第５章　図形処理入門その２

ポイント：ウインドウ上にグラフを描く

１．ダイアログから図形情報を取得する

２．プルダウンメニューとメッセージ

３．各種の図形処理

5.1 はじめに

本章では、第２章に続いて、典型的な図形処理手法を学ぶ。最初はダイアログから図形情報を取得し、図形に反映させる方法、次にプルダウンメニューやその他の基本的な図形処理方法について解説する。

5.2 プロパティダイアログと図形の関係

プレゼンターで構造の変形や応力状態を示す図形を描くとき、プロパティを用いて、各種の条件を変更することができる。本節では、この条件を設定するダイアログとそのパラメータについて述べ、図形に反映させる方法について説明する。条件を変更することのできるダイアログは、２種類用意されており、ひとつは、図 5-1 に示される変形などの倍率を変更する倍率プロパティダイアログであり、他の一つは、図 5-3 に示される解析表示選択ダイアログである。倍率のプロパティは、図 5-1 に示す 12 種類のパラメータを変更することができる。特に、最後のアニメーションスピードとコマ（表示コマ間隔）、つまり、再描画の間隔は、動的解析用プレゼンターでは威力を発揮する。また、応力とひずみは、ファイバーモデルの断面応力状態を表示するときに使用するパラメータである。

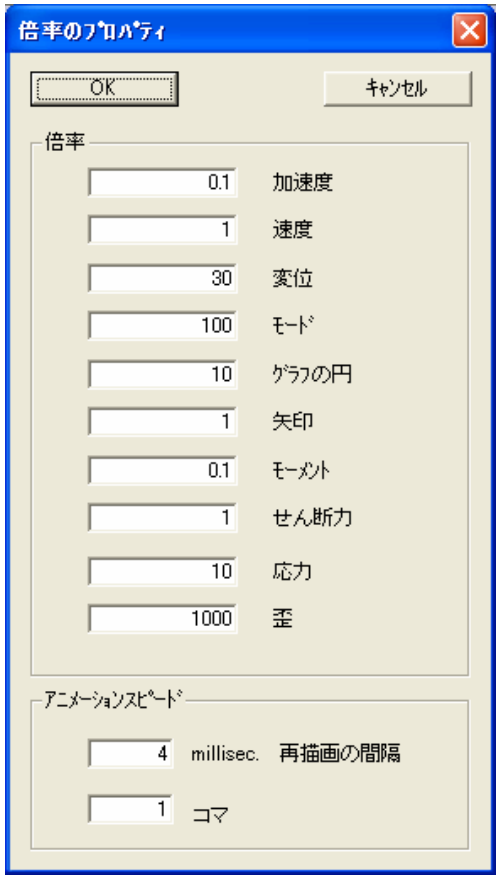

図 5-1 倍率プロパティダイアログ

この倍率プロパティを扱うクラスは、CDig_mag であり、そのクラス全体のプログラムを以下に示す。

```
#include "stdafx.h"
#include "sf31.h"
#include "dig_mag.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
// Cdig_mag ダイアログ
/////////////////////////////////////////////////////////////////////////////
Cdig_mag::Cdig_mag(CWnd* pParent /*=NULL*/)
        : CDialog(Cdig_mag::IDD, pParent)
{
        //{{AFX_DATA_INIT(Cdig_mag)
        m_edit_dis_1 = 1.0f;                    // 加速度を表示する場合の大きさ
        m_edit_dis_2 = 1.0f;                    // 速度を表示する場合の大きさ
        m_edit_dis_3 = 1.0f;                    // 変位を表示する場合の大きさ
        m_edit_dis_4 = 1.0f;                    // モード変位を表示する場合の大きさ
        m_edit_dis_5 = 0;                       // アニメーションスピード
        m_edit_disp_6 = 0.0f;                   // グラフの円を表示する場合の大きさ
        m_edit_arrow = 1.0f;                    // 反力などの矢印
        m_edit_bend = 1.0f;                     // 部材曲げモーメント
        m_edit_shear = 1.0f;                    // 部材のせん断力
        m_edit_strain = 1000.0f;                // ファイバー断面のひずみ
        m_edit_stress = 10.0f;                  // ファイバー要素の応力
        m_edit_dis_7 = 1;                       // アニメーション表示コマ間隔
        //}}AFX_DATA_INIT
}
void Cdig_mag::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(Cdig_mag)
        DDX_Text(pDX, IDC_EDIT_DISPLACE, m_edit_dis_1);
        DDX_Text(pDX, IDC_EDIT_DISPLACE2, m_edit_dis_2);
        DDX_Text(pDX, IDC_EDIT_DISPLACE3, m_edit_dis_3);
        DDX_Text(pDX, IDC_EDIT_DISPLACE4, m_edit_dis_4);
        DDX_Text(pDX, IDC_EDIT_DISPLACE5, m_edit_dis_5);
        DDX_Text(pDX, IDC_EDIT_DISPLACE6, m_edit_disp_6);
        DDX_Text(pDX, IDC_EDIT_ARROW, m_edit_arrow);
        DDX_Text(pDX, IDC_EDIT_BEND, m_edit_bend);
        DDX_Text(pDX, IDC_EDIT_SHEAR, m_edit_shear);
        DDX_Text(pDX, IDC_EDIT_STRAIN, m_edit_strain);
        DDX_Text(pDX, IDC_EDIT_STRESS, m_edit_stress);
        DDX_Text(pDX, IDC_EDIT_SPEED, m_edit_dis_7);
        //}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(Cdig_mag, CDialog)
        //{{AFX_MSG_MAP(Cdig_mag)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

パラメータであるメンバー変数一覧とその意味は、このクラスのコン

ストラクタでコメントとして書かれている。このクラスの記述を見れば分かるように、ほとんど何も処理するコードがない。コンストラクタの次に書かれている関数 DoDataExchange()は、ダイアログで書き込まれたデータをメンバー変数に書き換えるもので、このダイアログを閉じるときなどにコールされる。

これらのパラメータを具体的に利用しているクラスは、CSf31View であり、該当する部分のプログラムコードを以下に示す。まず、ツールバー上の倍率変更ツールチップを押すことで、メッセージ ID_DYN_MAG が発信される。このメッセージを受けて、このクラスのメッセージマップにある ON_COMMAND()関数がメンバー関数 OnDynMag()をコールする。後は、この関数が処理することになる。

```
BEGIN_MESSAGE_MAP(CSf3stView, CView)
        ON_COMMAND(ID_DYN_MAG, OnDynMag)
END_MESSAGE_MAP()
```

以下に、メンバー関数 OnDynMag()を示す。

```
//    ―――――――――――――――――――――――――――――――
//      ●　描画スケールの設定処理
//    ―――――――――――――――――――――――――――――――
void CSf31View::OnDynMag()
{
          if(F_ontimer == 1) return;
           UpdateData(TRUE);
           dig_mag.m_edit_dis_1  = F_scalep[0];
           dig_mag.m_edit_dis_2  = F_scalep[1];
           dig_mag.m_edit_dis_3  = F_scalep[2];
           dig_mag.m_edit_dis_4  = F_scalep[3];
           dig_mag.m_edit_disp_6 = m_mag;
           dig_mag.m_edit_dis_5  = F_Speed;
           dig_mag.m_edit_dis_7  = FI_Speed;
           dig_mag.m_edit_arrow  = m_arrow;
           dig_mag.m_edit_bend   = m_bend;
           dig_mag.m_edit_shear  = m_shear;
           dig_mag.m_edit_strain = m_strain;
           dig_mag.m_edit_stress = m_stress;
          UpdateData(FALSE);
       if(dig_mag.DoModal() == IDOK){
           UpdateData(TRUE);
           F_scalep[0] = dig_mag.m_edit_dis_1;
           F_scalep[1] = dig_mag.m_edit_dis_2;
           F_scalep[2] = dig_mag.m_edit_dis_3;
           F_scalep[3] = dig_mag.m_edit_dis_4;
           m_mag       =  dig_mag.m_edit_disp_6;
           F_Speed      = dig_mag.m_edit_dis_5;
           FI_Speed     = dig_mag.m_edit_dis_7;
           m_arrow      = dig_mag.m_edit_arrow;
           m_bend       = dig_mag.m_edit_bend;
```

```
            m_shear      = dig_mag.m_edit_shear;
            m_stress     = dig_mag.m_edit_stress;
            m_strain     = dig_mag.m_edit_strain;
            Section.m_stress = m_stress;
            Section.m_strain = m_strain;
            UpdateData(FALSE);
//          MessageBeep((WORD) -1);
}
}
```

それでは、まずこの関数を見ていこう。この関数は、アニメーションが実際に動作しているときは、処理できないように設定されており、パラメータ F_ontimer が１のときは、処理を中止する。次に CDig_mag クラスのメンバー変数にCSf３１1View クラスのメンバー変数からデータをセットし、関数 dig_mag.DoModal()でダイアログを表示した後、ユーザーがダイアログを用いてデータを入力する。その後、逆に CDig_mag クラスのメンバー変数から、CSf31View のメンバー変数にデータをセットする。後は、このパラメータを用いて透視図を描画することになる。これらのパラメータは、CSf31View のコンストラクタで初期設定され、その初期データは､関数 CMainFrame()のファイル入力でデータを受け取っている。また、変数 m_arrow、 m_bend、m_shear、 m_strain、 m_stress は、CSf31View クラスのメンバー変数であるが、他の変数はグローバル変数である。そのため、メンバー変数のパラメータは当該のウインドウにのみ影響を与え、他のパラメータは全てのウインドウ描画に影響を与える。

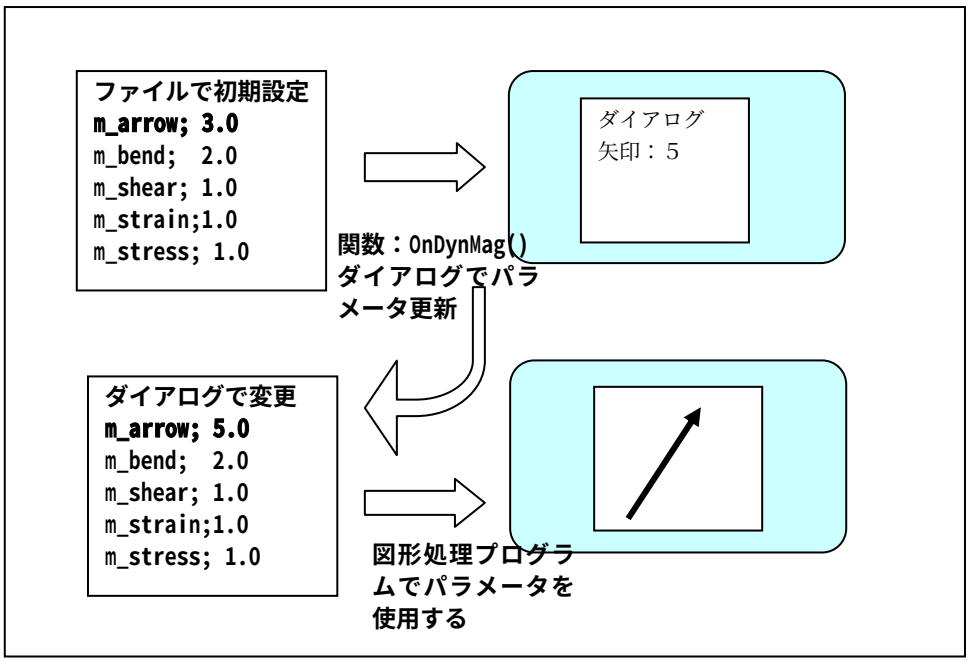

図 5-2 ダイアログと図形の関係

図5-3解析結果表示選択ダイアログ

次に、図 5-3 に示す解析結果表示選択ダイアログとその処理法について解説する。このダイアログでは、多くのパラメータを設定・変更することができる。まず、このダイアログを管理するクラス CDig_pstruct に関するプログラムを以下に示す。パラメータであるメンバー変数一覧とその意味は、このクラスのコンストラクタでコメントとして書かれている。

