第２章　図形処理入門

ポイント：図形処理の基本的なテクニックを学ぶ

１．図形処理のための基本的な手続き

２．図形の描き方やラバーバンドの描き方

３．OnDraw 関数の使い方

４．図形の拡大・縮小、回転などの処理法

５．マウスによる図形位置指定

2.1 はじめに

本章では、プレゼンターシステムの各種の技術を例として、図形処理の基本的なテクニックを学ぶことにする。SPACE で使用している図形、例えば、ワイヤフレームによる透視図や表、グラフなどでは、高度なグラフィック用の関数を必要としない。線を描いたり、区画を塗りつぶしたり、線の色を変えたりと、非常に単純な関数を用いるのみである。これらは全て VC++の関数群に含まれており､特殊な図形用ライブラリを使用していない。ただし、ウインドウ上に図形を描くとき、多くの手続きが必要となる。ここでは、まず、この手続きについて説明し、次に図形の描き方、ラバーバンドの描き方、あるいは図形の拡大・縮小を行うための具体的な処理方法について学ぶ。

次に、特異な関数である OnDraw()について説明する。画面上で、子ウインドウがいくつか表示されているとき、ユーザーがそれらを移動したり、上下関係を変えたりした場合、見えなかった部分が見えるようになる。特に SPACE では図形やグラフがアニメーションによって時刻と共に変化するが、変化したその状態で再表示されなければならない。この部分の図形処理は、いったいどのように行われるのか。その秘密はこの関数にあり、再描画が必要とウインドウシステムが判断したとき、自動的にこの関数が呼ばれ、ウインドウ内で図形の再描画が行われる。ここでは、この関数をどのように扱えば良いか、具体的に示して解説する。

最後に、ユーザーの指示によって図形を変化させる方法について学ぶ。この種の仕組みを適切に組み込むことで、マウスのボタンを押しながら前後にドラッグして、図形を拡大・縮小、あるいは回転させることができる。

2.2 簡単な図形描画と手続き

2.2.1 図形を描く前の手続き

ウインドウの中に図形を描くには、多くの手続きが必要となる。ここでは、その手続きの基本を勉強しよう。図形のキャンパスは、一般には子ウインドウになる。この子ウインドウはユーザーによってその大きさは不定であり、これを常に意識してプログラムする必要がある。また、ペンの色や線の太さはどのように設定するのか、あるいは、どこに描くのかを考えなければならない。この手続きを次にようにまとめる。

１．ウインドウの大きさを取得する。
２．そのウインドウと同じ大きさのメモリー記憶領域を確保する。
３．そのメモリー記憶領域をクリアする。
４．そこに図形を描く。
５．メモリー記憶領域のデータを VRAM に高速転送する。
６．メモリー記憶領域、その他のリソースを解放する。

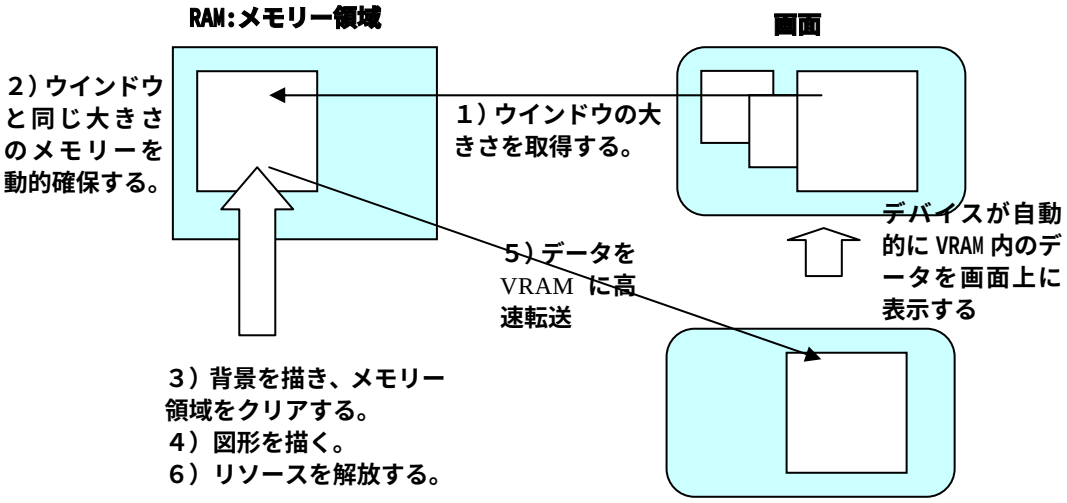

図 2-1 図形をウインドウに描く際の基本的な手続

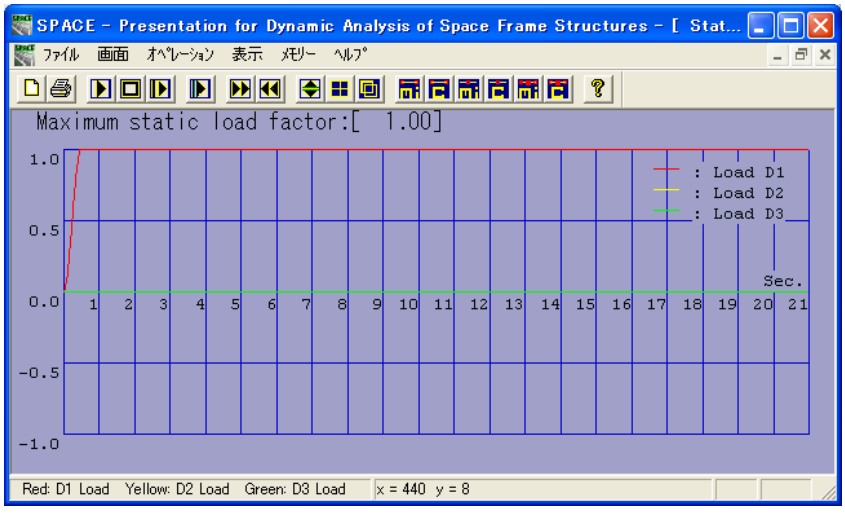

図 2-2 荷重係数の時間変化グラフ

上の手続きを記述した典型的な関数 set_load_frame()を以下に示す。この関数は、図 2-2 のように荷重の時間変化をグラフ表示する。関数 set_load_frame()を以下に示す。

これらは全て、VC++の関数である。引数などは、VC++のリファレンスマニュアルを参照されたい。

```
BOOL CSf31load::set_load_frame(CDC* pDC,HWND hWnd)                          !1
{
    BOOL hantei = FALSE;
//――――――――――――――――――――――――――――― ● ウインドウの大きさを取得
    CDC* pMemDC = new CDC;                                                  !2
    ::GetClientRect(hWnd ,&m_load_rect);                                    !3
    pMemDC->CreateCompatibleDC(pDC);                                        !4
//――――――――――――――――――――――――――――― ● メモリー領域を確保
    CBitmap *pMemBitmap  = new CBitmap;                                     !5
    pMemBitmap -> CreateCompatibleBitmap(pDC,m_load_rect.right,m_load_rect.bottom);  !6
    CBitmap *poldBitmap =(CBitmap*)pMemDC -> SelectObject(pMemBitmap);      !7
//――――――――――――――――――――――――――――― ● 背景を描く
    disp_frame_load(pMemDC, -1.);                                           !8
//――――――――――――――――――――――――――――― ● 図形を描く
    hantei= set_load_disp(pMemDC);                                          !9
//――――――――――――――――――――――――――――― ● VRAM にデータを高速転送
    pDC->BitBlt(0,0,m_load_rect.right,m_load_rect.bottom,pMemDC,0,0,SRCCOPY);  !10
//――――――――――――――――――――――――――――― ● リソースを解放
    delete pMemDC->SelectObject(poldBitmap);                                !11
    delete pMemDC;                                                          !12
    return (hantei);
}
```

先の手続きに関連して、上の関数を説明する。

図形描画の基本的手続き

１．関数の引数は２つであり、アクティブなデバイスコンテキスト pDC と、そのウインドウハンドル hWnd である。この関数では、この２つの引数を受け取り、アクティブなウインドウに図形を描くことになる。

２．新規のデバイスコンテキスト pMemDC を作成する。

３．ウインドウハンドル hWnd を渡して、このウインドウの四角形領域を m_load_rect に取得する。

４．メモリー上のデバイスコンテキスト pMemDC を、受け渡されたアクティブウインドウのデバイスコンテキスト pDC と同じ状態とする。

５．メモリー上に新たなビットマップ用記憶領域 pMemBitmap を作成する。

６．作成した記憶領域 pMemBitmap の大きさを、先にセットしたウインドウの四角領域 m_load_rect を用いて、受け渡されたアクティブなウインドウと同じとする。

７．デバイスコンテキスト pMemDC の選択先をこの pMemBitmap とする。つまり、pMemDC のデバイスを使用した図形は、全てこの pMemBitmap 領域内に描かれることになる。
以上の２から７は、ウインドウ上に図形や表などを描くとき、常に行う標準的な手続きである。
８．この関数では、メモリー内の pMemBitmap 領域内を、背景色で塗りつぶす処理を行う。
９．ここで、実際にメモリー内の pMemBitmap 領域に図形を描く。
10．pMemBitmap 領域のデータを VRAM に、関数 pDC->BitBlt()を使用して、高速転送を行う。
11．メモリー内の pMemBitmap 領域を解放する。
12．デバイスコンテキスト pMemDC を解放する。

VRAM とは、モニターに直接表示する情報を保持するメモリーであり、ここにビットマップで情報を書き込むと、デバイスが自動的に VRAM 内のデータを画面上に表示する。

上の処理では、一端メモリー内に図形を描いた後、VRAM にその図形を転送している。何故、そのような面倒な処理を行うのかについて考えよう。VRAM に直接図形を描くと、ウインドウ上ではその過程が見えることになる。これがアニメーションではチラツキの原因となる。それを避けるために、このように一端メモリーに描くことになる。子ウインドウと同じ状態をメモリー内に設定するために、上の処理２から７を行う。また、メモリーに図形を描いた後、関数 pDC->BitBlt()を使用して、情報を高速転送する。この処理は、図形を描く場合、全て同じである。無論、ここで動的確保したリソースやメモリー領域は、解放しなければならない。もし解放せずにアニメーションなどでこの関数を多用すると、リソースやメモリー領域が徐々に侵食され、最後には重大なエラーとなる。他の関数でも同様に、リソースを生成した場合は、必ず解放しなければならない。

2.2.2 図形の描き方

ここでは、具体的に図形を描く処理について学ぶ。例として用いる荷重係数の時間変化を示すグラフは、処理が単純であり、理解は容易である。ここでは、次の２つの関数を用いて図形が描かれる。最初の関数ではメモリー内の領域を背景色で塗りつぶす処理を、次の関数では具体的に図形を描く処理が記述されている。２つの関数の引数として、メモリー領域へのデバイスコンテキストが受け渡される。

```
disp_frame_load(pMemDC, -1.);
hantei= set_load_disp(pMemDC);
```

最初の関数 disp_frame_load()を以下に示す。仮引数はデバイスコンテキストのポインターpDC であり、このポインターを用いて各種の図形用関数を呼ぶことになる。この関数の実引数がメモリー領域のポインターpMemDC であることから、この関数内の図形用関数はメモリー領域に対して処理されることになる。

