = (int) fno -1;
        if(iix >= 0 && iix < fno_yy) {
          if( F_wind[iix][0] == 1){
          return  (F_hwind[iix]);
```

```
        }}
  return (NULL);
}
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●　構造図出力用コードのセット
//    ―――――――――――――――――――――――――――――――――――――――――
void CSf31View::setwindy(long fno,int ii)
{
        int iix = (int) fno;
        F_wind[iix-1][2]=ii;
}
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●　構造図出力用コードを取得
//    ―――――――――――――――――――――――――――――――――――――――――
int CSf31View::getwindy(long fno)
{
        int iix = (int) fno;
        return F_wind[iix-1][2];
}
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●　ウインドウの初期セット及ぶコード、ハンドルセット
//    ―――――――――――――――――――――――――――――――――――――――――
long CSf31View::setwindx(long fno,int ii,HWND hWnd)
{
        int i;
        int iix;
        iix = (int) fno;
        if(fno_yy != 0){
//    ―――――――――――――――――――――――――――――――
//      ● 指定した領域にデータをセット
//    ―――――――――――――――――――――――――――――――
        if(iix > 0 && iix <= fno_yy) {
          F_wind[iix-1][0]=1;
          F_wind[iix-1][1]=ii;
          F_wind[iix-1][2]=0;
          F_wind[iix-1][3]=0;
          F_wind[iix-1][4]=0;
          F_hwind[iix-1] = hWnd;
          return fno;
        }
//    ―――――――――――――――――――――――――――――――
//      ●　空きとなっている制御データ領域を探索し、
//        あればそこにデータをセット
//    ―――――――――――――――――――――――――――――――
     for(i = 0; i<fno_yy; i++){
       if(F_wind[i][0] == 0 ){
          F_wind[i][0]=1;
          F_wind[i][1]=ii;
          F_wind[i][2]=0;
          F_wind[i][3]=0;
          F_wind[i][4]=0;
          F_hwind[i] = hWnd;
          return i+1;
      }}
```

```
}
//    ―――――――――――――――――――――――――――――――――――――
//      ●　新規ウインドウ制御データ領域を確保
//    ―――――――――――――――――――――――――――――――――――――
              F_wind[fno_yy][0]=1;
              F_wind[fno_yy][1]=ii;
              F_wind[fno_yy][2]=0;
              F_wind[fno_yy][3]=0;
              F_wind[fno_yy][4]=0;
              F_hwind[fno_yy] = hWnd;
              fno_yy = fno_yy + 1;
              fno = fno_yy;
  return fno;
}
//    ―――――――――――――――――――――――――――――――――――――――――――――――――――――
//      ●　ウインドウが生きているかどうかチェック
//    ―――――――――――――――――――――――――――――――――――――――――――――――――――――
BOOL CSf31View::checkhwnd(HWND hWnd,int jj)
{
            BOOL ihan = IsWindow(hWnd);
            if(ihan) return (ihan);
            F_wind[jj][0]=0;
            F_wind[jj][1]=0;
            F_wind[jj][2]=0;
            F_wind[jj][3]=0;
            F_wind[jj][4]=0;
            F_hwind[jj] = NULL;
return (ihan);
}
//    ―――――――――――――――――――――――――――――――――――――――――――――――――――――
//      ●　閉じたウインドウの制御データ領域を解放する
//    ―――――――――――――――――――――――――――――――――――――――――――――――――――――
BOOL CSf31View::setwindcheck_x(HWND hWnd)
{
        BOOL ihan = FALSE;
        if(fno_yy != 0){
        for(int i = 0; i<fno_yy; i++){
        if(F_wind[i][0] != 0 ){
        if(F_hwind[i] == hWnd){
        if(hWnd == F_hwnd_timer  ) F_ontimer = 0;
          F_wind[i][0]=0;
          F_wind[i][1]=0;
          F_wind[i][2]=0;
          F_wind[i][3]=0;
          F_wind[i][4]=0;
          F_hwind[i] = NULL;
          ihan = TRUE;
          }
        }}}
return (ihan);
}
//    ―――――――――――――――――――――――――――――――――――――――――――――――――――――
//      ●　閉じたウインドウの制御データ領域を解放する
//    ―――――――――――――――――――――――――――――――――――――――――――――――――――――
```