```
#include "stdafx.h"
#include "sf3st.h"
#include "Dig_pstruct.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
CDig_pstruct::CDig_pstruct(CWnd* pParent /*=NULL*/)
        : CDialog(CDig_pstruct::IDD, pParent)
{
        //{{AFX_DATA_INIT(CDig_pstruct)
        m_radio_color = -1;                     // カラー表示の有無選択パラメータ
        m_radio_hinge = -1;                     // 塑性ヒンジ表示の有無選択パラメータ
        m_radio_displacement = -1;              // 変位表示方向選択パラメータ
        m_radio_graph = -1;                     // 応力などのグラフ選択パラメータ
        m_check_mem_used = FALSE;               // グループ表示の有無選択パラメータ
```

```
        m_edit_mem10 = 0;                       // グループ番号その１
        m_edit_mem11 = 0;                       // グループ番号その２
        m_edit_mem22 = 0;                       // グループ番号その３
        m_edit_mem33 = 0;                       // グループ番号その４
        m_edit_mem44 = 0;                       // グループ番号その５
        m_edit_mem55 = 0;                       // グループ番号その６
        m_edit_mem66 = 0;                       // グループ番号その７
        m_edit_mem77 = 0;                       // グループ番号その８
        m_edit_mem88 = 0;                       // グループ番号その９
        m_edit_mem99 = 0;                       // グループ番号その１０
        m_edit_x1 = 0.0f;                       // 透視図原点（ｘ）
        m_edit_x2 = 0.0f;                       // 透視図原点（ｙ）
        m_edit_x3 = 0.0f;                       // 透視図原点（ｚ）
        m_edit_p2 = 0.0f;                       // 応力表示の下位限界
        m_edit_p1 = 0.0f;                       // 応力表示の上位限界
        m_radio_x1 = -1;                        // 透視図原点移動パラメータ（ｘ）
        m_radio_x2 = -1;                        // 透視図原点移動パラメータ（ｙ）
        m_radio_x3 = -1;                        // 透視図原点移動パラメータ（ｚ）
        m_radio_arrow = -1;                     // 反力矢印表示有無パラメータ
        m_radio_dis = 0;                        // 表示種類パラメータ
        m_edit_mlimit = 0.0f;                   // 応力表示制限値
        m_edit_mass = 0;                        // 質量位置の表示パラメータ
        //}}AFX_DATA_INIT
}
void CDig_pstruct::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CDig_pstruct)
        DDX_Radio(pDX, IDC_RADIO_COLOR, m_radio_color);
        DDX_Radio(pDX, IDC_RADIO_HINGE, m_radio_hinge);
        DDX_Radio(pDX, IDC_RADIO_DISPLACEMENT, m_radio_displacement);
        DDX_Radio(pDX, IDC_RADIO_GRAPH, m_radio_graph);
        DDX_Check(pDX, IDC_CHECK_MEM__USED, m_check_mem_used);
        DDX_Text(pDX, IDC_EDIT_MEM10, m_edit_mem10);
        DDX_Text(pDX, IDC_EDIT_MEM11, m_edit_mem11);
        DDX_Text(pDX, IDC_EDIT_MEM22, m_edit_mem22);
        DDX_Text(pDX, IDC_EDIT_MEM33, m_edit_mem33);
        DDX_Text(pDX, IDC_EDIT_MEM44, m_edit_mem44);
        DDX_Text(pDX, IDC_EDIT_MEM55, m_edit_mem55);
        DDX_Text(pDX, IDC_EDIT_MEM66, m_edit_mem66);
        DDX_Text(pDX, IDC_EDIT_MEM77, m_edit_mem77);
        DDX_Text(pDX, IDC_EDIT_MEM88, m_edit_mem88);
        DDX_Text(pDX, IDC_EDIT_MEM99, m_edit_mem99);
        DDX_Text(pDX, IDC_EDIT_X, m_edit_x1);
        DDX_Text(pDX, IDC_EDIT_X2, m_edit_x2);
        DDX_Text(pDX, IDC_EDIT_X3, m_edit_x3);
        DDX_Text(pDX, IDC_EDIT_P2, m_edit_p2);
        DDX_Text(pDX, IDC_EDIT_P1, m_edit_p1);
        DDX_Radio(pDX, IDC_RADIO_X, m_radio_x1);
        DDX_Radio(pDX, IDC_RADIO_X2, m_radio_x2);
        DDX_Radio(pDX, IDC_RADIO_X3, m_radio_x3);
        DDX_Radio(pDX, IDC_RADIO_ARROW, m_radio_arrow);
        DDX_Radio(pDX, IDC_RADIO_DIS, m_radio_dis);
        DDX_Text(pDX, IDC_EDIT_MLIMIT, m_edit_mlimit);
```

```
        DDX_Radio(pDX, IDC_RADIO_MASS, m_edit_mass);
        //}}AFX_DATA_MAP
}
}
```

このクラスも非常に単純で、コンストラクタとデータ交換を行うCDialog::DoDataExchange(pDX)関数のみである。これらのパラメータを具体的に利用しているクラスは、やはりCSf31Viewであり、該当する部分のプログラムコードを以下に示す。まず、ユーザーが該当するウインドウでマウスを右クリックすると、先に述べたポップアップメニューが表示され、その中のプロパティが選択されると、メッセージID_WND_PROPが発信される。このメッセージによって、このクラスのメッセージマップにあるON_COMMAND()が、メンバー関数OnWndProp()をコールする。

```
BEGIN_MESSAGE_MAP(CSf3stView, CView)
   ON_COMMAND(ID_WND_PROP, OnWndProp)
END_MESSAGE_MAP()
```

以下に、メンバー関数OnWndProp()の一部を示す。

```
//    ――――――――――――――――――――――――――――――――
//      ●　プロパティ処理
//    ――――――――――――――――――――――――――――――――
void CSf31View::OnWndProp()
void CSf31View::OnWndProp()
{
//    ――――――――――――――――――――――
//      ●　ウインドウコードの取得
//    ――――――――――――――――――――――
      HWND hwnd = this->GetSafeHwnd();
      if(IsWindow(hwnd) == FALSE) return;
      long fno_yx = GetWindowLong(hwnd,GWL_USERDATA);
      int fno_xy = getwindx((long)fno_yx);
//    ――――――――――――――――――――――
//      ●　構造データプロパティを取得
//    ――――――――――――――――――――――
   if(fno_xy == STRUCT && (m_dat_struct == 1 || m_dat_struct == 2)){
      UpdateData(TRUE);
      Dlg_pstruct.m_radio_displacement=m_pst_disp;
      Dlg_pstruct.m_radio_graph=m_pst_graph;
      Dlg_pstruct.m_radio_arrow=m_pst_arrow;
      Dlg_pstruct.m_radio_hinge=m_pst_hinge;
      Dlg_pstruct.m_radio_color=m_pst_color;
      Dlg_pstruct.m_radio_dis=m_pst_dis;
      Dlg_pstruct.m_edit_mlimit=m_mlimit;
      Dlg_pstruct.m_edit_mass=m_pst_mass;
      Dlg_pstruct.m_edit_p1=m_pst_ef1;
      Dlg_pstruct.m_edit_p2=m_pst_ef2;
```

```
      Dlg_pstruct.m_edit_x1=m_base[0];
      Dlg_pstruct.m_edit_x2=m_base[1];
      Dlg_pstruct.m_edit_x3=m_base[2];
      UpdateData(FALSE);
//    ―――――――――――――――――――――
//      ●　モーダルダイアログ表示
//    ―――――――――――――――――――――
    if(Dlg_pstruct.DoModal() == IDOK){
      F_time_ii=0;
      UpdateData(TRUE);
      m_pst_disp = Dlg_pstruct.m_radio_displacement;
      m_pst_hinge = Dlg_pstruct.m_radio_hinge;
      m_pst_color = Dlg_pstruct.m_radio_color;
      m_pst_dis = Dlg_pstruct.m_radio_dis;
      m_pst_graph=Dlg_pstruct.m_radio_graph;
      m_pst_arrow=Dlg_pstruct.m_radio_arrow;
      m_pst_mass=Dlg_pstruct.m_edit_mass;
      m_pst_ef1=Dlg_pstruct.m_edit_p1;
      m_pst_ef2=Dlg_pstruct.m_edit_p2;
      m_mlimit=Dlg_pstruct.m_edit_mlimit;
      m_base[0] = 0.;
      m_base[1] = 0.;
      m_base[2] = 0.;
//    ―――――――――――――――――――――
//      ●　描画オプションの設定
//    ―――――――――――――――――――――
      m_pst_options = 0;
      if(nelem < 1001){
        if(Dlg_pstruct.m_check_mem_used){
          mm_opt[0]=Dlg_pstruct.m_edit_mem11;
          mm_opt[1]=Dlg_pstruct.m_edit_mem22;
          mm_opt[2]=Dlg_pstruct.m_edit_mem33;
          mm_opt[3]=Dlg_pstruct.m_edit_mem44;
          mm_opt[4]=Dlg_pstruct.m_edit_mem55;
          mm_opt[5]=Dlg_pstruct.m_edit_mem66;
          mm_opt[6]=Dlg_pstruct.m_edit_mem77;
          mm_opt[7]=Dlg_pstruct.m_edit_mem88;
          mm_opt[8]=Dlg_pstruct.m_edit_mem99;
          mm_opt[9]=Dlg_pstruct.m_edit_mem10;
          m_pst_options = lin_options();
         }}
//    ―――――――――――――――――――――
//      ●　描画中心の設定
//    ―――――――――――――――――――――
      int ii;
      if(Dlg_pstruct.m_radio_x1 == 1){
        ii = 1;
        AVERAG(&ii,&m_base[0],&node,posit);
      }
      if(Dlg_pstruct.m_radio_x1 == 2){
        m_base[0] = Dlg_pstruct.m_edit_x1;
      }
      if(Dlg_pstruct.m_radio_x2 == 1){
        ii = 2;
```