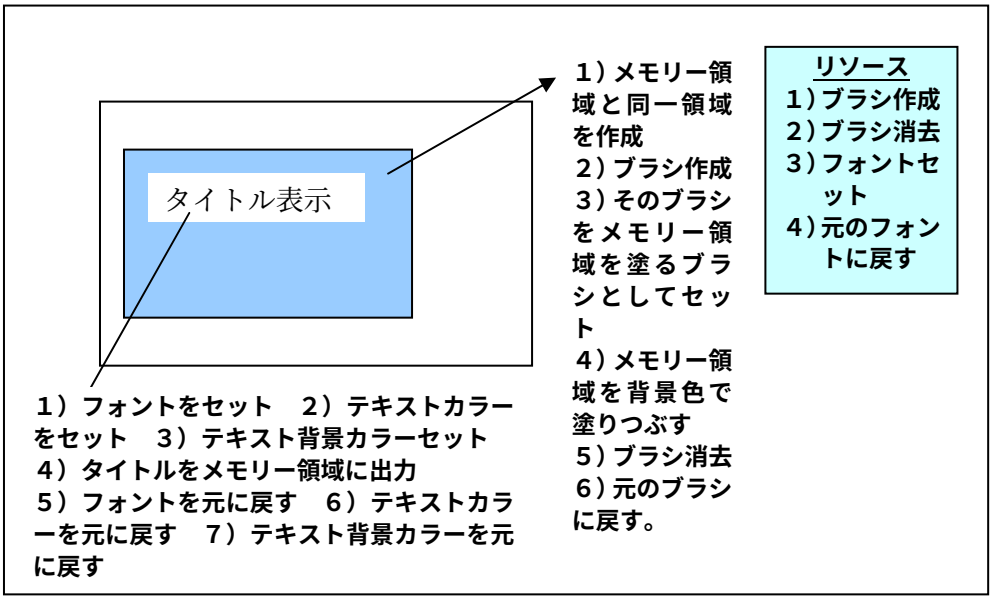

図 2-3 メモリー領域に背景を描き、タイトルを書く

```
void CSf31load::disp_frame_load(CDC* pDC,float ftime)
{
//――――――――――――――――――――――――――――――  ●　ウインドウと同じ領域をセット
     CRgn theRgn;                                                    !1
     theRgn.CreateRectRgn (m_load_rect.left,m_load_rect.top,         !2
                           m_load_rect.right,m_load_rect.bottom);
//――――――――――――――――――――――――――――――  ●　ブラシでその領域を塗りつぶす
     CBrush MyBrush( RGB(160,160,200));                              !3
     CBrush* pOldBrush = pDC->SelectObject(&MyBrush);                !4
     pDC->FillRgn (&theRgn,&MyBrush);                                !5
     pDC->SelectObject(pOldBrush);                                   !6
     MyBrush.DeleteObject();                                         !7
//――――――――――――――――――――――――――――――  ●　フォントなどをセットする
     CFont* pOldFont=(CFont*) pDC->SelectObject(&newFontx);          !8
     COLORREF oldcolor   =  pDC->SetTextColor(RGB(0,0,0));           !9
     COLORREF oldbkcolor =  pDC->SetBkColor(RGB(160,160,200));       !10
//――――――――――――――――――――――――――――――  ●　タイトルを書く
     char sTimeout[50];                                              !11
     sprintf(sTimeout,"Maximum static load factor:[%6.2f]",m_load_data_max) ; !12
     pDC->TextOut((int) m_load_rect.left+20,(int) m_load_rect.top,
                  sTimeout);                                         !13
//――――――――――――――――――――――――――――――  ●　フォントを元に戻す
     pDC->SelectObject(pOldFont);                                    !14
     pDC->SetTextColor(oldcolor);
     pDC->SetBkColor(oldbkcolor);
}
```

以下にプログラムの内容を説明しよう。

１．クラス theRgn のオブジェクトを生成する。
２．クラスのメンバー関数 CreateRectRgn()を用いて、四角領域をセットする。引数は、先に取得したウインドウの四角形領域 m_load_rect の左上と右下の座標である。
３．リソースの一つであるブラシを MyBrush として生成し、その色を関数 RGB()でセットする。この関数の引数は、赤、緑、青の順で、0 から 255 までで色の濃さを表す。例えば、RGB(0,0,0)は黒を、RGB(255, 255, 255)は白を意味する。
４．MyBrush を描画用オブジェクトとして設定し、前のオブジェクトを pOldBrush にセットする。
５．関数 pDC->FillRgn()を用いて、四角の領域、つまり、そのウインドウの全領域 theRgn をブラシ MyBrush で塗りつぶす。
６．塗り終えた後は、オブジェクトを前の pOldBrush に戻しておく。
７．リソースである MyBrush を消去し、解放する。
これで、確保したメモリーの全てが、背景色で塗りつぶされ、クリアされたことになる。
８．ここからは、グラフのタイトルを書く処理を行う。まず、描画用オブジェクトをフォント newFontx とし、以前のフォントを pOldFont にセットする。フォント newFontx は、他の関数で既に生成されている。
９．フォントのテキストカラーを関数 pDC->SetTextColor()を用いてセットする。この関数の引数は、RGB(0,0,0)であり、その内容は先に示したとおりである。また、元のフォントカラーを oldcolor に保存しておく。
10．フォントのテキスト背景色を関数 pDC->SetBkColor()を用いてセットする。引数は、pDC->SetTextColor()と同様である。この 2 つの関数をセットすると、文字の色と文字間の色が設定されたことになる。また、元のフォント背景カラーを oldbkcolor に保存しておく。
文字の背景色をセットする他に、次の関数を使用する場合がある。

```
pDC->SetBkMode(TRANSPARENT);
```

この関数は、文字の間は透視として設定するもので、前の画像がそのまま文字の間から見えることになる。
11．タイトル用のキャラクター領域 sTimeout[50]を生成する。
12．キャラクター領域にタイトルを出力する。

13. 関数 pDC->TextOut()を用いて、先にタイトルを出力したキャラクター領域を用いて、メモリー領域に描画する。
14. リソースを元のフォントやテキストカラー、テキスト背景色に戻しておく。

背景とタイトルを描いたメモリー領域に、次の関数を用いてグラフを描画する。図形をウインドウ上に描画する場合、ウインドウの大きさはユーザーによって自由に変更されるため、図形の大きさをどのように決定するかに注意しなければならない。多くの手法が考えられるが、SPACEで用いている方法の一つは、ウインドウの大きさの変更に合わせて図形の拡大率を変更し、常に図形がウインドウ内にうまく収まるように設計するものである。他の一つは、図形を任意の拡大率に従って描画する方法であり、大きさの調整はユーザーに任せることになる。つまり、ユーザーが拡大率を変更することで、図形の大きさが自動的に更新されるようにプログラムする方法である。後者については、第４章の透視図の描画で説明する。