```
BOOL CSf31View::setwindcheck()
{
          BOOL ihan = FALSE;
          HWND hWnd;
        if(fno_yy != 0){
          for(int i = 0; i<fno_yy; i++){
            if(F_wind[i][0] != 0 ){
              hWnd = F_hwind[i];
              if(checkhwnd(hWnd,i)){
              ihan = TRUE;
             }
        }}}
return (ihan);
}
```

3.5 ウインドウ管理の使用法

ウインドウに新規に図形を描画するとき、あるいは他の図形を再描画するときに、ウインドウ管理システムにウインドウコードが登録される。また、OnDraw 関数で図形を再描画するとき、アニメーションで図形を更新するとき、そのウインドウに何が表示されているかを知るために、管理システムからウインドウコードを取り出す。このようにウインドウ内の図形が再描画されるとき、管理システムが利用されるわけである。

初期設定：CMainFrame クラス
fno_yy=0;// ウインドウ個数

ウインドウ管理用データベース
F_wind：
F_hwind

登録：

```
m_dat_struct = 1;
long fno_yx = GetWindowLong(hwnd,GWL_USERDATA);
int ii=STRUCT;
long fno_zz =   setwindx(fno_yx,ii,hwnd);
if(fno_yx != fno_zz) SetWindowLong(hwnd,GWL_USERDATA,fno_zz);
int fno_xxx = 1;
setwindy(fno_zz,fno_xxx);
ii=fno_zz-1;
```

取り出し：

```
HWND hwnd = this->GetSafeHwnd();
if(IsWindow(hwnd) == FALSE) return;
long fno_yx = GetWindowLong(hwnd,GWL_USERDATA);
int fno_xy  = getwindx(fno_yx);
int fno_xxx = getwindy((long)fno_yx);
if(fno_xy == -1) return;
int ii=fno_yx-1;
```

図 3-5 マルチウインドウ管理の使用法

構造用ウインドウを作成する関数 CSf3stView::OnSwndSt()を用いて、ウインドウ管理システムにウインドウコードを登録する方法を説明し

よう。ここで、例として構造図形を初期描画する関数 OnSwndSt()取り上げ、登録方法を詳細に観察する。以下に、関数 OnSwndSt()を示す。

```
//    ―――――――――――――――――――――――――――――――――――――――
//      ●    OnSwndSt:節点変位の入力
//    ―――――――――――――――――――――――――――――――――――――――
//
void CSf31View::OnSwndSt()
{
//    ―――――――――――――――――――――――――――――――――
//      ●  構造用ウインドウ作成のための初期設定
//    ―――――――――――――――――――――――――――――――――
    HWND hwnd = this->GetSafeHwnd();
    if(IsWindow(hwnd) == FALSE) return;
//    ―――――――――――――――――――――――――――――――――
//      ●  動的領域の確保
//    ―――――――――――――――――――――――――――――――――
     if(F_read_disp == 0){
       F_disp = new float[3*node*F_all_step];
         if(F_disp == 0){
           MessageBox("Sorry! Memory allocation failure.");
           return;
         }
      F_read_disp=1;
//    ―――――――――――――――――――――――――――――――――
//      ●  ウインドウデータの初期セット
//    ―――――――――――――――――――――――――――――――――
         m_pst_dis = 0;
         int ierr =0;
         int jj;
         int inputx =1;
         int all_step =F_all_step;
//    ―――――――――――――――――――――――――――――――――
//      ●  節点変位データの入力
//    ―――――――――――――――――――――――――――――――――
         INDISP(&inputx,F_disp,&ierr,&jj,&node,&all_step);
         if(ierr == 1) {
           m_dat_struct=0;
           F_read_disp=0;
           m_nstep =0;
           MessageBox(" Sorry! System did not open this file.");
           return;
         }
//    ―――――――――――――――――――――――――――――――――
//      ●  データ用プログレスバーの作成
//    ―――――――――――――――――――――――――――――――――
         int p_input=0;
         CProgressDlg pdlg(p_input,all_step);
       if(pdlg.DoModal()== IDOK){
         m_nstep=F_mnstep;
       }else{
         m_dat_struct=0;
         F_read_disp=0;
```