```
          AVERAG(&ii,&m_base[1],&node,posit);
        }
        if(Dlg_pstruct.m_radio_x2 == 2){
          m_base[1] = Dlg_pstruct.m_edit_x2;
        }
        if(Dlg_pstruct.m_radio_x3 == 1){
          ii = 3;
          AVERAG(&ii,&m_base[2],&node,posit);
        }
        if(Dlg_pstruct.m_radio_x3 == 2){
          m_base[2] = Dlg_pstruct.m_edit_x3;
        }
        UpdateData(FALSE);
//    ――――――――――――――――――――――
//      ●　集中質量の描画設定と動的領域確保
//    ――――――――――――――――――――――
    if(m_pst_mass > 0){
      if(F_read_mass == 0){
        F_mass =   new int[node*2];
        if(F_mass == 0){
          m_pst_mass = 0;
          MessageBox("Sorry! Memory allocation failure. You should stop this application.");
        }else{
          int ierr;
          INPMAS(&node,F_mass,&ierr);          // 集中データ入力
          if(ierr == 0){
            F_read_mass = 1;
          }else{
            MessageBox(" File of mass could not open.");
            m_pst_mass = 0;
          }
     }}
    }
//    ――――――――――――――――――――――
//      ●　速度場の動的領域確保とデータ入力
//    ――――――――――――――――――――――
    if(m_pst_dis == 1 && F_read_vel == 0){
       F_vel = new float[3*node*F_all_step];
       if(F_vel == 0){
         MessageBox("Sorry! Memory allocation failure.");
         return;
       }
       int ier=input_vel();          // 速度場データ入力
       if(ier == 1) {
          MessageBox(" Sorry! System did not open this file.");
          F_read_vel = 0;
          m_pst_dis =0;
      }else{
          F_read_vel = 1;
      }
    }
//    ――――――――――――――――――――――
//      ●　相対加速度場の動的領域確保とデータ入力
//    ――――――――――――――――――――――
```

```
    if(m_pst_dis == 2 && F_read_acc == 0){
      F_acc = new float[3*node*F_all_step];
      if(F_acc == 0){
        MessageBox("Sorry! Memory allocation failure.");
        return;
      }
      int ier=input_acc();          // 相対加速度場データ入力
      if(ier == 1) {
        MessageBox(" Sorry! System did not open this file.");
        F_read_acc = 0;
        m_pst_dis =0;
      }else{
        F_read_acc = 1;
      }
    }
//    ――――――――――――――――――――――
//     ●  絶対加速度場の動的領域確保とデータ入力
//    ――――――――――――――――――――――
    if(m_pst_dis == 3 && F_read_accab == 0){
      F_accab = new float[3*node*F_all_step];
      if(F_accab == 0){
        MessageBox("Sorry! Memory allocation failure.");
        return;
      }
      int ier=input_accab();          // 絶対加速度場データ入力
      if(ier == 1) {
        MessageBox(" Sorry! System did not open this file.");
        F_read_accab = 0;
        m_pst_dis =0;
      }else{
        F_read_accab = 1;
      }
    }
//    ――――――――――――――――――――――
//     ●  節点反力・外力の動的領域確保とデータ入力
//    ――――――――――――――――――――――
    if(m_pst_arrow == 1 && F_read_ndbalanceF == 0){
      F_ndbalanceF = new float[3*node*F_all_step];
      if(F_ndbalanceF == 0){
        MessageBox("Sorry! Memory allocation failure.");
        return;
      }
      int ier=input_ndbal();          // 節点反力・外力データ入力
      if(ier == 1) {
        MessageBox(" Sorry! System did not open this file.");
        F_read_ndbalanceF = 0;
        m_pst_dis =0;
      }else{
        F_read_ndbalanceF = 1;
      }
    }
//    ――――――――――――――――――――――
//     ●  不釣合力の動的領域確保とデータ入力
//    ――――――――――――――――――――――
```

```
    if(m_pst_arrow == 2 && F_read_unbalanceF == 0){
      F_unbalanceF = new float[3*node*F_all_step];
      if(F_unbalanceF == 0){
        MessageBox("Sorry! Memory allocation failure.");
        return;
      }
      int ier=input_unbal();         // 不釣合力データ入力
      if(ier == 1) {
        MessageBox(" Sorry! System did not open this file.");
        F_read_unbalanceF = 0;
        m_pst_dis =0;
      }else{
        F_read_unbalanceF = 1;
      }
    }
//    ―――――――――――――――――――――
//     ●　各応力の動的領域確保とデータ入力
//    ―――――――――――――――――――――
    if((m_pst_hinge != 0 || m_pst_color!=0 || m_pst_graph != 0)&& F_read_spring == 0){
      F_fay = new float[5*mnbsb*m_nstep];
      if(F_fay == 0){
        MessageBox("Sorry! Memory allocation failure.");
        return;
      }
      F_n_spring= new float[5*mnbsb*m_nstep];
      if(F_n_spring == 0){
        MessageBox("Sorry! Memory allocation failure.");
        return;
      }
      F_my_spring= new float[5*mnbsb*m_nstep];
      if(F_my_spring == 0){
        MessageBox("Sorry! Memory allocation failure.");
        return;
      }
      F_mz_spring = new float[5*mnbsb*m_nstep];
      if(F_mz_spring == 0){
        MessageBox("Sorry! Memory allocation failure.");
        return;
      }
      F_stat_spring = new int[5*mnbsb*m_nstep];
      if(F_stat_spring == 0){
        MessageBox("Sorry! Memory allocation failure.");
        return;
      }
      int ier;
      ier=input_spring();         // 応力データ入力
      if(ier == 1) {
        MessageBox(" Sorry! System did not open this file.");
        m_pst_hinge = 0;
        F_read_spring = 0;
        return;
      }
      F_read_spring = 1;
    }
```

```
//    ―――――――――――――――――――――
//      ●　図形の再描画
//    ―――――――――――――――――――――
        CRect rcFrame;
        GetClientRect(&rcFrame);
        InvalidateRect( rcFrame ,TRUE);
      }
    }
}
```

このプログラムの内容を説明する。多少プログラムは長いが、内容はさほど難しくない。ダイアログによるパラメータの取得、描画オプションの設定、動的領域の設定、再描画である。それでは、少し詳しく見ていこう。

最初は、ウインドウ画面番号とウインドウコードを取得する。次に、このウインドウコードが構造用であれば、ダイアログに関するCSf31View クラスのメンバー変数から CDig_pstruct クラスのメンバー変数にデータをコピーする。次に、関数 Dlg_pstruct.DoModal()を用いてダイアログを表示する。ユーザーがデータを設定した後、OK ボタンを押すと、今度は逆に、CDig_pstruct クラスのメンバー変数からCSf3stView クラスのメンバー変数に値をコピーする。この時、関数UpdateData(FALSE)を実行することで、関数 DoDataExchange()がコールされ、ダイアログ表示変数に値がセットされる。

まず、描画の出力オプションを指定する場合は、そのデータを配列mm_opt[]にセットする。また、この配列を使用して、関数 lin_options()をコールし、その関数値を出力オプション m_pst_options にセットする。この関数はデータ入力仕様にしたがって、描画する部材を選択する。この関数の詳細はマニュアルプレゼンターを参照されたい。次に、描画における構造物の中心位置を設定する。この場合の選択番号として、1 はその方向の構造物の中心を描画中心に、また、2 は入力したデータを中心にセットする。これを３方向について行う。

次に、構造透視図を描くとき、オプションとして、集中質量の表示する場合で、しかも一度もこの質量保存領域を動的に確保していない場合、そのデータを保存する領域を、new 文を用いて動的に確保する。動的に領域確保ができた場合、F_read_mass = 1 に設定し、質量が存在する節点位置を知るために、INPMAS()サブルーチンを用いて質量データを読み込む。確保できない場合は、エラー表示してこの関数から抜ける。

以後、速度場、相対加速度、絶対加速度、及び反力・外力、不釣合力について、上記と同様にユーザーの要求に従ってファイルからデータを

入力する。さらに、構造透視図を描くとき、オプションとして、塑性ヒンジ、線でカラー軸力表示、曲げモーメント表示を選択し、しかも、一度もこれらの動的領域を確保していない場合、次の動的領域を確保する。無論、動的領域の確保に失敗するとエラー表示を行い、この関数を抜ける。

以上で、この関数の手続きは終わり、最後に、関数 InvalidateRect()を用いて、このウインドウを再描画する。この関数が実行されると、指定しているウインドウ領域が無効となり、関数 OnDraw()が実行されて再描画され、このウインドウが更新されることになる。

5.3 プログレスバーの設定法

本節では、プログレスバーの設定法について学ぶ。このプログレスバーは、長時間のデータ入力が必要となる際、ユーザーにその状況を知らせるツールである。このプログレスバーは、SPACE 全般で、また他のプログラムでも多用されており、その表示法は GUI における基本的なテクニックと言えよう。

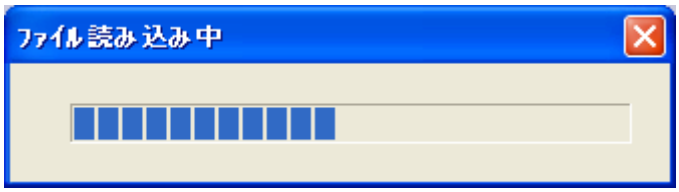

図 5-4 プログレスバー

データ入力状況を知らせるプログレスバー（図 5-4 参照）は、通常のファイル入力コード以外に、この図形を表示するためのダイアログクラスが必要となる。このクラス名を CprogressDlg とし、ダイアログを図 5-4 のように設計する。

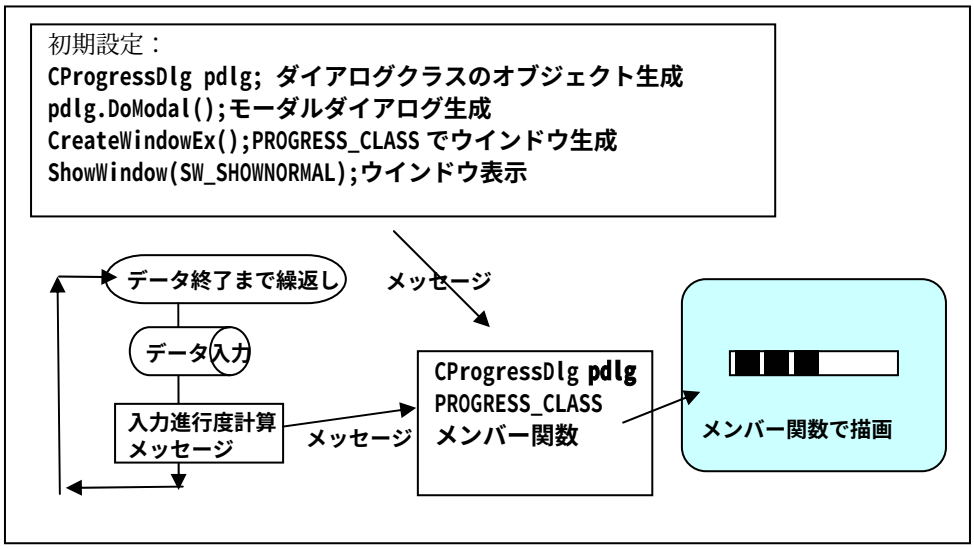

図 5-5 プログレスバーのメカニズム

プログレスバーを利用するためには、通常の入力処理に加えて、幾つかの準備が必要である。ダイアログクラス CprogressDlg を作成し、そのオブジェクトに各種のメッセージを送って、プログレスバーの表示などこのオブジェクトを制御する。まず、このオブジェクトを初期設定し、次にデータ入力途中で入力進行状況をメッセージで知らせる処理を行わなければならない。従って、その種の仕組みを入力処理に組み込まなくてはならない。特に、SPACE では多くの入力処理は FORTRAN で行っているため、さらに面倒な手続きが必要となる。以下に具体的な例を用いて練習しよう。

プレゼンターにおける節点変位入力を例にして、このプログレスバーの設定方法を解説する。最初に、節点変位を入力するための関数を観察する。この関数 OnSwndSt()は、プルダウンメニューの「解析画面」を選択することでコールされ、変位が呼び込まれることになる。以下にこの関数を示す。