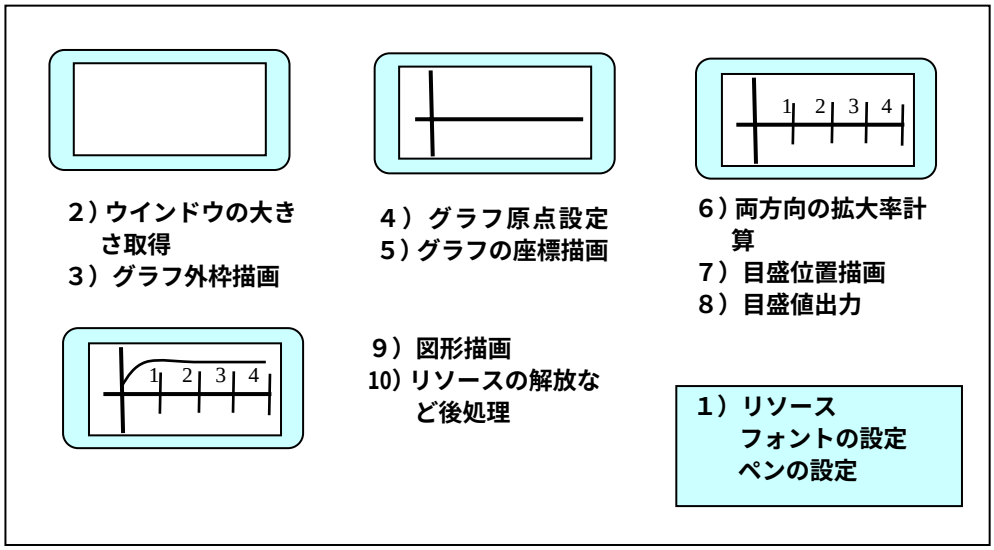

図 2-4 グラフの一般的な描き方

ここでは、前者の方法による関数 set_load_disp()を以下に示す。この関数は、図 2-2 のグラフを描画する。

```
BOOL CSf31load::set_load_disp(CDC* pDC)
{
//――――――――――――――――――――――――――――――――――――
//――――――――――――――――――――――――――――  ●  1）  初期設定
//――――――――――――――――――――――――――――――――――――
     m_data_set_ok = TRUE;
     CFont newFont;                                                    !1
     int point =16;                                                    !2
```

```
    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");                                    !3
    CFont* pOldFont = (CFont*) pDC->SelectObject(&newFont);          !4
    int i1,i2,i3;                                                    !5
    i1= 0;
    i2= 10;
    i3= 10;
    COLORREF oldcolor =    pDC->SetTextColor(RGB(i1,i2,i3));         !6
    i1= 160;                                                         !7
    i2= 160;
    i3= 200;
    COLORREF oldbkcolor =  pDC->SetBkColor(RGB(i1,i2,i3));           !8
    CPen* pOriginalPen = pDC->SelectObject (&NewPenx_a[108]);        !9
//――――――――――――――――――――――――――――――――――――――――――――
//――――――――――――――――――――――――――― ●　２）拡大率とグラフ外枠・目盛の描画
//――――――――――――――――――――――――――――――――――――――――――――
    char buffer[20];                                                 !10
    int x_axi;                                                       !11
    float y_axi;
//――――――――――――――――――――――― ●　グラフ外枠設定
    ar_w = m_load_rect.right - m_load_rect.left - 60;                !12
    ar_h = (m_load_rect.bottom  - m_load_rect.top - 60)*0.5;
    if(ar_w < 1) m_data_set_ok = FALSE;                              !13
    if(ar_h < 1) m_data_set_ok = FALSE;
//――――――――――――――――――――― ●　グラフ外枠が描けない場合処理終了
    if( m_data_set_ok == FALSE) return ( m_data_set_ok);             !14
//――――――――――――――――――――― ●　グラフ座標原点設定
    ar_sx = m_load_rect.left + 40;                                   !15
    ar_sy = m_load_rect.top + ar_h + 30;                             !16
//――――――――――――――――――――― ●　グラフ座標描画
    pDC->MoveTo ( ar_sx, ar_sy);                                     !17
    pDC->LineTo ( ar_sx + ar_w, ar_sy);
    pDC->MoveTo ( ar_sx, ar_sy + ar_h);
    pDC->LineTo ( ar_sx, ar_sy - ar_h);
//――――――――――――――――――――― ●　表示データが０の場合終了
    if(m_load_max == 0 ) m_data_set_ok =FALSE;                       !18
    if( m_data_set_ok == FALSE) return ( m_data_set_ok);
//――――――――――――――――――――― ●　ｘ軸拡大率計算 ar_dt
    int ist, jst;
    ar_dt = (float) ar_w /m_load_time;                               !19
    float ar_dtt = m_load_delt*ar_dt;
//――――――――――――――――――――― ●　ｙ軸表示拡大率の調整
    float load_data_max =1.;                                         !20
    float ld_dt = 0.5;
    if(m_load_data_max > 1.) {load_data_max =1.5;                    !21
    ld_dt = 0.5;}
    if(m_load_data_max > 1.5) {load_data_max =2.0;                   !22
    ld_dt = 1.0;}
    if(m_load_data_max > 2.){                                        !23
```

```
        int iload_data_max =m_load_data_max+1;
        load_data_max = iload_data_max;
        ld_dt = 1.0;
      }
  //――――――――――――――――――――――  ●  ｙ軸拡大率計算 ar_bi
      ar_bi = (float)ar_h / load_data_max;                          !24
  //――――――――――――――――――――――  ●  ｘ軸目盛描画
      jst = ar_sy;                                                  !25
      int op_time = (int) m_load_time;
      for ( int i=0;i < op_time+1; i++)                             !26
      {
        ist = ar_sx + ar_dt * ((float)i+1);                         !27
        x_axi= (i+1);
        sprintf(buffer,"%2d",x_axi);                                !28
        pDC->TextOut(ist-16,jst+2,buffer);
        pDC->MoveTo ( ist, jst+ar_h);                               !29
        pDC->LineTo ( ist, jst-ar_h);
      }
  //――――――――――――――――――――――  ●  ｙ軸目盛描画
      if(m_load_data_max != 0.){                                    !30
        int iij =load_data_max/ld_dt;                               !31
        float ar_yy = ar_bi*ld_dt;                                  !32
        jst = ar_sy ;
        y_axi= 0.;
        sprintf(buffer,"%3.1f",y_axi);                              !33
        pDC->TextOut(ar_sx-27,jst,buffer);
        for ( i=0;i <iij; i++)                                      !34
        {
          jst = ar_sy - ld_dt*ar_bi * (i+1);                        !35
          ist = ar_sy + ld_dt*ar_bi * (i+1);
          y_axi= (i+1)*ld_dt;
          sprintf(buffer,"%3.1f",y_axi);                            !36
          pDC->TextOut(ar_sx-27,jst,buffer);
          y_axi= -y_axi;
          sprintf(buffer,"%3.1f",y_axi);
          pDC->TextOut(ar_sx-35,ist,buffer);
          pDC->MoveTo ( ar_sx, jst);                                !37
          pDC->LineTo ( ar_sx+ar_w, jst);
          pDC->MoveTo ( ar_sx, ist);
          pDC->LineTo ( ar_sx+ar_w, ist);
        }
      }
  //――――――――――――――――――――――  ●  ｘ軸タイトル
      sprintf(buffer,"Sec.");                                       !38
      pDC->TextOut(ar_sx+ar_w-35,ar_sy-16,buffer);
  //――――――――――――――――――――――  ●  ｙ軸タイトル
      sprintf(buffer,": Load D1");                                  !39
      pDC->TextOut(ar_sx+ar_w-90,ar_sy-ar_h+10,buffer);
      sprintf(buffer,": Load D2");
      pDC->TextOut(ar_sx+ar_w-90,ar_sy-ar_h+26,buffer);
      sprintf(buffer,": Load D3");
      pDC->TextOut(ar_sx+ar_w-90,ar_sy-ar_h+42,buffer);
  //――――――――――――――――――――――――――――――――――――――――――――――――
  //――――――――――――――――――――――――――――  ●  ３）  実際のグラフ描画
```

```
//――――――――――――――――――――――――――――――――――――――――――――
     pDC->SelectObject ( &NewPenx_a[106]);                              !40
     int jj;
     for( int j = 0; j < 3; j++){                                       !41
       if(j == 1) pDC->SelectObject ( &NewPenx_a[107]);                 !42
       if(j== 2) pDC->SelectObject ( &NewPenx_a[113]);
       pDC->MoveTo ( ar_sx+ar_w-120, ar_sy-ar_h+16*j+15);               !43
       pDC->LineTo ( ar_sx+ar_w-100, ar_sy-ar_h+16*j+15);
       jj= j*m_load_max;                                                !44
       int istt = ar_sx;
       int jstt = -FF_load[jj]*ar_bi+ ar_sy;
       pDC->MoveTo ( istt, jstt);                                       !45
       for(  i = 1; i < m_load_max-1; i++)                              !46
       {
         ist = ar_sx + ar_dtt * (float)i;                               !47
         jst = -FF_load[i+jj]*ar_bi + ar_sy;
         pDC->LineTo ( ist, jst);                                       !48
       }
       ist = ar_sx + ar_dt * m_load_time;                               !49
       jst = -FF_load[jj+m_load_max-1]*ar_bi + ar_sy;
       pDC->LineTo ( ist, jst);
     }
//――――――――――――――――――――――――――――――――――――――――――――
//――――――――――――――――――――――――――――  ●　４）後処理
//――――――――――――――――――――――――――――――――――――――――――――
     m_time_b_tm = -1.;
     pDC->SelectObject ( pOriginalPen);                                 !50
     pDC->SelectObject(pOldFont);
     pDC->SetTextColor( oldcolor);
     pDC->SetBkColor(oldbkcolor);
     return (m_data_set_ok);
}
```

関数は少し長いが、最初の図形用プログラムなのでゆっくりと眺め、理解していこう。プログラムは４つの部分に分かれている。最初の１）は初期設定であり、この関数で使用するリソースの設定を行っている。２）は少し長いプログラムとなっているが、ここでは、図形の拡大率とグラフの外枠、目盛などを描画している。３）では、実際のグラフを描く処理を行っているが、データが配列に収まっているため処理内容は非常に単純である。最後の４）は、後処理で、リソースなどを元の状態に戻している。それでは、実際のプログラムに即して、内容の説明を行うことにしよう。

１．初期設定のルーチンで、まず、新規のフォントクラス newFont を生成する。
２．このフォントの大きさを 16 ポイントとするために、変数に値 16 をセットする。

３．フォントクラスのメンバー関数 newFont.CreateFont()を用いて、このフォントの内容を設定する。
４．デバイスコンテキストで使用するフォントオブジェクトを newFont に設定する。以前のフォントオブジェクトを pOldFont に保存する。
５．テキストカラーを決定するために、変数に RGB 値を設定する。
６．先に設定した変数を用いて、関数 pDC->SetTextColor()によりテキストカラーを決定する。
７．同じく、テキストの背景カラーを決定するために、変数に RGB 値を設定する。
８．設定した変数を用いて、関数 pDC->SetBkColor()により背景カラーを決定する。
９．使用するペンの種類を NewPenx_a[108]に設定する。また、以前のフォントを pOriginalPen に保存する。なお、システムが立ち上がる際に、NewPenx_a[108]は既に設定されている。
10．ここ以降は、拡大率とグラフ外枠・目盛の描画処理である。
目盛の値などの出力時に使用するキャラクター配列 buffer[20]を定義する。
11．グラフの原点座標を保存する変数を定義する。
12．グラフを描画するｘ方向軸とｙ方向軸の大きさを設定する。ｘ軸の大きさ ar_w はウインドウの横方向大きさから 60 を引いた値、ｙ軸の大きさ ar_h はウインドウの縦方向大きさ-60 の半分とする。値 60 は図形の余白を示す。
13．両者の値が負の場合は、この関数の戻り値 m_data_set_ok を FALSE とする。
14．この関数の戻り値 m_data_set_ok が FALSE の場合、ウインドウが小さくてグラフ軸が描けないことを意味し。処理を中止してこの関数から抜ける。
15．ウインドウの大きさから、グラフの原点であるｘ座標 ar_sx を設定する。
16．同じく、ｙ座標の原点 ar_sy を設定する。
17．先に設定したグラフ原点と座標軸の大きさを用いて、グラフの座標軸を描画する。最初にｘ軸を、次にｙ軸を描く。
18．グラフ用データ総数 m_load_max が０の場合、この関数の戻り値 m_data_set_ok を FALSE とする。この関数の戻り値 m_data_set_ok が FALSE の場合、処理を中止してこの関数を抜ける。
19．ｘ軸方向のデータ増分値 ar_dt と目盛増分値 ar_dtt を求める。こ

こで、m_load_delt は目盛表示間隔である。
20. ｙ軸方向の表示最大値 load_data_max と表示間隔 ld_dt を設定する。
21. 実際のｙ軸データの最大値が１より大きい場合は、表示最大値 load_data_max を 1.5 に、表示間隔 ld_dt を 0.5 に設定する。
22. 実際のｙ軸データの最大値が 1.5 より大きい場合は、表示最大値 load_data_max を 2.0 に、表示間隔 ld_dt を 1.0 に設定する。
23. 実際のｙ軸データの最大値が 2.0 より大きい場合は、表示最大値 load_data_max を load_data_max＋1 の値に、表示間隔 ld_dt を 1.0 に設定する。
24. ｙ軸方向の拡大率 ar_bi を、ｙ軸グラフの大きさ ar_h をデータ最大値 load_data_max で割って求める。
25. これ以降でｘ軸目盛とその値を描画する。まず、ｙ軸位置 jst をｙ軸の原点座標 ar_sy にセットする。
26. ｘ軸の個数（秒数）分、ループ処理を行う。
27. 目盛のｘ座標位置 ist を計算する。ｘ座標の値 x_axi を計算する。
28. ｘ座標の値をバッファ領域 buffer に出力し、次に画面に描画する。
29. ｘ座標の目盛を描く。
30. ｙ軸の最大値がゼロでない場合、次の処理を行い、ｙ軸の目盛を描く。
31. ｙ軸の最大値と目盛の増分値によって、目盛の描画個数 iij を計算する。
32. 一目盛の画面上の大きさ ar_yy を計算する。次に、ｙ軸の原点座標 ar_sy を jst にセット、ｙ軸の値 y_axi に０をセットする。
33. ｙ軸原点の値を画面に出力する。
34. 目盛の個数 iij 分、以下の処理をループする。
35. ｙ軸の正方向座標 jst を計算する。また、負方向の座標 ist も計算する。
36. 正側のｙ軸目盛の値をバッファに出力し、次に画面に描く。また、負側のｙ軸目盛の値をバッファに出力し、次に画面に描く。
37. ｙ軸の目盛を描く。ここでは、正側と負側の２本の目盛を描く。
38. ｘ軸のタイトル「sec.」を描く。
39. 動的解析における長期荷重のタイトル「Load D1」、「Load D2」、「Load D3」を画面に出力する。
40. これ以降、ｘ軸目盛を描く。まず、ペンを NewPenx_a[106]に変更する。
41. ３種類の荷重について、ループ処理を行う。

42. 荷重の種類で描くペンを変更する。
43. 先の荷重タイトルの横に線分を描く。
44. 各荷重に対する配列 FF_load[]の先頭番地 jj を求める。各荷重の先頭の値をセットする。
45. ペンを各荷重の先頭へ移動する。
46. 荷重数分、ループ処理を行う。
47. ｘ座標とｙ座標を計算する。
48. ペンを上で計算した値まで、描きながら移動する。
49. 最終位置の座標を計算し、ペンを描きながら移動する。
50. この関数で設定したリソースを、全て以前のリソースに戻す。

以上が、ウインドウに適合するグラフの典型的な処理方法である。この方法によれば、ウインドウが変更になってもグラフは描き直され、必ずウインドウに適合したグラフが描かれることになる。他の再描画の方法については後節で説明する。このグラフの描き方をよく理解して、自分のプログラムで利用すると良いでしょう。

2.2.3 ラバーバンドの描き方

本節では、ウインドウ用図形処理で、しばしば使用されるラバーバンドの描き方について勉強しよう。ラバーバンドとは、領域を特定したり、オブジェクトを指定したりするときに使用するものである。