```
            m_nstep =0;
            return;
          }
        }else{
          m_nstep=F_mnstep;
        }
//      ―――――――――――――――――――――――――――――――
//        ●　ウインドウ管理システムにコードを登録
//      ―――――――――――――――――――――――――――――――
        m_dat_struct = 1;
        long fno_yx = GetWindowLong(hwnd,GWL_USERDATA);
        int ii=STRUCT;
        long fno_zz =   setwindx(fno_yx,ii,hwnd);
        if(fno_yx != fno_zz) SetWindowLong(hwnd,GWL_USERDATA,fno_zz);
        int fno_xxx = 1;
        setwindy(fno_zz,fno_xxx);
        ii=fno_zz-1;
//      ―――――――――――――――――――――――――――――――
//        ●　回転行列の設定と透視変換を行う
//      ―――――――――――――――――――――――――――――――
        for(int i=0;i<3;i++){
          m_scrnpsx[i]=F_scrnps[i];
          m_viewpsx[i]=0;
          m_scalepx[i]=F_scalep[i];
        }
        m_scalpsx=F_scalps;
        ROTSET(&m_scrnpsx[0],&F_viewps[0],&m_scalpsx,&m_rr[0][0],&m_viewpsx[0]);
        int fno_xy = getwindx((long)fno_yx);
//      ―――――――――――――――――――――――――――――――
//        ●　ヘッダーとフッターを書く
//      ―――――――――――――――――――――――――――――――
        int idm =1;
        float dm;
        m_header = getheader(idm);
        CString buffer =(CString) F_title;
        m_footer = getfooter(idm,idm,  idm,idm,  dm,dm,dm,buffer);
//      ―――――――――――――――――――――――――――――――
//        ●　このウインドウ領域を再描画
//      ―――――――――――――――――――――――――――――――
        CRect rcFrame;
        GetClientRect(&rcFrame);
        InvalidateRect( rcFrame ,TRUE);
}
```

ウインドウ登録システムにウインドウコードを登録する。

ユーザーが構造図を表示しようとして、新しい子ウインドウを作成したと仮定し、ここから処理の内容を追ってみることにしよう。表示された子ウインドウは、まだ何も表示されず真っ白の状態であり、このウインドウも管理されていない状態である。そこで、ユーザーは、この子ウインドウを、プルダウンメニューを用いて構造画面ウインドウに設定しようとする。すると、この関数 CSf3stView::OnSwndSt()がコールされ、

以下の処理が実行されることになる。

まず、このウインドウのハンドルを取得し、hwnd にセットする。さらに、表示用の節点変位を入れる配列 F_disp[]を動的に確保する。動的確保に失敗するとエラー表示を行った後、関数を抜ける。成功するとサブルーチン INDISP()で節点変位をファイルから入力する。その際、関数 pdlg.DoModal()によって、プログレスバーを表示する。いよいよ、次からが、ウインドウコードを登録する部分である。

最初に VC++の MSF である関数 GetWindowLong()で、渡したハンドルから画面番号を取り出す。ウインドウがまだ何も表示されていない場合は、取り出した画面番号は 0 である。ここで、画面番号とは、管理用配列の要素番号+1 の値である。ただし、他のグラフなどが既に表示がされていると、その画面番号が変数 fno_yx に取り出される。次に、ウインドウコード(この場合は STRUCT)を ii にセットした後、メンバー関数 setwindx()をコールする。この関数への受け渡し引数は、画面番号 fno_yx、ウインドウコード ii とこのウインドウのハンドル hwnd であり、戻ってくる関数値は画面番号である。次に、関数 setwindy()をコールし、構造図出力用コードをセットする。

メンバー関数 setwindx()では、まず、設定ウインドウの最大値 fno_yy がゼロであるかどうかチェックする。ゼロの場合は、直ちに登録処理に移る。ゼロでない場合は、登録しようとする画面番号が既成のウインドウであるかどうかチェックする。この画面が既に登録された既成のウインドウである場合は、F_wind にデータをセットし、リターン値として画面番号を返し、関数を閉じる。

ウインドウを登録する場合は、まず、空きとなっている領域を探索する。もし、あればそこに登録し、その画面番号をリターン値として返し、関数を閉じる。空きがない場合は、最大画面番号を１増やし、そこに登録し、その番号をリターン値として返し、関数を閉じる。

元の CSf3stView::OnSwndSt()の解説に戻ろう。関数 setwindx()からの戻り値と元の画面番号が同じであれば、その次のステップに進むが、異なると SetWindowLong()関数を用いて、画面番号をこのウインドウハンドルにセットする。これで、子ウインドウの実態であるハンドルとシステムが管理するウインドウ管理配列との結合が完了し、設定が終了する。

次からは、透視図用のデータをセットし、サブルーチン ROTSET()を用いて回転行列を求める通常の処理となる。次に、setwindy()を用いて、構造データを透視図変換する。また、構造画面ウインドウのヘッダーと

フッターをセットする。最後に、InvalidateRect()関数を用いて、当該の子ウインドウを書き換えると、構造図が初めて描画されることになる。

以上で、子ウインドウをグローバル配列で初期化、管理する手法を示した。しかし、何故このような管理が必要となるのか。次にその利用を示す。先に述べた関数 InvalidateRect()は、ウインドウの再描画を促すが、それはどこのプログラムで再描画するのか。あるいはウインドウを大きくしたり、小さくしたり、また、裏にあったウインドウが表になるとき、やはり、ウインドウの再描画が生じる。これら再描画は、オペレーションシステムが自動的にメンバー関数 OnDraw(CDC* pDC)をコールすることによって行われる。ウインドウが再描画されるとき、そのウインドウに何が描かれていたかをオブジェクトが知っている必要がある。例えば、構造図であったり、地震加速度波形であったり、また、節点変位であったりする。あるいは、ウインドウがアイコンになっているかもしれない。このためにもウインドウの管理が必要となる。

メンバー関数 OnDraw()で該当する部分を以下に示す。ここでは、ウインドウ管理システムを利用して、そのウインドウのコードを取り出す。