```
//     ―――――――――――――――――――――――――――――――
//     ●    OnSwndSt:節点変位の入力
//     ―――――――――――――――――――――――――――――――
//
void CSf31View::OnSwndSt()
{
    HWND hwnd = this->GetSafeHwnd();
    if(IsWindow(hwnd) == FALSE) return;
    if(F_read_disp == 0){
        F_disp = new float[3*node*F_all_step];                          11
        if(F_disp == 0){
            MessageBox("Sorry! Memory allocation failure.");
            return;
         }
        m_pst_dis = 0;
        int ierr =0;
        int jj;
        int inputx =1;
         int all_step =F_all_step;
        INDISP(&inputx,F_disp,&ierr,&jj,&node,&all_step);              !2
        if(ierr == 1) {
           m_dat_struct=0;
           F_read_disp=0;
           m_nstep =0;
           MessageBox(" Sorry! System did not open this file.");
           return;
        }
        int p_input=0;
        CProgressDlg pdlg(p_input,all_step);                           !3
        if( pdlg.DoModal()== IDOK){                                    !4
          m_nstep=F_mnstep;                                            !5
        }else{                                                         !6
```

```
            m_dat_struct=0;                                          !7
            F_read_disp=0;
            m_nstep =0;
           return;
          }
      }else{
          m_nstep=F_mnstep;
      }
       m_dat_struct = 1;
       long fno_yx = GetWindowLong(hwnd,GWL_USERDATA);                !8
       int ii=STRUCT;
       long fno_zz =   setwindx(fno_yx,ii,hwnd);
       if(fno_yx != fno_zz) SetWindowLong(hwnd,GWL_USERDATA,fno_zz);
       ii=fno_zz-1;
       F_read_disp=1;
       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]);     !9
        int fno_xy = getwindx((long)fno_yx);
        int fno_xxx = 1;
        setwindy(fno_zz,fno_xxx);
        int idm =1;
        float dm;
        m_header = getheader(idm);                                   !10
        CString buffer =(CString) F_title;
        m_footer = getfooter(idm,idm, idm,idm, dm,dm,dm,buffer);
        CRect rcFrame;
        GetClientRect(&rcFrame);                                      !11
        InvalidateRect( rcFrame ,TRUE);                               !12
}
```

プログレスバーに関連する部分を中心に、コメント番号に従って説明する。

１．構造物節点の変位を保存する配列を動的に確保する。
２．FORTRAN のサブルーチンを用いて、節点変位をファイルから入力するために、まず当該ファイルをオープンする。
３．ダイアログクラス CprogressDlg のコンストラクタを用いて、オブジェクト pdlg を生成する。
４．このオブジェクトのモーダルダイアログを表示するために、関数 pdlg.DoModal()を実行する。この実行によって、プログレスバーが表示され、データ入力の状況が示される。データ入力が完了すると、IDOK となり、次のコードを実行する。
５．データ入力したステップ数をメンバー変数 m_nstep にセットし、先

に進む。

６．データ入力が途中でキャンセルされた場合は次のコードを実行する。

７．キャンセルされた場合は、制御変数を全て０にセットし、この関数から抜ける。

８．このウインドウの管理コードを取得する。

９．構造物のスケールや回転行列を計算し、構造物の初期透視図を描く準備を行う。次に、このウインドウの管理コードを、関数 setwindy() を使用してセットする。

10．ウインドウに表示されるヘッダーとフッターが、関数 getheader() と getfooter()で表される。

11. 関数 GetClientRect()によって、このウインドウの領域を取得する。

12. 関数 InvalidateRect()によって、この領域を無効にする。このことで、システムは、関数 OnDraw()を用いてその無効の領域を再描画する。このウインドウは既に構造透視図を描く設定となっているため、新たに図形が描かれることになる。

次に、実際にプログレスバーを進行させる際の手続きを見てみよう。最初に、このクラスのコンストラクタを示す。

```
//    ――――――――――――――――――――――――――――――――
//    ●    CProgressDlg:節点変位の入力
//    ――――――――――――――――――――――――――――――――
CProgressDlg::CProgressDlg(int pinput,int nstep, CWnd* pParent /*= NULL*/)
        : CDialog(CProgressDlg::IDD, pParent)
{
        aborted = 0;          ! 終了コードを示す
        m_nstep=nstep;        ! データ入力するステップ数
        m_pinput = pinput;    ! プログレスバーを制御するパラメータ、初期値は 0 である。
}
```

このコンストラクタでは、プログレスバーの進行状況を表示するために、上記３つのパラメータを初期設定する。

次に、メッセージマップと DoModal 関数によって最初に実行される関数 OnInitDialog()を以下に示す。

```
BEGIN_MESSAGE_MAP(CProgressDlg, CDialog)
        //{{AFX_MSG_MAP(CProgressDlg)
        //}}AFX_MSG_MAP
        ON_COMMAND(CM_DO_INPUT,Doinput)
        ON_COMMAND(CM_DO_INPUT1,Doinput1)                    !1
        ON_COMMAND(CM_DO_INPUT2,Doinput2)
```

```
        ON_COMMAND(CM_DO_INPUT3,Doinput3)
        ON_COMMAND(CM_DO_INPUT4,Doinput4)
        ON_COMMAND(CM_DO_INPUT5,Doinput5)
        ON_COMMAND(CM_DO_INPUT6,Doinput6)
        ON_COMMAND(CM_DO_INPUT7,Doinput7)
        ON_COMMAND(CM_DO_INPUT8,Doinput8)
END_MESSAGE_MAP()

//    ――――――――――――――――――――――――――――――――――
//    ●    OnInitDialog:DoModal()で最初に実行される関数
//    ――――――――――――――――――――――――――――――――――
BOOL CProgressDlg::OnInitDialog()
{
        CDialog::OnInitDialog();
        hWndPB = CreateWindowEx(0,PROGRESS_CLASS,NULL,WS_CHILD|WS_VISIBLE,
                30,20,280,20,m_hWnd,0,AfxGetInstanceHandle(),NULL);          !2
        ::SendMessage(hWndPB,PBM_SETRANGE,0,MAKELPARAM(0,100));              !3
        CenterWindow();                                                      !4
        ShowWindow(SW_SHOWNORMAL);                                           !5
        if(m_pinput == 1) PostMessage(WM_COMMAND,CM_DO_INPUT);
        if(m_pinput == 0) PostMessage(WM_COMMAND,CM_DO_INPUT1);              !6
        if(m_pinput == 2) PostMessage(WM_COMMAND,CM_DO_INPUT2);
        if(m_pinput == 3) PostMessage(WM_COMMAND,CM_DO_INPUT3);
        if(m_pinput == 4) PostMessage(WM_COMMAND,CM_DO_INPUT4);
        if(m_pinput == 5) PostMessage(WM_COMMAND,CM_DO_INPUT5);
        if(m_pinput == 6) PostMessage(WM_COMMAND,CM_DO_INPUT6);
        if(m_pinput == 7) PostMessage(WM_COMMAND,CM_DO_INPUT7);
        if(m_pinput == 8) PostMessage(WM_COMMAND,CM_DO_INPUT8);
        return TRUE;  // コントロールにフォーカスを設定しないとき、戻り値は TRUE となります
                      // 例外: OCX プロパティ ページの戻り値は FALSE となります
}
void CProgressDlg::OnCancel()
{
        aborted = 1;                                                         !7
        CDialog::OnCancel();
}
```

上記のプログラムをコメント番号に従って説明する。

１．メッセージマップの中で、メッセージ CM_DO_INPUT1 によって、関数 Doinput1()をコールする。この関数は、節点変位の読み込み処理を行う。

２．プログレスダイアログバーを表示するために、PROGRESS_CLASS のウインドウを生成する。

３．進行状況を知らせるバーの範囲を設定するために、PBM_SETRANGE のメッセージを送る。ここで、MAKELPARAM(0,100)を使用して、範囲を 0 から 100 に設定する。

４．関数 CenterWindow()で、このプログレスダイアログバー用ウインド

ウを画面の中心に置く。
５．関数 ShowWindow(SW_SHOWNORMAL)で、このウインドウを表示する。
６．パラメータ m_pinput が０であることから、メッセージ CM_DO_INPUT1 を送る。
７．データ入力が途中でキャンセルされた場合は、この関数に制御が移り、ここでは制御変数 aborted を１にセットする。

次に、実際にデータを入力し、プログレスバーを進行させる関数を示す。この関数 Doinput1()は、多くのメッセージ送受信関数を使用しており、ここでは、特に重要な２つの関数の役割をまとめる。

SendMessage(hWndPB,PBM_SETPOS,100,0)；ウインドウにメッセージを送り、処理が終了するまで待つ。
PostMessage(WM_COMMAND,CM_DO_INPUT1)；ウインドウにメッセージを送り、処理が終了するまで待たないので、結果を受け取ることができない。

次に、プログレスバーを進行させる関数を示す。