ここで説明するプログラムは、モデラーで使用されている関数であり、その仕様は以下のようである。まず、マウスの左ボタンを押してドラッグすると、そのマウスの動きに従って四角の線分が表示される。最後に、左ボタンを離すことで、その四角に囲まれた領域が特定されることになる。以上の処理を以下のプログラムで行う事になる。

モデラーの開発コード名は、SF2stであり、ここで説明するクラスは、Csf2stView である。

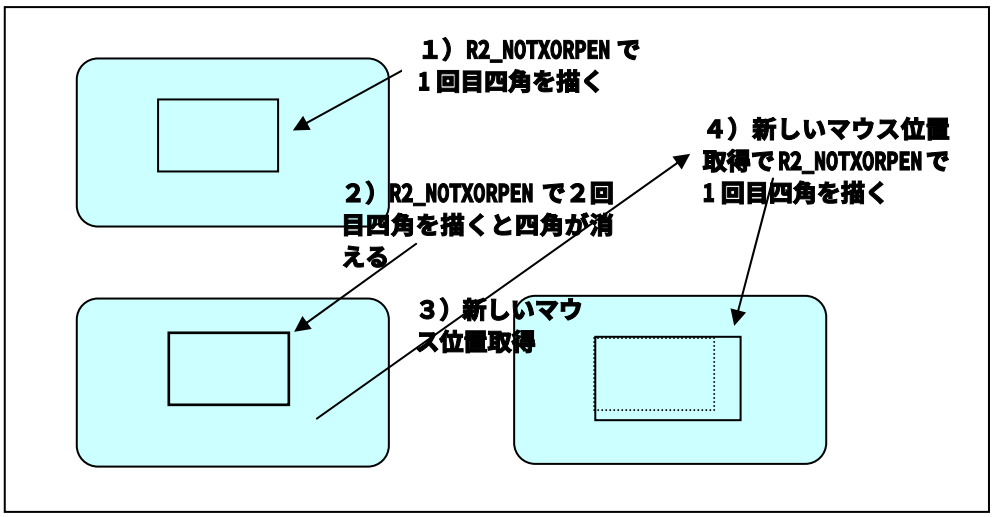

図 2-5 ラバーバンドの描き方

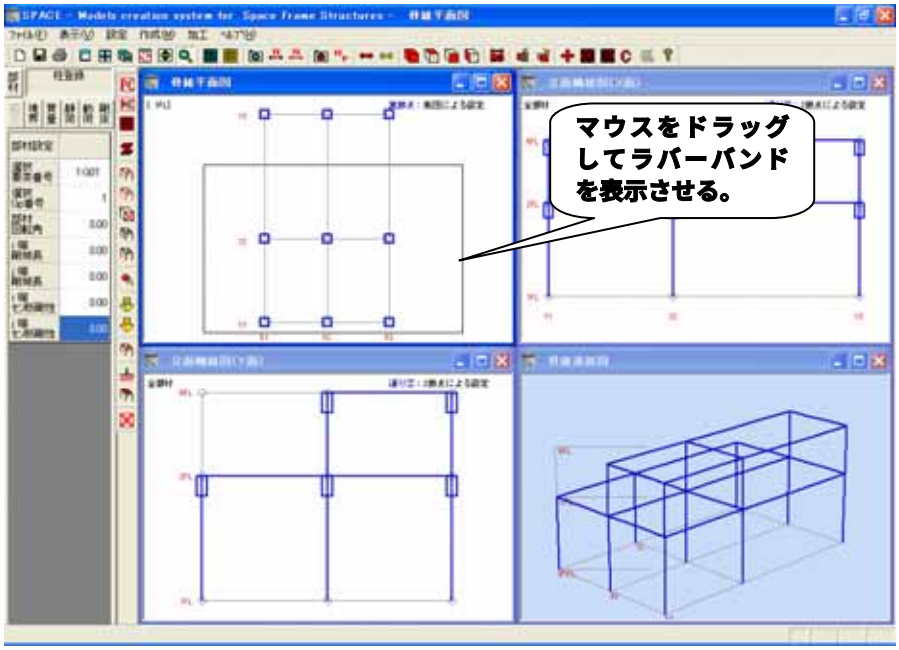

図 2-6 ラバーバンドの表示

ラバーバンドは、メッセージによる関数の駆動メカニズムとメッセージマップ内の次の３つの関数によって作られる。ここで、大きな役割を果たす変数は、ButtonDown と F_hantei である。通常、何も指定していないときでも、マウスが移動するとメッセージが常にアクティブなオブジェクト CSf2stView に送られる。そのメッセージによって OnMouseMove が実行されるが、変数 ButtonDown が FALSE であるため、直ちにこの関数を抜け、実際に処理が何も行われないことになる。逆に、何か動作させたい場合は、変数 ButtonDown を TRUE にする必要がある。また、マウス操作で、多くの処理を行いたい場合は、使用状況によって、変数 F_hantei の値を変化させ、適切な動作を行わせるわけである。

ここでは、SPACE モデラーのラバーバンドの仕様を用いて説明する。無論、他の仕様を用いてもラバーバンドを作成することは可能である。ここで説明するプログラムは典型的なラバーバンド作成法であり、十分にそのメカニズムを理解することが大切である。

ラバーバンドを作成するには、まず、この変数 ButtonDown を TRUE に変え、さらに switch 文で使用される変数 F_hantei を４とする必要がある。この処理は一般的に関数 OnLButtonDown()で行うことになる。関数 OnMouseMove()では、他の処理も多く行われるので、どのような場合にラバーバンドが描かれるかを整理しておかなければならない。この仕様に従って、関数 OnLButtonDown()で変数 ButtonDown を TRUE に、変数 F_hantei を４にすると、関数 OnMouseMove()の中で、ラバーバンドが描かれることになる。

変数 F_hantei を４にセットするのは、モデラーの仕様であり、この４の場合、ラバーバンドを描くように設定されている。

ラバーバンドの描画終了は、関数 OnLButtonUp()で行われる。ドラッグが終わり、マウスの左ボタンがアップされるとこの OnLButtonUp 関数が呼ばれ、まず、その中でラバーバンド処理であることをチェックし、変数 ButtonDown を FALSE に、変数 F_hantei を 0 に変更することになる。次に、最終的なラバーバンドの位置を用いて、独自の処理が実行される関数が呼ばれることになる。

ラバーバンドを実現するプログラムのメカニズムが理解できただろうか。このメカニズムは、次に示すクラス CSf2stView のメッセージマップによって実現する。マウスからのメッセージを受けて、次の３つの太文字で示される関数が、先に示した３つの関数を各々呼び出すことになる。

```
//    ――――――――――――――――――――――――――――――――
//    ●    ウインドウメッセージマップ
//    ――――――――――――――――――――――――――――――――
IMPLEMENT_DYNCREATE(CSf2stView, CView)
BEGIN_MESSAGE_MAP(CSf2stView, CView)
        //{{AFX_MSG_MAP(CSf2stView)
        ON_WM_LBUTTONDOWN()
        ON_WM_MOUSEMOVE()
        ON_WM_LBUTTONUP()
        ON_WM_RBUTTONDOWN()
        ON_WM_RBUTTONUP()
        ON_WM_TIMER()
         ・
         ・
        ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
        ON_MESSAGE(IDS_MESSAGE_WIND, OnMessageWind)
END_MESSAGE_MAP()
```

最初に、左ボタンダウンの関数 OnLButtonDown()を観察しよう。この関数の引数は２つで、nFlags と point である。最初の引数は、左ボタンが押されたとき、他のキーが同時に押されているかどうかを表す。また、次のキーはボタンが押されたときのアクティブウインドウの位置情報が保存されている。ラバーバンドに関連する部分を以下に示す。

```
//    ――――――――――――――――――――――――――――――――
//    ●    OnLButtonDown : 左ボタンダウン処理
//    ――――――――――――――――――――――――――――――――
void CSf2stView::OnLButtonDown(UINT nFlags, CPoint point)
{
        CSf2stDoc* pDoc = GetDocument();
        F_hantei=-1;
        HWND hwnd = this->GetSafeHwnd();
```

```
          if(IsWindow(hwnd) == FALSE) return;
          int  fno_yx = GetWindowLong(hwnd,GWL_USERDATA);
          int fno_xy = getwindx(fno_yx);
          if(fno_xy == -1) return;
          fno_xx = (int) fno_yx;
//      ――――――――――――――――――――  節点等の集団検索
          if ( m_kensaku == 1 &&
                  (m_radio_tohroku_houhou == 1 || m_radio_tohroku_houhou == 3)){
             if(fno_xy == HEIMEN ||  fno_xy == RITUMEN_X || fno_xy == RITUMEN_Y) {
                     if(nFlags & MK_SHIFT) {
                           m_kensaku=0;
                           m_shohkyo_shori=0;
                     if(Sparameter.m_box_contl == 0){
                           if(m_radio_tohroku_houhou == 1){
                                   MessageBox("登録処理を中止します");
                           }else{
                                   MessageBox("集団消去処理を中止します");
                           }
                     }
                           Invalidate(TRUE);
                           W_update_shori();
                           return;
                      }
//――――――――――――――――――――――――――――――――――
//――――――――――――――――――――  ●　ラバーバンドのための設定
//――――――――――――――――――――――――――――――――――
                           if(m_radio_tohroku_houhou == 3){
                           m_shohkyo_shori=1;
                           m_bmx=point.x;                                  !1
                           m_bmy=point.y;                                  !2
                           ButtonDown = TRUE;                              !3
                           SetCapture();                                   !4
                           CView::OnLButtonDown(nFlags, point);             !5
                           F_hantei =4;                                    !6
//                                 MessageBox("節点・部材の集団消去を行います。");
                           return;
                           }
          }
     .
     .
}
```

ラバーバンドに関連して、次に説明するコード部分が実行されることになる。

１．左ボタンを押した時のマウスのｘ座標をメンバー変数 m_bmx に保存する。
２．同じく、左ボタンを押した時のｙ座標をメンバー変数 m_bmy に保存する。
３．変数 ButtonDown を TRUE とする。

４．関数 SetCapture()によって、マウスの位置がアクティブウインドウを外れても、このオブジェクトへメッセージが届くようにする。
５．Cview クラスの関数 OnLButtonDown()を実行する。
６．F_hantei を４にして、ラバーバンド処理を行うようにする。

これで、ラバーバンドのための初期設定が終了し、いよいよ OnMouseMove 関数によって、実際のラバーバンドを描くことになる。次に、この関数 OnMouseMove()のラバーバンドに関連する部分を示す。

```
//     ―――――――――――――――――――――――――――――――
//     ●    OnMouseMove : マウス移動処理
//     ―――――――――――――――――――――――――――――――
void CSf2stView::OnMouseMove(UINT nFlags, CPoint point)
{
  CString str;
  CMainFrame* pFrame = (CMainFrame*) AfxGetApp()->m_pMainWnd;
  CStatusBar* pStatus = &pFrame->m_wndStatusBar;
//―――――――――――――――――――――――――――――――――――
//――――――――――――――――――――  ●  初期設定
//―――――――――――――――――――――――――――――――――――
  if(ButtonDown){                                                    !1
        CDC* pDC = GetDC();                                          !2
        HWND hwnd = this->GetSafeHwnd();
        CPoint Dpoint =   F_point - point;                           !3
        int F_movedxy =  Dpoint.x   + Dpoint.y ;
        float xx1=(float)(F_point.x);                                !4
        float xx2=(float)(F_point.y);
        float xx3=(float)(point.x);
        float xx4=(float)(point.y);
        int  oldROPCode =pDC->SetROP2(R2_NOTXORPEN);                 !5
        F_point = point;                                             !6
        int kkk =0;                                                  !7
        float a_Zoomy;
        int ii= fno_xx-1;
switch (F_hantei){                                                   !8
case 0:
・
・
break;
case 1:
・
・
//―――――――――――――――――――――――――――――――――――
//――――――――――――――――――――  ●  ラバーバンド処理
//―――――――――――――――――――――――――――――――――――
case 4:       // ラバーバンド                                       !9
    if(!IsRectEmpty(&currentRect))
      pDC->Rectangle(currentRect.left,currentRect.top,
      currentRect.right,currentRect.bottom);                         !10
      if(point.x < m_bmx)                                            !11
```