```
void CSf31View::OnDraw(CDC* pDC)
{
    HWND hwnd = this->GetSafeHwnd();
    if(IsWindow(hwnd) == FALSE) return;
    long fno_yx = GetWindowLong(hwnd,GWL_USERDATA);
    int fno_xy  = getwindx(fno_yx);
    int fno_xxx = getwindy((long)fno_yx);
    if(fno_xy == -1) return;
    int ii=fno_yx-1;
//------------------------------------------------  ● プリンターによる描画
    if(pDC->IsPrinting())
     .
     .
}
```

ウインドウ登録システムからウインドウコードを取り出す。

まず、このウインドウのハンドルを取得する。そのハンドルを用いて、関数 GetWindowLong()で画面番号を取得する。次に、この画面番号を引数にして、関数 getwindx()でウインドウコード fno_xy を、関数 getwindy()で構造図出力用コード fno_xxx を取り出す。この２つのコードと画面番号を用いて、このウインドウに描画する図形が選択される。

このウインドウ管理データベースの初期設定は、CMainFrame クラスで次のように行う。

```
//    ------------------------------------------------------
//    ●    CMainFrame クラスのコンストラクタ
```

```
//     ―――――――――――――――――――――――――――――――――――――――――――――
//
CMainFrame::CMainFrame()
{
//――――――――――――――――――――――――――――――――――――――――――――――― ●
    int ihan,ndim,nnode;
   fno_yy=0;
  F_time_ii =0;
  F_Speed = 4;
  Fl_Speed = 1;
   ・
   ・
}
```

ウインドウ登録数を０とする初期設定を行う。