```
void CProgressDlg::Doinput1()
{
        long top_byte = m_nstep;                                    !1
        int ier=0;
        int jj=0;
        ier=1;
        int inputx =1;                                              !2
        for(jj=0;jj<m_nstep;jj++){                                  !3
          INDISP(&inputx,F_disp,&ier,&jj,&node,&m_nstep);           !4
          if(ier == -1 || aborted == 1){                            !5
            F_mnstep=jj;
            ier=-1;
            INDISP(&inputx,F_disp,&ier,&jj,&node,&m_nstep);         !6
            ::SendMessage(hWndPB,PBM_SETPOS,100,0);                 !7
            CDialog::OnOK();                                        !8
            return;
          }
          int pct = (int)(100.0 * (double) jj/(double) m_nstep);    !9
          ::SendMessage(hWndPB,PBM_SETPOS,pct,0);                   !10
          MSG msg;                                                  !11
          if( PeekMessage((LPMSG)&msg,(HWND)NULL,(WORD)NULL,(WORD)NULL,TRUE));  !12
          {
            TranslateMessage((LPMSG)&msg);                          !13
            DispatchMessage((LPMSG)&msg);                           !14
          }
        }
        ier=-1;
        F_mnstep=m_nstep;                                           !15
        INDISP(&inputx,F_disp,&ier,&jj,&node,&m_nstep);             !16
```

PeekMessage と GetMessage は、どちらもメッセージを受け取りにいくが、前者は、キューにメッセージがない場合、待たずに戻る点が異なる。

```
        ::SendMessage(hWndPB,PBM_SETPOS,100,0);                    !17
        CDialog::OnOK();                                           !18
}
```

上記のプログラムをコメント番号に従って説明する。この中で、進行状況を示すパラメータ pct を SendMessage(hWndPB,PBM_SETPOS,pct,0)で、プログレスバークラスのハンドル hWndPB を用いて、メッセージを送信し、進行状況を表示させている。

1. 初期設定を行う。最初のステップでファイルをオープンするために、パラメータ ier を１にセットする。
2. ファイルの選択パラメータ inputx を１にセットする。
3. 最大ステップ m_nstep 分、ループ処理を行う。
4. サブルーチン INDISP で、ファイルから１ステップ分の節点変位を読み込む。ただし、最初のコールでは、ファイルのオープン処理を行う。
5. ファイルのオープン時、もしくは、途中でデータが終了した場合、次の処理を行い、この関数から抜ける。
6. 途中終了の場合であり、ファイルのクローズ処理を行うため、サブルーチン INDISP()をコールする。
7. プログレスバーの進行状況を 100 にするようにメッセージを送る。
8. 関数 CDialog::OnOK()を用いて、このダイアログの OK 処理を行う。
9. 入力状況を計算する。入力ステップ数の割合をパーセント pct で求める。
10. 入力状況 pct をセットし、関数 SendMessage()を用いてプログレスバーを進行させる。
11. MSG msg を生成する。
12. データ入力実行中メッセージキューにメッセージが入っている場合、つまり、何らかのユーザーからのメッセージがあった場合、このメッセージを msg に取り出し、13,14 の処理を行う。
13. 関数 TranslateMessage((LPMSG)&msg)を用いて、メッセージを詳細なメッセージに変換する。
14. 関数 DispatchMessage((LPMSG)&msg)を用いて、ウインドウプロシジャであるコールバック関数へ返信する。
15. 以上で、データ入力ループは終了であり、後は終了処理を実行する。
16. ファイルのクローズ処理を行うため、サブルーチン INDISP()をコールする。

17. 関数 SendMessage()を用いて、入力状況 100 をセットしてプログレスバーを終了させる。
18. 関数 CDialog::OnOK()を用いて、このダイアログの OK 処理を行う。

次に、節点変位を１ステップごとに入力するサブルーチンを示す。この内容は単純であるため、コメントを参照することで理解することは容易である。

```
C     ―――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /INDISP
C     ―――――――――――――――――――――――――――――――
C     ●    ファイルから１ステップづつデータを入力する
C     ―――――――――――――――――――――――――――――――
      subroutine INDISP(intptx,disp,ierr,iss,node,nstep)
      common /sf01/fnfile,jdfile,kdfile,iidat,title,lengf,ltitle,timex
      character fnfile(100)*50,title*50,timex*20(100)
      integer*4 jdfile(100),kdfile(100),lengf(100)
      real*4 disp(3,node,nstep)
C     ―――――――――――――――――  パラメータ ierr が０の場合は初期設定
      if(ierr .ne.0) goto 9999
      ierr=0
C     ―――――――――――――――――  入力するファイル番号をセットする
         nfl=38
         if(intptx.eq.2) nfl=46
         if(intptx.eq.3) nfl=45
         if(intptx.eq.4) nfl=48
         if(intptx.eq.5) nfl=21
      if(jdfile(NFL).ne.1)then
         ierr=1
         return
      endif
C     ―――――――――――――――――  ファイルをオープンする
      open(unit=20,file=fnfile(NFL),FORM='UNFORMATTED', status='old',err=199)
C     ―――――――――――――――――  データ保存配列のゼロクリアを行う。
      do 210 i=1,nstep
      do 210 j=1,node
      do 210 k=1,3
      disp(k,j,i)=0.
 210  continue
      return
C     ―――――――――――――――――  パラメータ ierr が-1 の場合は終了処理を行う
9999  continue
      if(ierr.eq.-1) goto 9998
C     ―――――――――――――――――  １ステップ進めて、全節点について変位を読み込む
      is=iss+1
      nn=0
      do 10 j=1,node
      read(20,end=198)(disp(k,j,is),k=1,3)
 10   continue
      return
C     ―――――――――――――――――  終了処理、ファイルをクローズする
```

```
9998  continue
      close(20)
      return
C     ―――――――――――――――――  ファイルのオープン時のエラー
 199   continue
      ierr=1
      return
C     ―――――――――――――――――  データ途中でファイルが終了した場合の処理
 198  continue
      ierr=-1
      close(20)
      nstep = iss
      return
      end
```

5.4 ポップアップメニュー作成法

本節では、これもウインドウ用プログラムでは必須のテクニックであるポップアップメニューの作成法について解説する。

動的プレゼンターでは、マウス右ボタンの押下によってポップアップメニューが表示されるが、状況によってはマウス操作が他の機能と競合する場合もある。まず、右ボタンダウン処理を行う関数 OnRButtonDown() を観察してみよう。

```
//    ―――――――――――――――――――――――――――――――――
//    ●    OnRButtonDown ボタンダウン処理
//    ―――――――――――――――――――――――――――――――――
//
void CSf31View::OnRButtonDown(UINT nFlags, CPoint point)
{
         F_hantei=2;
         HWND hwnd = this->GetSafeHwnd();
         CDC* pDC = GetDC();
         if(IsWindow(hwnd) == FALSE) return;
         int  fno_yx = GetWindowLong(hwnd,GWL_USERDATA);
         int fno_xy = getwindx(fno_yx);
         fno_xx = (int) fno_yx;

         if(nFlags & MK_SHIFT) F_hantei =0;
         if(nFlags & MK_CONTROL) F_hantei =1;
         if(F_hantei == 0 && On_mem_option == 1) F_hantei = 3;
         if(F_hantei == 0 && On_mem_option == 2) F_hantei = 4;
         if(F_hantei == 0 && On_mem_option == 3) F_hantei = 8;
         if(F_hantei == 1 && On_mem_option == 1) F_hantei = 5;
         if(F_hantei == 1 && On_mem_option == 2) F_hantei = 6;
         if(F_hantei == 1 && On_mem_option == 3) F_hantei = 9;
         if(F_hantei == 2 && On_mem_option != 0) F_hantei = 7;