関数 SetROP2(R2_NOTXORPEN)によって、描画モードを変更、XOR（排他的 OR）とする。

描画モード XOR で２回目の矩形図形を描く。２回この XOR で図形を描くと、元の状態に戻ることになる。

```
        {
          currentRect.left  = point.x;
           currentRect.right = m_bmx;
        }else{
           currentRect.left  = m_bmx;
           currentRect.right = point.x;
        }
        if(point.y < m_bmy)                                          !12
        {
           currentRect.top = point.y;
           currentRect.bottom = m_bmy;
        }else{
           currentRect.top = m_bmy;
           currentRect.bottom = point.y;
        }
        pDC->Rectangle(currentRect.left,currentRect.top,
        currentRect.right,currentRect.bottom);                       !13
        pDC->SetROP2(oldROPCode);                                    !14
        break;
         •
         •
        break;
default:
}
}
ReleaseDC (pDC);
CView::OnMouseMove(nFlags, point);
}
}
```

描画モード XOR で１回目の矩形図形を描く。矩形の大きさは、マウスのドラッグ位置で決定される。

描画モードを元の状態 oldROPCode に戻す。

関数 OnMouseMove()のラバーバンドに関連する部分について説明する。

１．変数 ButtonDown が TRUE の場合、次の処理を行い、ラバーバンドを描く処理を実行する。FALSE の場合は、この関数より抜けることになる。
２．アクティブなウインドウに関するポインターとウインドウのハンドルを取得する。
３．ラバーバンド処理以外で必要となるマウスの移動量を計算する。
４．同じく、前回のマウス位置 F_point を変数にセットする。これもラバーバンド処理以外で必要となる。
５．関数 SetROP2(R2_NOTXORPEN)によって、描画モードを変更、XOR（排他的 OR）とする。
６．今回のマウスの位置を次回の処理に備えて、F_point に保存する。
７．各処理のために初期設定を行う。
８．Switch 文の変数 F_hantei を用いて、処理内容を分類する。
９．ここから、実際のラバーバンドを描く処理となる。前回のラバーバ

ンドの四角 currentRect が空でない場合、次の処理を行う。
10. 前回の CurrentRect を用いて、四角を描く。この処理によって、前回描いた四角を消し去り、四角の線で消された背景が元通りとなる。
11. 今回のラバーバンドの四角を求め直す。最初、ｘ方向の左位置と右位置を設定する。
12. 次に、ｙ方向の上端と下端位置を設定する。
13. 新しく求め直した CurrentRect を用いて、四角を描く。
14. 関数 SetROP2(R2_NOTXORPEN)を用いて、描画モードを元の状態 oldROPCode に戻す。

ラバーバンド処理で重要なのは、次の２点である。一つはイベント駆動のメカニズムを利用している点、他は pDC->SetROP2(R2_NOTXORPEN)の関数によって四角を２重描きすることで、消された四角の線分下の背景が元通りとなる点である。これらを十分に理解することで、他のプログラムでもこの仕組みが利用可能となろう。

2.3 OnDraw()関数の特異な役割

ここでは、ウインドウ用のプログラムで、最も重要な関数 OnDraw()について解説する。多くのウインドウ用図形処理関数は、この OnDraw 関数から呼ばれる。

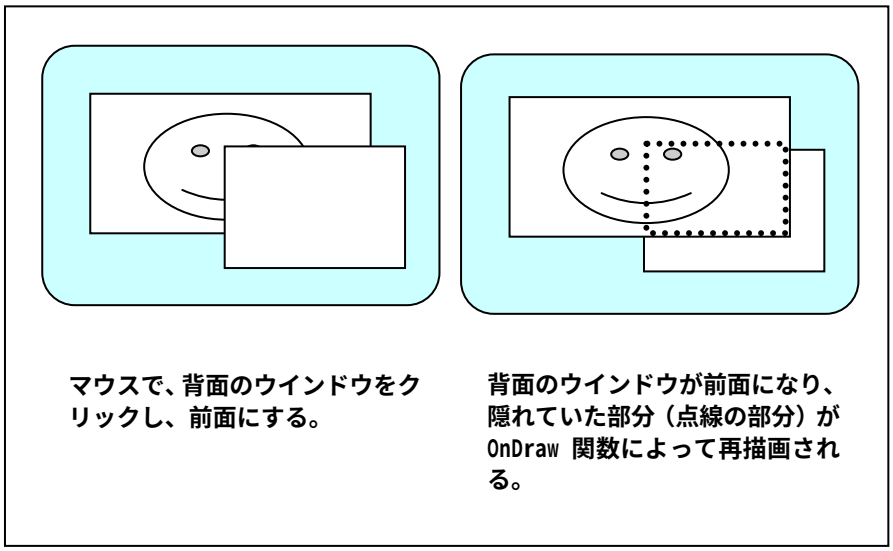

図 2-7 ウインドウ内で背面から前面に回った部分の再表示

ユーザーによって見えていない部分が見えるように操作される場合、OS が自動的にその部分を再描画しているように見える。しかし、実際はプログラムが手当てをしなければ何も表示されることはない。OS は再描画のメッセージを発するのみで、その部分を手当てするわけではない。

そのため、OS からのメッセージを受けて、関連するオブジェクの OnDraw 関数が実行され、その中で適切な描画関数をコールしなければならない。従って、OnDraw 関数は、どの時点においてもコールされることを意識して作成する必要がある。まず、動的プレゼンターの CSf31View クラスの中の OnDraw 関数を見てみよう。

```
//     ――――――――――――――――――――――――――――――――
//     ●    CSf31View クラスの子ウインドウの描画
//     ――――――――――――――――――――――――――――――――
//
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())
{
    m_x1_pr =0;
    m_x2_pr =0;
    m_y_pr  =0;
    m_han_pr =0;
switch (fno_xy){
case STRUCT:
    if( F_time_ii > m_nstep && fno_xxx != 2) F_time_ii = m_nstep;
    if(F_time_ii <= 0   ){
      if( fno_xxx == 0) pre_disp_mem_persp_pr(hwnd,pDC);
      if( fno_xxx == 2) disp3_mem_persp_pr(hwnd,pDC);
      if( fno_xxx == 3) disp7_mem_persp_pr(hwnd,pDC);
      if( fno_xxx != 3 && fno_xxx != 1 && fno_xxx != 0 && fno_xxx != 2)
        disp_mem_persp_pr(hwnd,pDC);
    }else{
      if( fno_xxx == 0) pre_disp_mem_persp_pr(hwnd,pDC);
      if( fno_xxx == 1 && F_time_ii <= m_nstep ) disp2_mem_persp_pr(hwnd,pDC);
      if( fno_xxx == 2) disp3_mem_persp_pr(hwnd,pDC);
      if( fno_xxx == 3) disp7_mem_persp_pr(hwnd,pDC);
      if( fno_xxx == 4 && F_time_ii <= m_nstep ) disp5_mem_persp_pr(hwnd,pDC);
      if( fno_xxx == 5 && F_time_ii <= m_nstep ) disp6_mem_persp_pr(hwnd,pDC);
    }
    break;
//     ――――――――――――――――――――――――――――――――
//     ●    CSf31View クラスの描画
//      fno_xxx : 0  option
//      fno_xxx : 1  structure
//      fno_xxx : 2  mode displacements
//      fno_xxx : 3  maximum stresses
//      fno_xxx : 4  maximum response of displacement, velocity and acceleration
```

```
//      fno_xxx : 5  mode decomposition
//     ――――――――――――――――――――――――――――――――
case WAVE:
    if(m_dat_struct == 0)  sf31wave.set_wave_pr(pDC,hwnd);
    if(m_dat_struct == 1)  sf31ftt.set_frame_pr(pDC,hwnd);
    if(m_dat_struct == 2)  sf31shear.set_frame_pr(pDC,hwnd);
    break;
case SLOAD:
    sf31load.set_load_frame_pr(pDC,hwnd);
    break;
case STRESS:
    set_member_frame_pr(pDC,hwnd);
    break;
case MODEWD:
    sf31mode.set_mode_frame_pr(pDC,hwnd);
    break;
case DISMAX:
    sf31mode.set_mode_frame_pr(pDC,hwnd);
    break;
case SECTION:
    set_section_frame_pr(pDC,hwnd,F_time_ii);
    break;
default:
    break;
}
//――――――――――――――――――――――――――――  ● 画面における描画
}else{
    CSf31Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    F_print_hen =0;
switch (fno_xy){
case STRUCT:
    if( F_time_ii > m_nstep && fno_xxx != 2 && fno_xxx != 0){
    if(m_nstep > 0)F_time_ii = m_nstep;
    }
    if(F_time_ii <= 0   ){
      disp_mem_persp(pDC);
    }else{
      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 == 3) disp_mem_persp(pDC);
      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(pDC,hwnd);
    if(m_dat_struct == 1)  sf31ftt.set_frame(pDC,hwnd);
    if(m_dat_struct == 2)  sf31shear.set_frame(pDC,hwnd);
    break;
case SLOAD:
    sf31load.set_load_frame(pDC,hwnd);
    break;
```

```
    case STRESS:
            set_member_frame(pDC);
        break;
    case MODEWD:
            sf31mode.set_mode_frame(pDC,hwnd);
        break;
    case DISMAX:
            sf31mode.set_mode_frame(pDC,hwnd);
        break;
    case SECTION:
            Section.set_frame(pDC, hwnd);
        break;
    default:
        break;
    }}
    }
```

子ウインドウを管理するクラスCSf31Viewに関するプログラムはひとつしかないが、実際は画面に表示されている子ウインドウの数だけオブジェクトが存在する。メッセージが届いたとき、まず、どのウインドウオブジェクトが対象となっているかを理解する必要がある。そこで、実際の処理に入る前に、ウインドウ管理システムを用いて、次のような手続きを踏む。最初に、このウインドウのハンドル hwnd を得る。このハンドルを用いて、ウインドウが既に消去されているかどうかチェックし、画面に表示されていない場合は、この関数を抜ける。次に、このハンドルを用いて、関数 GetWindowLong()で画面番号 fno_yx を取得する。画面番号から、関数 getwindx()と getwindy()を用いて、ウインドウコード fno_xy と構造図出力用コード fno_xxx を得る。ウインドウコード fno_xy が-1 のとき、つまり、そのウインドウに何も表示されていないとき、この関数を抜ける。

次に、if(pDC->IsPrinting())文によって、プリンターへの印刷処理と画面表示処理に分類される。このように OnDraw 関数では、画面への描画と共にプリンターへの出力コードも記述しておく必要がある。これについては第７章で述べる。ここでは、画面への描画部分を説明しよう。