switch (F_hantei){
case 0:
```

```
        if(fno_xy != 1 ) return;
        pikpoint(point,F_hantei);
        break;
    case 1:
          .
          .
    //    ―――――――――――――――――――――
    //      ●  ポップアップメニュー
    //    ―――――――――――――――――――――
    case 2:
            CMenu PopMenu;
            HMENU hPopMenu = HMENU(PopMenu);
            CMenu SubMenu;
            HMENU hSubMenu = HMENU(SubMenu);
            VERIFY(SubMenu.LoadMenu(IDR_MENU1));
            CMenu* pPopup = SubMenu.GetSubMenu(0);
            ASSERT(pPopup != NULL);
            CPoint ClientPoin = point;
            ClientToScreen(&ClientPoin);
            pPopup->TrackPopupMenu(TPM_LEFTALIGN | TPM_RIGHTBUTTON,
                                   ClientPoin.x,ClientPoin.y, GetParent());
            pPopup->DestroyMenu();
            break;
    }
            ReleaseDC (pDC);
            CView::OnRButtonDown(nFlags, point);
    }
```

右ボタン押下による動作の仕様は、現在でも７種類存在する。ここでは、メンバー変数 On_mem_option の値とキィの同時押下との組み合わせで、その動作を決定している。動作はパラメータ F_hantei の値で決められ、ポップアップメニューは２に設定されている。

それでは、この関数 OnRButtonDown()を見ていこう。まず、このウインドウのハンドル hwnd を取得し、関数 IsWindow(hwnd)を用いて、そのウインドウが表示されていることを確かめる。取得したハンドルからウインドウ画面番号とウインドウコードを得る。

次に、F_hantei コードをキィにして、switch 文で動作処理を決める。ポップアップメニュー（ショートカットメニュー）は、ケース２であり、ここ以降の 12 行のコードは、ポップアップメニューを表示するための標準的な手続きである。次にこの手続きの概略を説明しよう。

まず、メニュークラス PopMenu を生成し、そのハンドルを得る。同様にメニュークラス Submenu を生成し、そのハンドルを得る。リソースのメニューIDである IDR_MENU1で定義されているメニューを Submenu に呼び込む。このメニューのサブメニューを関数 SubMenu.GetSubMenu(0)で取り出し、pPopup に割り付ける。さらに、マウスの位置 point を

ClientPoint にセットし、関数 pPopup->TrackPopupMenu()で、その位置にポップアップメニューが表示されることになる。メニューの中でユーザーの選択が終了すると、該当するメニューの ID を持ったメッセージが発信される。この関数から戻った後、メニュークラスが消去される。このポップアップメニューは、図 5-6 に示される。

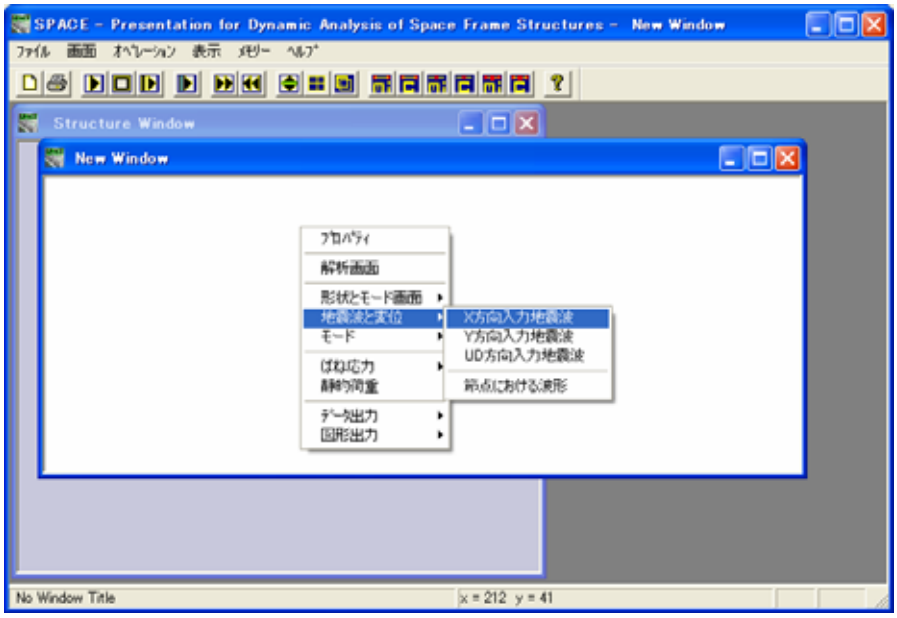

図 5-6 ポップアップメニューの表示

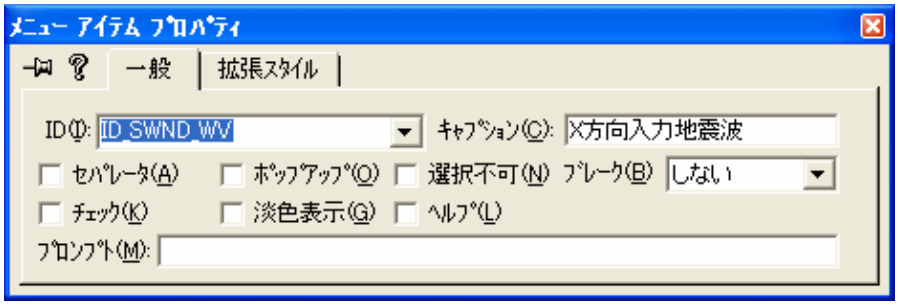

図 5-7 ポップアップメニューのアイテムの割付

次に、このポップアップメニューで呼び出される関数を見てみよう。ここでは、地震波履歴の表示を例に取り上げる。ポップアップメニューのひとつを、x 方向の地震波履歴を描画するメニューとする。図 5-7 に示すようにメニューのプロパティで ID を ID_SWND_WV とすると、このメッセージによって、メッセージマップに示されるように関数 OnSwndWv()がコールされる。

```
//    ――――――――――――――――――――――――――――――――
//    ●　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_SWND_WV, OnSwndWv)
        ON_COMMAND(ID_SWND_SS, OnSwndSs)
        ・
        ・
END_MESSAGE_MAP()
```

ｘ方向の地震波を表示する関数 OnSwndWv()では、まず予備設定を行う。ここでは、関数 wave_data_set()の引数として１を与え、ｘ方向の地震波としている。関数 wave_data_set(int nwave)では、サブルーチン INUGG()を用いて、この解析で地震波を設定しているかどうかチェックする。もし、設定してない場合は、エラーメッセージを表示して関数を抜ける。設定している場合は、ウインドウコードを WAVE にセットした後、ウインドウ管理システムに登録する。次は、実際に sf31wave.Sf31set 関数で地震波を入力し、そのデータを配列にセットする。

このウインドウのハンドルを取得した後、sf31wave.set_wave_frame 関数を用いて、図 5-9 に示す地震波の履歴を描画する。この２つの関数については、次節で解説する。

```
//      ―――――――――――――――――――――――――――――――
//      ●     ｘ方向地震波表示の予備設定
//      ―――――――――――――――――――――――――――――――
//
void CSf31View::OnSwndWv()
{
        int ii =1;
        wave_data_set(ii);
}
//    ―――――――――――――――――――――――――――――――
//      ●    地震波と節点履歴表示
//    ―――――――――――――――――――――――――――――――
//      ―――――――――――――――――――――――――――――――
//      ●     地震波表示
//      ―――――――――――――――――――――――――――――――
//
void CSf31View::wave_data_set(int nwave)
{
        HWND hwnd = this->GetSafeHwnd();
        if(IsWindow(hwnd) == FALSE) return;
        int ierr,nug;
        char filenx[82] ;
```

```
C     ―――――――――――――――――― 地震波形の使用状況チェック
        INUGG(&nwave,&nug,&F_delugg,&F_igra[0],&ierr,filenx);
        if(ierr != 0 ) {
          if(nwave == 1 )
            MessageBox("Sorry! System cannot open this X direction earthquake file.");
          if(nwave == 2 )
            MessageBox("Sorry! System cannot open this Y direction earthquake file.");
          if(nwave == 3 )
            MessageBox("Sorry! System cannot open this UD direction earthquake file.");
         return;
        }
C     ―――――――――――――――――― ウインドウ管理システムに画面状況を登録
      long fno_yx = GetWindowLong(hwnd,GWL_USERDATA);
      int ii = WAVE;
      m_dat_struct = 0;
      long fno_zz =   setwindx(fno_yx,ii,hwnd);
      if(fno_yx != fno_zz) SetWindowLong(hwnd,GWL_USERDATA,fno_zz);
C     ―――――――――――――――――― 地震波形入力と描画
      int nuff = F_f1sec/F_delugg;
      int nugg = nug +nuff;
      sf31wave.Sf31set(nugg,nwave,nuff,F_xgal[nwave-1]);
      CDC* pDC= GetDC();
C     ―――――――――――――――――― 画面にヘッダーとフッター描画
      int idm = 4+nwave;
      float dm;
      m_header = getheader(idm);
      CString buffer = filenx;
      m_footer = getfooter(idm,idm, idm,idm, dm,dm,dm,buffer);
C     ―――――――――――――――――― 地震波形を描画
      HWND hWnd=WindowFromDC(pDC->m_hDC);
      sf31wave.set_wave_frame(pDC,hWnd);
      ReleaseDC (pDC);
}
```

5.5 地震波描画と進行状況表示

本節では、地震波の履歴図（図 5-9）を描く処理について解説する。このグラフ表示処理は、第２章で説明した荷重履歴図とほぼ同じであり、詳細な説明は省くことにする。

アニメーションを実行するとき、構造図の挙動にシンクロして地震波の履歴図内を赤い点が移動する。この動きを処理する仕組みについて、ここで解説する。SPACE では、この処理は多くの箇所で使用されており、非常に有用なテクニックと言えよう。ここでは、先に説明したラバーバンドを用いた技術を応用しており、理解することは容易であろう。

まず、地震波を入力し、その履歴図を描画する関数を示す。地震波のファイルからの入力は、関数 CSf31wave::Sf31set()で行う。次に、関数 CSf31wave::set_wave_frame()と CSf31wave::set_wave_disp()で波形の

アニメーションのループ処理の中で、グラフの進行状況を次の設定で表示する。

２回四角を排他的 OR で描画する。
１回目は、現在の時刻で四角を描く（実線）。
２回目は、１ステップ前の時刻を使用して、四角を同じく排他的 OR で描画する。
これで、前ステップの四角表示が消え、しかも元の図形が復活することになる。

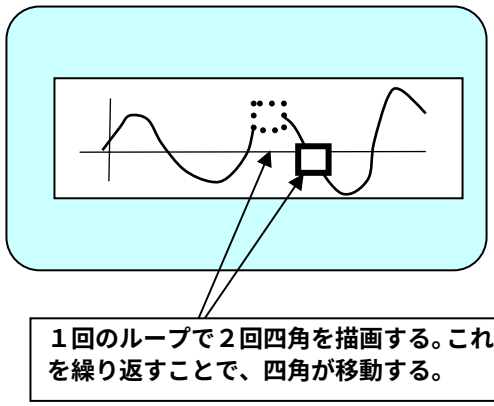

図 5-8 グラフ進行状況の表示

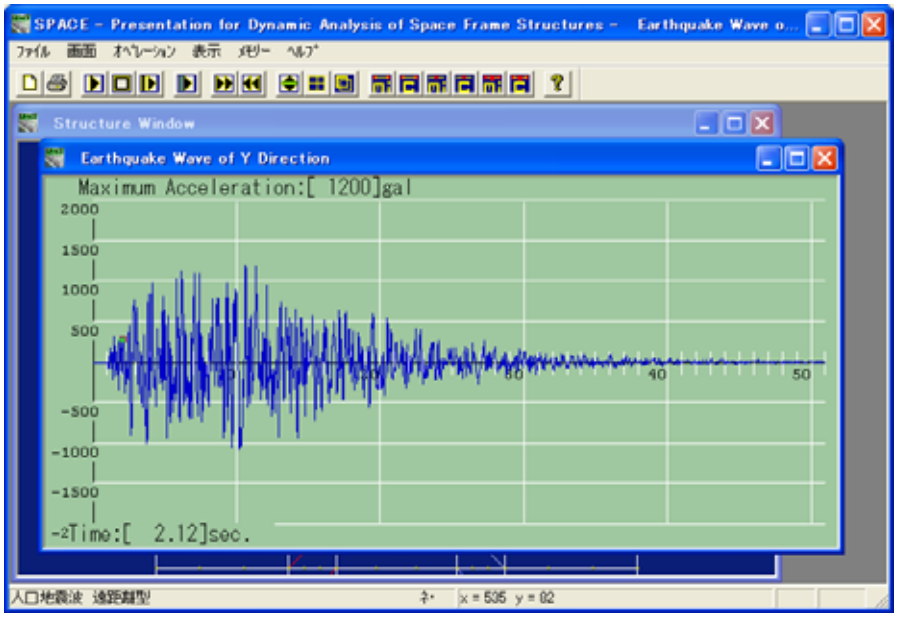

図 5-9 地震波の履歴図

描画を行う。地震波形を描画する関数を以下に示す。これら３つの関数は、既に説明した関数とほぼ同じであり、コメント行を参考にすれば理解することは容易である。

```
//      ―――――――――――――――――――――――――――――――――
//      ●     Sf31set:地震波形の入力
//      ―――――――――――――――――――――――――――――――――
void CSf31wave::Sf31set(int isize,int nwave,int nuff,float m_wave_data_maxx )
{
//      ―――――――――――――――――――  地震波をデータ保存する配列の動的確保
          if(m_data_in == TRUE){
              delete [] FF_wave;
          }
          m_wave_max = isize;
          int size = isize ;
```

```
          FF_wave = new float[size];
          if(FF_wave == 0){
              MessageBox("Sorry! Memory allocation failure.");
              return;
          }
//        ――――――――――――――――――――  ファイルからの地震波の入力
          int ierr;
          INUG(&ierr,&nwave,FF_wave,&m_wave_max,&nuff,&m_wave_delt,&m_ierr);
          if(ierr != 0 ) return;
//        ――――――――――――――――――――  各種の描画用初期設定
          m_data_in = TRUE;
          m_type = 0;
          m_nnode=nwave;
          m_freed=-1;
          m_pst_dis=0;
          m_wave_delt_v = 1./m_wave_delt;
          m_wave_max_v = 1./(float)m_wave_max;
          m_wave_data_max = m_wave_data_maxx;
          m_wave_time = (float)m_wave_max * m_wave_delt;
}
//        ――――――――――――――――――――――――――――――――
//        ●   set_wave_frame:地震波の描画の仕組み
//        ――――――――――――――――――――――――――――――――
BOOL CSf31wave::set_wave_frame(CDC* pDC,HWND hWnd)
{
//        ――――――――――――――――――――  メモリー領域を動的確保する
          BOOL hantei = FALSE;
          CDC* pMemDC = new CDC;
          ::GetClientRect(hWnd ,&m_Wave_rect);
          pMemDC->CreateCompatibleDC(pDC);
          CBitmap *pMemBitmap  = new CBitmap;
          pMemBitmap -> CreateCompatibleBitmap(pDC,m_Wave_rect.right,m_Wave_rect.bottom);
          CBitmap *poldBitmap =(CBitmap*)pMemDC -> SelectObject(pMemBitmap);
//        ――――――――――――――――――――  メモリー領域を背景で塗りつぶす
          disp_frame_wave(pMemDC, -1.);
//        ――――――――――――――――――――  地震波を描画する
          hantei= set_wave_disp(pMemDC);
//        ――――――――――――――――――――  VRAM に高速転送する
          pDC->BitBlt(0,0,m_Wave_rect.right,m_Wave_rect.bottom,pMemDC,0,0,SRCCOPY);
//        ――――――――――――――――――――  オブジェクトを消去する
          delete pMemDC->SelectObject(poldBitmap);
          delete pMemDC;
          return (hantei);
}
//        ――――――――――――――――――――――――――――――――
//        ●   set_wave_disp：地震波形の描画
//        ――――――――――――――――――――――――――――――――
BOOL CSf31wave::set_wave_disp(CDC* pDC)
{
//        ――――――――――――――――――――  描画領域を設定
          m_data_set_ok = TRUE;
          CPen* pOriginalPen = pDC->SelectObject (&NewPenx_a[109]);
          ar_w = m_Wave_rect.right - m_Wave_rect.left - 50;
          ar_h = (m_Wave_rect.bottom - m_Wave_rect.top - 40)/2;
```

```
        if(ar_w < 1) m_data_set_ok = FALSE;
        if(ar_h < 1) m_data_set_ok = FALSE;
        if( m_data_set_ok == FALSE) return ( m_data_set_ok);
        if(m_wave_max == 0 ) m_data_set_ok =FALSE;
        if( m_data_set_ok == FALSE) return ( m_data_set_ok);
//      ――――――――――――――――――――  ペンやフォントを作成
        CFont newFont;
        int point =16;
        newFont.CreateFont(point,0,0,0,FW_NORMAL,FALSE,FALSE,0,
                ANSI_CHARSET,
                OUT_DEFAULT_PRECIS,
                CLIP_DEFAULT_PRECIS,
                DEFAULT_QUALITY,
                DEFAULT_PITCH | FF_MODERN,
                "ＭＳ　ゴシック");
        CFont* pOldFont = (CFont*) pDC->SelectObject(&newFont);
                 int i1,i2,i3;
           i1= 0;
           i2= 0;
           i3= 0;
        COLORREF oldcoler = pDC->SetTextColor(RGB(i1,i2,i3));
           i1= 160;
           i2= 200;
           i3= 160;
        COLORREF oldbkcoler = pDC->SetBkColor(RGB(i1,i2,i3));
        if(m_type == 2) pDC->SetBkColor(RGB(160,200,220));
//      ――――――――――――――――――――  波形の原点を設定
        m_data_set_ok = TRUE;
        char buffer[20];
        int x_axi;
        float y_axi;
        float load_data_max,ld_dt;
        ar_sx = m_Wave_rect.left + 40;
        ar_sy = m_Wave_rect.top + ar_h + 20;
//      ――――――――――――――――――――  ｘ方向、ｙ方向の座標軸を描く
        pDC->MoveTo ( ar_sx, ar_sy);
        pDC->LineTo ( ar_sx + ar_w, ar_sy);
        pDC->MoveTo ( ar_sx, ar_sy + ar_h);
        pDC->LineTo ( ar_sx, ar_sy - ar_h);
//      ――――――――――――――――――――  ｘ方向の目盛増分値計算
        int ist, jst;
        ar_dt = (float) ar_w * m_wave_max_v * m_wave_delt_v;
        float ar_dtt = (float) ar_w * m_wave_max_v;
        ar_bi = (float)ar_h;
        jst = ar_sy;
        int op_time = (int) m_wave_time;
        int op_timenx =1;
        pDC->SelectObject ( &NewPenx_a[50]);
        if(m_wave_time > 13 && m_wave_time < 51) {
          op_timenx =5;
          op_time = (int) m_wave_time/op_timenx;
        }
        if(m_wave_time > 50 ) {
          op_timenx =10;
```

```
            op_time = (int) m_wave_time/op_timenx;
          }
//      ――――――――――――――――――――  ｘ軸の小目盛を描画
          if(op_timenx != 1 ){
             for ( int i=0;i < (int) m_wave_time; i++)
             {
                ist = ar_sx + ar_dt * ((float)(i+1));
                pDC->MoveTo ( ist, jst+10);
                pDC->LineTo ( ist, jst-10);
             }
          }
//      ――――――――――――――――――――  ｘ軸大目盛の描画と値の表示
          for (int i=0;i < op_time; i++)
          {
            ist = ar_sx + ar_dt * ((float)(i+1)*op_timenx);
            x_axi= (i+1)*op_timenx;
            sprintf(buffer,"%2d",x_axi);
            pDC->MoveTo ( ist, jst+ar_h);
            pDC->LineTo ( ist, jst-ar_h);
            pDC->TextOut(ist-16,jst+2,buffer);
           }
//      ――――――――――――――――――――  ｙ軸最大値と倍率計算
           SAIDAI(&m_wave_data_max,&load_data_max,&ld_dt);
           int iij =load_data_max/ld_dt+ 0.01;
           ar_bi = (float)ar_h /load_data_max;
//      ――――――――――――――――――――  ｙ軸目盛の描画と値の表示
          for ( i=0;i <iij; i++)
          {
             jst = ar_sy - ld_dt*ar_bi * (i+1);
             ist = ar_sy + ld_dt*ar_bi * (i+1);
             y_axi= (i+1)*ld_dt;
             if(ld_dt < 0.1) sprintf(buffer,"%3.2f",y_axi);
             if(ld_dt >= 0.1 && ld_dt < 1.) sprintf(buffer,"%3.1f",y_axi);
             if(ld_dt >= 1 && ld_dt < 100.) sprintf(buffer,"%3.0f",y_axi);
             if(ld_dt >= 100 ) sprintf(buffer,"%5.0f",y_axi);
             pDC->TextOut(ar_sx-35,jst,buffer);
             y_axi= -y_axi;
             if(ld_dt < 0.1) sprintf(buffer,"%3.2f",y_axi);
             if(ld_dt >= 0.1 && ld_dt < 1.) sprintf(buffer,"%3.1f",y_axi);
             if(ld_dt >= 1 && ld_dt < 100.) sprintf(buffer,"%3.0f",y_axi);
             if(ld_dt >= 100 ) sprintf(buffer,"%5.0f",y_axi);
             pDC->TextOut(ar_sx-35,ist,buffer);
             pDC->MoveTo ( ar_sx, jst);
             pDC->LineTo ( ar_sx+ar_w, jst);
             pDC->MoveTo ( ar_sx, ist);
             pDC->LineTo ( ar_sx+ar_w, ist);
          }
//      ――――――――――――――――――――  波形の倍率計算と図形初期設定
             ar_bi = (float)ar_h /load_data_max*m_wave_data_max;
             pDC->SelectObject ( &NewPenx_a[108]);
             int istt = ar_sx;
             int jstt = -FF_wave[0]*ar_bi+ ar_sy;
             pDC->MoveTo ( istt, jstt);
//      ――――――――――――――――――――  地震波形を描画
```