まず、ウインドウコードをキィにして、else 文の後の switch 文によって、図形表示関数を選択する。ケース STRUCT では、構造透視図を描く関数を呼ぶ。この後は、図形処理用関数で図形を再描画することになる。ここでは、多くの仕様があるため、このように複雑に分類され、図形処理用関数が呼ばれる。ケース WAVE では、関数 sf31wave.set_wave()で、波形描画を行う。他に、ケース SLOAD、ケース STRESS、ケース MODEWD、ケース DISMAX、ケース SECTION がある。これらの図形処理の詳細については、マニュアルプレゼンターを参照されたい。

以上のように、プレゼンターではどのような図形処理を行い、子ウインドウに何を描画しているかを全て認識し、この Ondraw()関数に組み込んでおかなければならない。これを怠ると真っ白なウインドウが現れたり、異なった図形が表示されることになる。

2.4 図形の拡大・縮小、ユーザーインターフェイス

本節では、ユーザーとのインターフェイスを通じて、透視図を拡大・縮小、回転などを行う処理について解説する。これらのインターフェイスと処理は、やはり子ウインドウを管理するクラス CSf31View で実現している。ここでの処理は、先に説明したラバーバンド描画とほぼ同じで、同じイベント駆動のメカニズムを利用する。

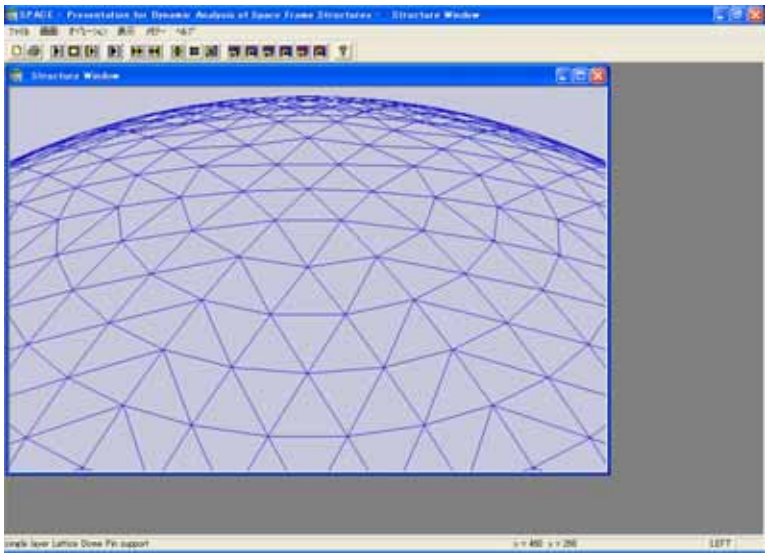

図 2-8 図形の拡大

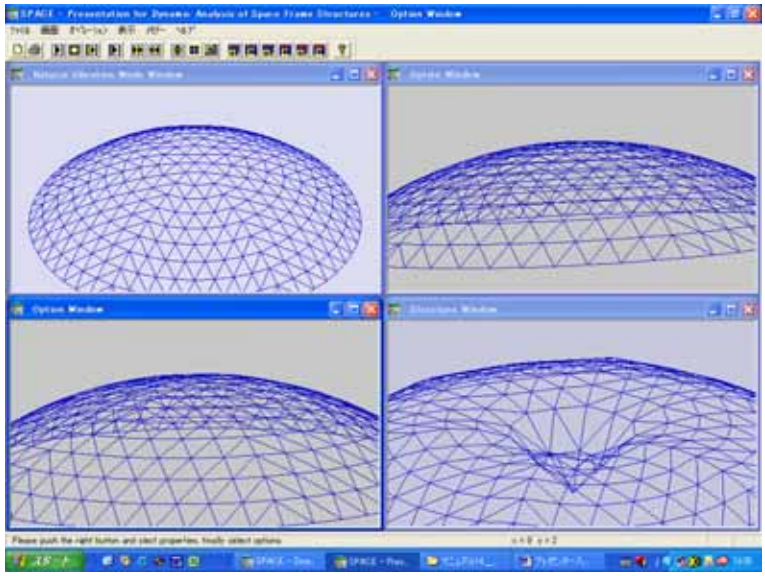

図 2-9 図形の回転

関連するメッセージマップと関数を以下に示す。解説する関数は、次の4つである。マウスによる操作は、クラス CSf31stView のメッセージマップで処理され、該当する関数に制御が移る。

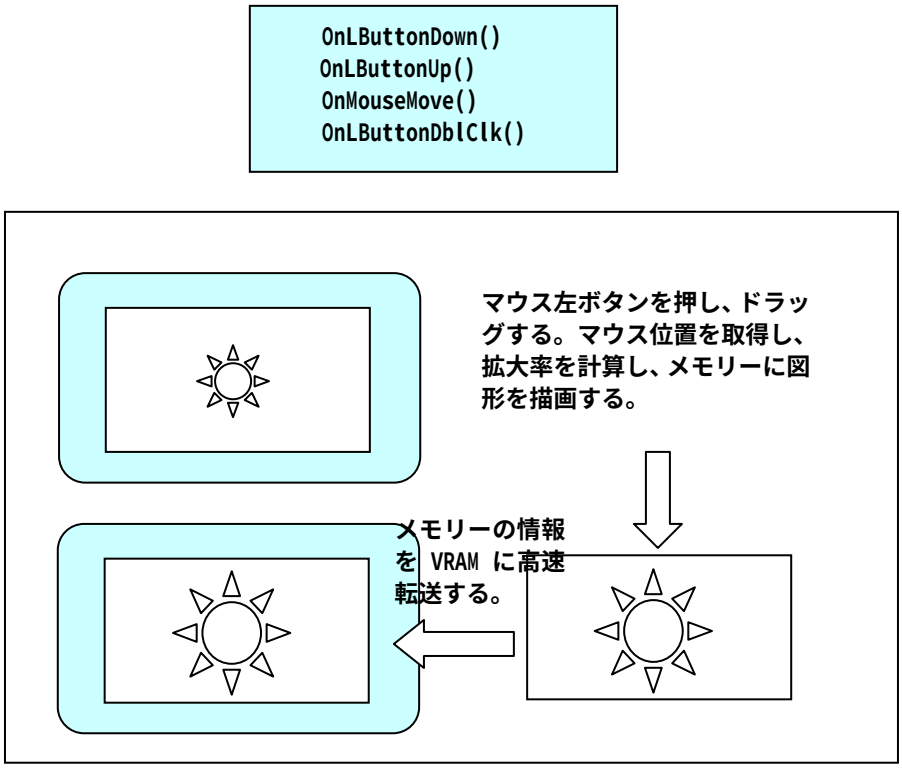

図 2-10 図形の拡大・縮小を行う仕組み

構造透視図が描かれているウインドウの中で、マウスの左ボタンを押し、そのままドラッグし、最後にマウスボタンを離すという一連の操作で、図形は拡大・縮小する。そのとき、マウスからのイベントメッセージがメッセージマップで処理され、該当する関数に制御が受け渡される。例えば、一連の操作で、関数は、OnLButtonDown()がコールされ、マウスがドラッグされている間は、OnMouseMove()がコールされる。マウスボタンの押下が解除されると、関数 OnLButtonUp()が呼ばれ、操作終了処理が行われる。それでは、ここで一連の処理を具体的なコードを用いて詳細に検討しよう。以下に関連するメンバー関数とメッセージマップを示す。ここでは、理解を容易とするために、プレゼンターのプログラムを簡略化して示す。

```
BEGIN_MESSAGE_MAP(CSf31View, CView)
        ON_WM_LBUTTONDOWN()                    // マウス左ボタンをダウン
        ON_WM_LBUTTONUP()                      // マウス左ボタンをアップ
        ON_WM_MOUSEMOVE()                      // マウス移動
        ON_WM_LBUTTONDBLCLK()                  // マウス左ボタンをダブルクリック
END_MESSAGE_MAP()
```

```
//    ―――――――――――――――――――――――――――――――――
//      ●　画面用の図形処理
//    ―――――――――――――――――――――――――――――――――
//    ―――――――――――――――――――――――――――――――――
//      ●　左ボタンダウン
//    ―――――――――――――――――――――――――――――――――

void CSf31View::OnLButtonDown(UINT nFlags, CPoint point)
{
//    ―――――――――――――――――――――――
//      ●　画面コードを取得
//    ―――――――――――――――――――――――
        F_hantei=-1;
        HWND hwnd = this->GetSafeHwnd();
        if(IsWindow(hwnd) == FALSE) return;
        int  fno_yx = GetWindowLong(hwnd,GWL_USERDATA);
        int fno_xy = getwindx(fno_yx);
        if(fno_xy == -1) return;
        fno_xx = (int) fno_yx;
        if(fno_xy != STRUCT ) return;
//    ―――――――――――――――――――――――
//      ●　拡大・縮小、回転を行なうための初期設定（構造）
//    ―――――――――――――――――――――――
        m_fno_xy = fno_xy;
        if(nFlags & MK_SHIFT) F_hantei =0;
        if(nFlags & MK_CONTROL) F_hantei =1;
        F_point = point;
        ButtonDown = TRUE;
        SetCapture();
        CView::OnLButtonDown(nFlags, point);
}
//    ―――――――――――――――――――――――――――――――――
//      ●　左ボタンダブルクリック（図形の移動処理）
//    ―――――――――――――――――――――――――――――――――
void CSf31View::OnLButtonDblClk(UINT nFlags, CPoint point)
{
        F_hantei=-1;
        HWND hwnd = this->GetSafeHwnd();
        if(IsWindow(hwnd) == FALSE) return;
        int  fno_yx = GetWindowLong(hwnd,GWL_USERDATA);
        int fno_xy = getwindx(fno_yx);
        if(fno_xy == -1) return;
        fno_xx = (int) fno_yx;
        if(fno_xy != STRUCT ) return;
        m_fno_xy = fno_xy;
        if(nFlags & MK_SHIFT) F_hantei =2;
        if(nFlags & MK_CONTROL) F_hantei =3;
        F_point = point;
        ButtonDown = TRUE;
        SetCapture();
        CView::OnLButtonDblClk(nFlags, point);
}
//    ―――――――――――――――――――――――――――――――――
//      ●　マウス移動
```

左ボタンをクリックするとこの関数が呼ばれ、処理の初期設定を行う。第１引数は、マウスの左ボタンを押すと同時に、他のキーを押したかどうか示す。第２引数はマウスの位置情報を示す。

ウインドウのハンドルを取得し、そのハンドルで関数 GetWindowLong から画面番号 fno_yx を求める。この画面番号から、画面コードを取得し、これが透視図用画面 STRUCT でない場合は、この関数から抜ける。

図形の原点移動を行うためのオプション F_hantei を決定する。
シフトキーと同時に押すと F_hantei を０に、コントロールキィと同時に押すと F_hantei を１にする。

左ボタンをダブルクリックするとこの関数が呼ばれる。第１引数は、マウスの左ボタンを押すと同時に、他のキーを押したかどうか示す。第２引数はマウスの位置情報を示す。

図形の原点移動を行うためのオプション F_hantei を決定する。
シフトキィと同時に押すと F_hantei を２に、コントロールキィと同時に押すと F_hantei を３にする。

```
//    ―――――――――――――――――――――――――――――――――
void CSf31View::OnMouseMove(UINT nFlags, CPoint point)
{
//    ―――――――――――――――――――――――
//      ●　ウインドウコードの取得と初期設定
//    ―――――――――――――――――――――――
         if(ButtonDown){
         char sTimeout[50];
         CDC* pDC = GetDC();
         HWND hwnd = this->GetSafeHwnd();
          CPoint Dpoint =   F_point - point;
         int F_movedxy  =  Dpoint.x   + Dpoint.y ;
          F_point = point;
        int kkk;
         float a_Zoomy;
        int ii= fno_xx-1;
//    ―――――――――――――――――――――――
//      ●　拡大・縮小、回転処理
//    ―――――――――――――――――――――――
switch (F_hantei){
//    ―――――――――――――――――――――――
//      ●　回転データセット（x、y軸）
//    ―――――――――――――――――――――――
case 0:
        a_Zoomy= (float)Dpoint.x* 0.01 ;
        a_Zoom = (float)Dpoint.y* 0.01;
        kkk=1;
        RRSETE(&kkk,&a_Zoomy,&a_Zoom,&m_viewpsx[0]);
        break;
//    ―――――――――――――――――――――――
//      ●　回転データセット（ｚ軸）
//    ―――――――――――――――――――――――
case 1:
        a_Zoom = (float)F_movedxy* 0.02;
        kkk=2;
        RRSETE(&kkk,&a_Zoom,&a_Zoomy,&m_viewpsx[0]);
            break;
//    ―――――――――――――――――――――――
//      ●構造図の原点移動（X－Z）
//    ―――――――――――――――――――――――
case 2:
        m_base[0] = m_base[0] + (float)Dpoint.x* 10 ;
        m_base[2] = m_base[2] - (float)Dpoint.y* 10;
        break;
//    ―――――――――――――――――――――――
//      ●　構造図の原点移動（Y－Z）
//    ―――――――――――――――――――――――
case 3:
        m_base[1] = m_base[1]+ (float)Dpoint.x* 10;
        m_base[2] = m_base[2]- (float)Dpoint.y* 10;
        break;
//    ―――――――――――――――――――――――
//      ●　拡大・縮小データのセット
//    ―――――――――――――――――――――――
```

マウスの移動処理を行う。引数の point によって、マウスの現在位置が時々刻々と得られる。

ButtonDown 変数が FALSE の場合、この関数から抜ける。TRUE の場合、以下の処理を行う。まず、ウインドウのハンドルを取得し、次に、マウスの移動量を求める。

図形の回転処理を行うため、ｘ、ｙ軸の回転角度をマウスの移動距離によって決定よって決定し、視点位置とスケールを求め直す。

図形の回転処理を行うため、ｚ軸の回転角度をマウスの移動距離によって決定し、視点位置とスケールを求め直す。

マウスの移動量から、図形の原点移動（x-z）用パラメータ(m_base[])をセットする。

マウスの移動量から、図形の原点移動（y-z）用パラメータ(m_base[])をセットする。

図形の拡大・縮小データをマウスの移動量から計算し、透視図用の変換行列を作成する。

```
default:
      a_Zoom = (float)F_movedxy * 0.002*FF_scalps  ;
      kkk=4;
      RRSET(&kkk,&a_Zoom,&m_viewpsx[0],&m_scalpsx);
      break;
}
//    ―――――――――――――――――――――
//     ●  回転計算を行なった後、図形出力
//    ―――――――――――――――――――――
       ROTSET(&m_scrnpsx[0],&F_viewps[0],&m_scalpsx,&m_rr[0][0],&m_viewpsx[0]);
       disp_mem_persp(pDC);                          //  図形表示
       ReleaseDC (pDC);
}
       CView::OnMouseMove(nFlags, point);
}
//    ―――――――――――――――――――――――――――――――――
//     ●   左ボタンアップ
//    ―――――――――――――――――――――――――――――――――
void CSf31View::OnLButtonUp(UINT nFlags, CPoint point)
{
//    ―――――――――――――――――――――
//     ●  拡大・縮小、回転処理を解除
//    ―――――――――――――――――――――
        ButtonDown = FALSE;
        ReleaseCapture();
        CView::OnLButtonUp(nFlags, point);
}
```

図形の回転行列を計算した後、構造の透視図を描く。

左ボタンアップの処理を行う。ここでは、OnMouseMove 関数で必要となる変数 ButtonDown を FALSE にセットする。関数 ReleaseCapture を実行することで、マウスからのメッセージが他のウインドウへ送ることができる。

最初に、メッセージマップに関する記述がある。関連するメンバー関数は４つであり、いずれもユーザーがマウス操作をすることで、メッセージが出され、それをこの４つの関数が受け取ることになる。メッセージを受け取ったこれらの関数は、_WM_を取り除いた関数名の関数に制御を移す。

次に、上記の関数について、簡単に説明する。これらの処理は非常に簡単なので容易に理解できよう。まず、左ボタンをダウンすると、メッセージマップの中で処理され、関数 OnLButtonDown()がコールされる。最初に、重要な役割を担う F_hantei コードについて説明する。F_hantei コードは動作の制御に用い、ここでは次の仕様となっている。

```
    マウスによる動作コード（F_hantei）
 -1: 図形の拡大･縮小
  0: ｘ軸、ｙ軸に関する図形の回転
  1: ｚ軸に関する図形の回転
  2: ショートカットメニューの表示（右ボタン）
  2: 原点移動（全体座標 X-Z 軸の原点移動）
  3: 原点移動（全体座標 Y-Z 軸の原点移動）
```

左ボタンダウン OnLButtonDown()について説明する。この関数の引数

は、nFlags と point であり、前者は、ボタンを押したときのキィの状態を表し、後者はマウスの位置を示す。関数内部では、まず、動作を決める動作コード F_hantei に-1 を設定する。次に、このウインドウの画面番号とウインドウコードを取得する。ウインドウコードが構造透視図 STRUCT でない場合は、この関数から抜け出る。ここでは、構造図の拡大・縮小処理を行うためである。次に、引数である nFlags を参照して、キィ入力の状態で動作を選択する。シフトキィを押して左ボタンを押すと、動作コードを F_hantei = 0 とし、コントロールキィを押して左ボタンを押すと、動作コードを F_hantei = 1 とする。この動作コードの設定によって、前者はｘ軸とｙ軸に関する図形の回転を、後者はｚ軸に関する図形の回転を行う。また、-1 の場合は、図形の拡大・縮小を表す。マウスの位置を F_point にセットし、操作が継続していることを示す ButtonDown パラメータを TRUE にする。さらに、SetCapture()関数を実行し、これ以後マウスが当該ウインドウを外れても OnMouseMove()関数が実行されることを保証する。最後に、Cview クラスの同関数に制御を渡す。

次の関数は、OnLButtonDblClk()であり、左ボタンをダブルクリックした場合に、この関数が呼ばれる。その後、押したままドラッグすると図形原点が移動する。前記の関数と同様にそのウインドウのハンドル、画面番号、ウインドウコードを取得する。次に、シフトキィが同時に押されていると動作コードを F_hantei = 2 に、同じくコントロールキィの場合は F_hantei = 3 に設定する。後は、マウスの位置を F_point にセットし、ButtonDown パラメータを TRUE にする。さらに、SetCapture()関数を実行する。

次の関数は、OnMouseMove()であり、実際の動作を行う関数である。最初に、ButtonDown パラメータが TRUE になっているかどうかチェックする。このパラメータが FALSE の場合はこの関数は何も処理せず、戻ることになる。関数の内部では、最初に、このウインドウのデバイスコンテキストとウインドウハンドルを取得する。次に、操作処理を行うために、マウスの移動量を測定して、次のように移動用パラメータに割り付ける。前回のマウス位置 F_point と今回の位置 point の差をとり、Dpoint にセットする。移動量 F_movedxy をセットし、次の処理のために保存する。また、次回のためにマウス位置を F_point にセットする。

動作コード F_hantei をパラメータとして、switch 文を用いて制御を分類する。ケース 0 は、ｘ軸とｙ軸に関する図形の回転であり、回転量は、マウス移動量 F_movedxy よりセットする。その後、サブルーチン

RRSETE()で回転行列作成用パラメータm_viewpsxを求める。引数kkkは、回転行列作成用パラメータの作成種別コードである。ケース１は、ｚ軸に関する図形の回転であり、サブルーチンRRSETE()で回転行列作成用のパラメータm_viewpsxを求める。ケース２は全体座標X-Z軸に対する原点移動であり、原点を設定するm_baseに移動量をセットする。同様に、ケース３は全体座標Y-Z軸に対する原点移動であり、原点を設定するm_baseに移動量をセットする。最後に、動作コードF_hanteiが-1の場合で、これはdefault処理となり、ここでは図形の拡大･縮小を行う。

マウスの移動データが、ドラッグする毎に送られ、その都度、図形が再描画される。このため、図形の滑らかな動きが実現できている。

全ての操作を処理するパラメータが設定された後、サブルーチンROTSET()を用いて回転行列を求める。この新しい回転行列を使用して、関数disp_mem_persp()を用いて新しい構造透視図を表示する。処理が終了した後、クラスCviewのOnMouseMove関数に制御を渡す。

最後に、メンバー関数OnLButtonUp()について説明する。この関数は、一連のマウスによる図形描画の終了処理を行う。最初に、マウスの操作継続を終了させるために、操作が継続していること示すButtonDownパラメータをFALSEにする。さらに、現在操作対象を固定しているウインドウを、関数ReleaseCapture()を用いて、操作対象から解放する。

上記の操作処理には、視点やスケールの変更、回転行列の作成などの処理が必要となる。SPACEのプレゼンターシステムでは、これらの処理を行うサブルーチンをFORTRANで記述している。内容は、非常に単純なので、コメント行を見るのみで、説明がなくても理解できよう。以下に関連するサブルーチンを示す。

```
C     ―――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /rrset
C       ―――――――――――――――――――――――――――――――
C     ●   視点・スケールの変更
C     ―――――――――――――――――――――――――――――――
c―――――――――――――――――――――――――――――――――
c  in        k         :  設定番号
c                      :  1;gxs   2;gys   3;gzs   4;sc
c            sc        :  スケール
c            gxs       :  視点位置（制御用）  x方向
c            gys       :                      y方向
c            gzs       :                      z方向
c            scc       : スケールの入力値（変更不可）
c―――――――――――――――――――――――――――――――――
        subroutine rrset(k,pp,gxs,sc)
        dimension gxs(3)
        if(k.eq.1) then
        gxs(1)=pp
        elseif(k.eq.2) then
        gxs(2)=pp
```

このサブルーチンでは、透視図における視点位置とスケールの更新を行う。

```
          elseif(k.eq.3) then
          gxs(3)=pp
          elseif(k.eq.4) then
          sc=sc-pp
          if(sc.le.0.000001) sc=sc+pp
          endif
          return
          end
C     ―――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /rrsete
C       ―――――――――――――――――――――――――――――――――
C     ●    視点・スケールの変更その２
C     ―――――――――――――――――――――――――――――――――
          subroutine rrsete(k,pp,pp1,gxs)
          dimension gxs(3)
          if(k.eq.1) then
          gxs(1)=gxs(1)+pp
          gxs(2)=gxs(2)-pp1
          elseif(k.eq.2) then
          gxs(3)=gxs(3)-pp
          endif
         return
          end
C     ―――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /rotset
C       ―――――――――――――――――――――――――――――――――
C     ●   回転行列のセット
C     ―――――――――――――――――――――――――――――――――
c―――――――――――――――――――――――――――――――――――
c  in       sc        :  スケール
c           gll       :  視点からスクリーンのまでの距離（制御用）
c           gxs       :  視点位置（制御用）
c           gps       :  視点方向中心位置
c  out      rr(4,4)   :  回転行列
c―――――――――――――――――――――――――――――――――――
      subroutine rotset(gxs,gps,sc,rr,thxx)
      dimension rx(4,4),ry(4,4),rz(4,4),rr(4,4)
      dimension gxs(3),gps(3),thxx(3)
      data pai/3.1415926/
      call  KAIT(TXSB,TYSB,TZSB,gps,gxs,gll)              ! 視点からの角度と距離を計算
      tzsb=tzsb + pai*0.5
      TXSB=thxx(1)+txsb                                   ! 増分角度を足しこむ
      TYSB=thxx(2)+tysb
      TZSB=thxx(3)+tzsb
      call rtx(TYSB,rx)                                   ! x 軸回転行列作成
      call rty(TXSB,ry)                                   ! y 軸回転行列作成
      call rtz(TZSB,rz)                                   ! z 軸回転行列作成
      call rotz(rx,ry,rz,rr,sc,gll)                       ! 透視変換行列作成
      return
      end
C     ―――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /kait
C       ―――――――――――――――――――――――――――――――――
C     ●
```