```
            for(  i = 1; i < m_wave_max-1; i++)
            {
                  ist = ar_sx + ar_dtt * (float)i;
                  jst = -FF_wave[i]*ar_bi + ar_sy;
                  pDC->LineTo ( ist, jst);
            }
//      ――――――――――――――――――――  リソースの解放
            m_time_b_tm = -1.;
            pDC->SelectObject ( pOriginalPen);
            pDC->SelectObject(pOldFont);
            pDC->SetTextColor( oldcoler);
            pDC->SetBkColor(oldbkcoler);
            return (m_data_set_ok);
}
```

次に、アニメーションにシンクロして赤い点が地震波履歴図内を移動する仕組みについて解説する。第４章で説明したアニメーションのメカニズムの中で、地震波における進行状況を表示する関数 set_wave_dat() がコールされている。この関数によって、現在の時刻が地震波形の上に赤い点として表示されることになる。まず、アニメーションのメカニズムで説明された関数 OnMessageWind()で、この進行状況に関連する箇所を以下に示す。

```
//      ――――――――――――――――――――――――――――――――
//      ●    OnMessageWind
//      ――――――――――――――――――――――――――――――――
//
 LONG CSf31View::OnMessageWind(UINT wParam, LONG lParam)
 {
    LONG d_han = 1;
    int fno_yx = lParam;
    int ii = fno_yx-1;
    HWND hWnd = this->GetSafeHwnd();
    if(checkhwnd(hWnd,ii) == FALSE) return (d_han);
    if(::IsIconic(hWnd)) return (d_han);
    CWnd* pWnd = CWnd::FromHandle(hWnd);
    CClientDC dc(pWnd);
    int fno_xy = getwindx((long)fno_yx);
    int fno_xxx = getwindy((long)fno_yx);
    if( F_operation == 2){
      int ihan =0;
      if(F_Time <= F_delt_cl) ihan=-1;
    switch (fno_xy){
    case STRUCT:
      if( fno_xxx == 0) pre_disp_mem_persp(hWnd,ii);
      if( fno_xxx == 1 && F_time_ii <= m_nstep) pre3_disp_mem_persp(hWnd,ii);
      if( fno_xxx == 2) pre4_disp_mem_persp(hWnd,ii);
      if( fno_xxx == 4 && F_time_ii <= m_nstep) pre5_disp_mem_persp(hWnd,ii);
      if( fno_xxx == 5 && F_time_ii <= m_nstep) pre6_disp_mem_persp(hWnd,ii);
```

```
          break;
      case WAVE:
        if(m_dat_struct == 0) sf31wave.set_wave_dat((CDC*)&dc, hWnd, F_Time,ihan);
        if(m_dat_struct == 2) sf31shear.set_wave_dat((CDC*)&dc, hWnd, F_Time,ihan);
        break;
      case SLOAD:
        •
        •
return(d_han);
  }
          d_han = 0;
return(d_han);
}
```

上の関数の中でコールされる関数 sf31wave.set_wave_dat()を示す。ここでは、多くの波形を同様な処理で行うため、そのタイプによって分類する。

```
//    ――――――――――――――――――――――――――――――――
//    ●　Sf31set_dis：各種の波形の移動処理のための分類
//    ――――――――――――――――――――――――――――――――
void CSf31wave::set_wave_dat(CDC* pDC,HWND hWnd,float ftime, int ihan)
{
        if(m_type == 0)set_wave_dat_y(pDC,hWnd,ftime, ihan);
        if(m_type == 1)set_wave_dat_y(pDC,hWnd,ftime, ihan);
        if(m_type == 2)set_wave_dat_y(pDC,hWnd,ftime, ihan);
        if(m_type >= 3 && m_type <= 8)set_wave_dat_x(pDC,hWnd,ftime, ihan);
        if(m_type >= 9)set_wave_dat_z(pDC,hWnd,ftime, ihan);
}
```

上の関数で太文字で示される関数が地震波形に関連し、その内容を以下に示す。この関数は地震波形上を赤い点が移動する仕組みを提供する。

```
//    ――――――――――――――――――――――――――――――――
//    ●　set_wave_dat_y：地震波形上の赤い点の移動処理
//    ――――――――――――――――――――――――――――――――
void CSf31wave::set_wave_dat_y(CDC* pDC,HWND hWnd,float ftime, int ihan)
{
//    ――――――――――――――――――  再描画要求があった場合
        if(m_data_set_ok == FALSE) return;                              !1
        if(ihan != 0) set_wave(pDC, hWnd);
        if(m_rewite_x) set_wave(pDC, hWnd);
        m_rewite_x = FALSE;
        CPen* pOriginalPen = pDC->SelectObject ( &NewPenx_a[111]);      !2
        int ist;
        int jst;
//    ――――――――――――――――――  通常の処理の場合
        if (ihan == 0)                                                  !3
        {
//    ――――――――――――――――――  一ステップ前の時刻で２回目の四角を描く
          if ( m_time_b_tm != -1. )                                     !4
```

```
            {
              ist = ar_sx + ar_dt * m_time_b_tm;                          !5
              jst =  -m_time_b_ii*ar_bi + ar_sy;
              CRect Rect;                                                 !6
              Rect.SetRect (ist -2, jst - 2,ist + 2, jst + 2);            !7
              pDC->SetROP2(R2_NOTXORPEN);                                 !8
              pDC->Rectangle ( &Rect);                                    !9
              pDC->SetROP2(R2_COPYPEN);                                   !10
             }
//      ――――――――――――――――――――  現在の時刻で１回目の四角を描く
              ist = ar_sx + ar_dt * ftime;                                !11
              m_time_b_ii = dat_set(ftime);                               !12
              m_time_b_tm = ftime;
              jst = -m_time_b_ii*ar_bi + ar_sy;                           !13
              CRect Rectx;
              Rectx.SetRect (ist -2, jst - 2,ist + 2, jst + 2);           !14
              pDC->SetROP2(R2_NOTXORPEN);                                 !15
              pDC->Rectangle ( &Rectx);                                   !16
              pDC->SetROP2(R2_COPYPEN);                                   !17
//      ――――――――――――――――――――  再描画処理を要求された場合の処理
           }else{
              m_time_b_ii = dat_set(ftime);                               !18
              m_time_b_tm = ftime;
              ist = ar_sx + ar_dt * ftime;
              jst = -m_time_b_ii*ar_bi + ar_sy;
              CRect Rectx;
              Rectx.SetRect (ist -2, jst - 2,ist + 2, jst + 2);
              pDC->SetROP2(R2_NOTXORPEN);
              pDC->Rectangle ( &Rectx);
              pDC->SetROP2(R2_COPYPEN);
           }
//      ――――――――――――――――――――  時刻を表示する
           pDC->SelectObject ( pOriginalPen);                             !19
           CFont* pOldFont=(CFont*) pDC->SelectObject(&newFontx);
           COLORREF oldcoler   =      pDC->SetTextColor(RGB(0,0,0));
           COLORREF oldbkcoler =      pDC->SetBkColor(RGB(160,200,160));
           if(m_type == 2) pDC->SetBkColor(RGB(20,120,250));
           char sTimeout[50];
           sprintf(sTimeout,"Time:[%6.2f]sec.  ",ftime) ;
           pDC->TextOut((int) m_Wave_rect.left+20,(int) m_Wave_rect.bottom-20, sTimeout);
//      ――――――――――――――――――――  リソースを元に戻す
           pDC->SelectObject(pOldFont);                                   !20
           pDC->SetTextColor(oldcoler);
           pDC->SetBkColor(oldbkcoler);
           if(ftime >= m_wave_time) m_data_set_ok = FALSE;
}
```

コメント番号に従ってプログラムの内容を説明する。十分に理解し、応用していただきたい。

1. 最初に、描画すべきかどうかメンバー変数 m_data_set_ok を用いてチェックする。再描画要求があった場合、関数 set_wave()を用いて地震

波形を描き直す。
2. ペンを選択する。
3. 以後は、通常の処理であり、一般には一回の処理で２回の四角を描く。
4. 最初に２回目の描画処理を行う。前回描画があったかどうかチェックし、一回目の四角の描画があった場合、その四角を消去するために描画モードを SetROP2(R2_NOTXORPEN)を使用して、排他的 OR とする。これで２回同じ描画を行うと元の状態となる。
5. 前回の時刻 m_time_b_tm と波形の大きさ m_time_b_ii から四角の位置を計算する。ここで、ar_sx はｘ座標原点、ar_sy はｙ座標原点のウインドウ座標を表す。
6. 四角を表すクラス Crect のオブジェクト Rect を生成する。
7. メンバー関数 Rect.SetRect()を用いて、四角の大きさを設定する。
8. 描画モードを、SetROP2(R2_NOTXORPEN)を使用して排他的 OR とする。
9. メンバー関数 pDC->Rectangle (&Rect)を用いて、四角を描画する。
10. 描画モードを元の状態に戻す。
11. 以後、第１回目の描画を行う処理を行う。現在の時刻を元にｘ座標位置を求める。
12. 現在の時刻と関数 dat_set()で地震波の大きさから、ｘ方向とｙ方向の位置を求める。この２つの値を次回の描画のために変数に保存する。
13. ｙ方向位置を求める。さらに、四角を表すクラス Crect のオブジェクト Rectx を生成する。
14. 第１回目の四角を関数 Rectx.SetRect()を用いて描く。
15. 描画モードを、SetROP2(R2_NOTXORPEN)を使用して排他的 OR とする。
16. メンバー関数 pDC->Rectangle (&Rectx)を用いて、四角を描画する。
17. 描画モードを元の状態に戻す。これで、第１回目の描画処理を終了する。
18. 以後、再描画された場合の四角描画処理を行う。上の第１回目の描画処理と同様である。
19. この関数の最後の処理であり、四角描画時刻をウインドウ上に表示する。フォントを選択し、テキストの色をセットする。最後に、関数 sprintf()と pDC->TextOut()で時刻を表示する。
20. フォントやテキストカラーを元に戻す。最後に、現在の時刻がこの地震波の最大時刻を越えているかどうかチェックし、超えている場合は、メンバー変数 m_data_set_ok を FALSE にする。