ｘ軸、ｙ軸、ｚ軸に関する各々の回転行列を作成し、それらの行列積を取ることで、全体の回転行列 rr を得る。

```
C     ―――――――――――――――――――――――――――――――――――――
      subroutine kait(txsb,tysb,tzsb,ps,pe,rl)
      parameter (PAI=3.14159265)
      dimension ps(3),pe(3)
      txsb=0.
      tysb=0.
      tzsb=0.
      R1=ps(1)-pe(1)
      R2=ps(2)-pe(2)
      R3=ps(3)-pe(3)
      RL=(R1**2+R2**2+R3**2)
      if(RL.eq.0.) return
      RL=SQRT(RL)
      RL2=(R1**2+R2**2)
      R12=SQRT(RL2)
      TXSB=0.0
      if(R2.EQ.0.0) goto 80
      TZSB=(-PAI/2.0-(ATAN(R1/R2)))
      if(R2.GT.0.0) TZSB=(PAI/2.-(ATAN(R1/R2)))
   90 TYSB=(ATAN(R3/R12))
      goto 50
   80 TZSB=0.0
      if(R1.LT.0.0) TZSB=pai
      if(R1.NE.0.0) goto 90
      TYSB=-pai*0.5
      if(R3.GT.0.0) TYSB=pai*0.5
 50   continue
      return
      end
C     ―――――――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /rotz
C       ―――――――――――――――――――――――――――――――――――――
C     ●    透視回転行列の掛け算
C     ―――――――――――――――――――――――――――――――――――――
c―――――――――――――――――――――――――――――――――――――
c  in       rx(4,4)   :ｘ軸回転行列
c           ry(4,4)   :ｙ軸回転行列
c           rz(4,4)   :ｚ軸回転行列
c           sc        :スケール
c           gll       :視点からスクリーンのまでの位置（制御用）
c           dm1       :ダミー行列
c  out      rr(4,4)   :回転行列
c―――――――――――――――――――――――――――――――――――――
      subroutine rotz(rx,ry,rz,rr,sc,gll)
      dimension rx(4,4),ry(4,4),rz(4,4),rr(4,4),dm1(4,4)
      real*8 sum
      do 10 i=1,3
      do 10 j=1,3
      sum=0.
      do 12 k=1,3
      sum=sum+rz(i,k)*ry(k,j)
 12   continue
      dm1(i,j)=sum
 10   continue
```

構造物の全体座標系における、視点位置からスクリーン原点に向かうベクトルのｘ軸、ｙ軸、ｚ軸に関する回転角度及び、このベクトルの長さを計算する。

３軸の回転行列とスケール、視点からスクリーンまでの距離を用いて、透視図変換行列を作成する。

```
      cosθx do 20 i=1,3
       do 20 j=1,3
      sum=0.
      do 22 k=1,3
       sum=sum+dm1(i,k)*rx(k,j)
 22   continue
      rr(i,j)=sum
 20   continue
      glx=100000.
      if(abs(gll) .ge. 0.00001) glx=1./gll
      do 40 i=1,3
      rr(i,4)=rr(i,2)*glx
 40   continue
      rr(4,4)=glx+1./sc
      rr(4,3)=sc
      return
      end
C
―――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /rtx
C       ―――――――――――――――――――――――――――――――
C     ●    ｘ軸回転行列作成
C     ―――――――――――――――――――――――――――――――
c――――――――――――――――――――――――――――――――
c  in       th        :回転角度（ラジアン）
c  out      rt(4,4)   :回転行列
c――――――――――――――――――――――――――――――――
      subroutine rtx(th,rt)
      dimension rt(4,4)
      cth=cos(th)
      sth=sin(th)
      rt(1,1)=1.
       rt(1,2)=0.
      rt(1,3)=0.
      rt(2,1)=0.
      rt(3,1)=0.
      rt(2,2)=cth
      rt(3,3)=cth
      rt(3,2)=-sth
      rt(2,3)=sth
      return
      end
C     ―――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /rty
C       ―――――――――――――――――――――――――――――――
C     ●    ｙ軸回転行列作成
C     ―――――――――――――――――――――――――――――――
c――――――――――――――――――――――――――――――――
c  in       th        :回転角度（ラジアン）
c  out      rt(4,4)   :回転行列
c――――――――――――――――――――――――――――――――
      subroutine rty(th,rt)
      dimension rt(4,4)
      cth=cos(th)
```

回転行列と透視変換行列の掛け算を行い、次の透視変換行列を求める。

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & g \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & gr_{12} \\ r_{21} & r_{22} & r_{23} & gr_{22} \\ r_{31} & r_{32} & r_{33} & gr_{32} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ここでは、図形の大きさを変更するために、透視変換公式を少し変えて使用する。

ｘ軸の回転行列を作成する。

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & \sin\theta_x \\ 0 & -\sin\theta_x & \cos\theta_x \end{bmatrix}$$

```
      sth=sin(th)
      rt(1,2)=0.
       rt(2,1)=0.
      rt(2,2)=1.
      rt(2,3)=0.
      rt(3,2)=0.
      rt(1,1)=cth
      rt(3,3)=cth
      rt(3,1)=-sth
      rt(1,3)=sth
      return
      end
C     ―――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /rtz
C       ―――――――――――――――――――――――――――――――
C     ●    z軸回転行列作成
C     ―――――――――――――――――――――――――――――――――
c――――――――――――――――――――――――――――――――――
c  in       th        :回転角度（ラジアン）
c  out      rt(4,4)   :回転行列
c――――――――――――――――――――――――――――――――――
      subroutine rtz(th,rt)
      dimension rt(4,4)
      cth=cos(th)
      sth=sin(th)
      rt(1,3)=0.
      rt(2,3)=0.
      rt(3,1)=0.
      rt(3,2)=0.
      rt(3,3)=1.
      rt(2,2)=cth
      rt(1,1)=cth
      rt(2,1)=-sth
      rt(1,2)=sth
      return
      end
```

y軸の回転行列を作成する。

$$\begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix}$$

z軸の回転行列を作成する。

$$\begin{bmatrix} \cos\theta_z & \sin\theta_z & 0 \\ -\sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2.5 マウスによる位置取得

ここでは、マウスのクリックによって、図形内の位置を取得する方法について学ぶ。透視図や平面図・立面図において、画面からその位置を指定したい場合がある。マウスをその位置に移動し、クリックすることでシステムに位置を指定するわけであるが、実際はウインドウ上の座標値が得られるのみで、図面の節点や部材を直接指定できるわけではない。ウインドウ座標値を構造データの座標に変換するか、もしくは、構造データを画面座標に変換した情報を利用するか、どちらを選択して節点座標とマウス位置座標を付き合わせる必要がある。透視図におけるマウス

位置座標には、奥行き情報がないので、後者の方法を使用する。

ここでは、プレゼンターの透視図を表示しているウインドウ内で、マウスをクリックして位置を指定し、図 2-12 に示すダイアログで節点の状態を示す処理を例題として選び、位置取得の方法を学ぶ。

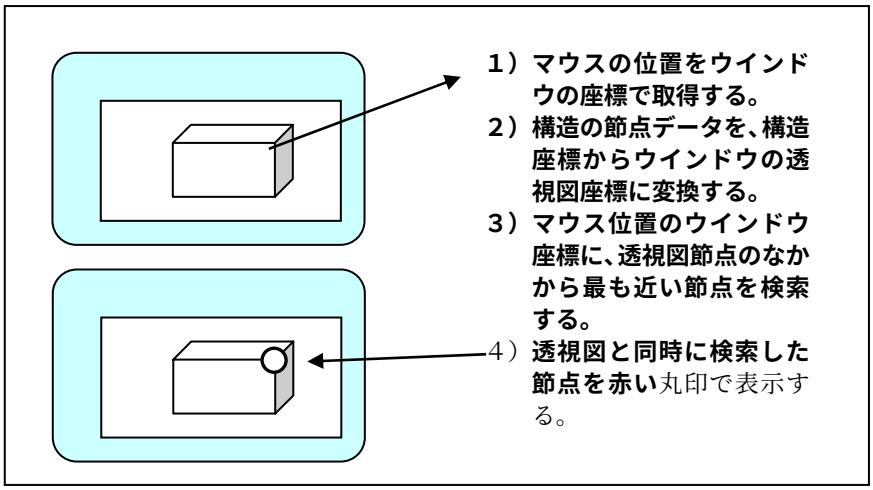

図 2-11 マウスで位置指定した後、図形の位置を求める

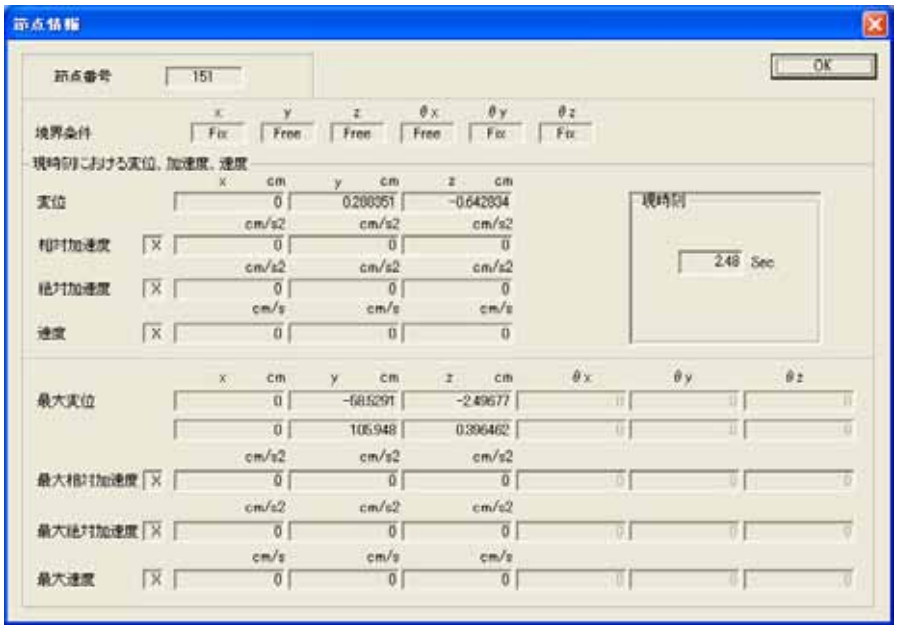

図 2-12 マウスで位置指定した後、ダイアログで節点変位の状態を示す

まず、右ボタンダウン処理を表す関数を次に示す。ここで、シフトキィと同時に押すと処理オプション F_hantei を 0 に、同様に、コントロールキィと同時に押すと処理オプション F_hantei を 1 にする。シフトキィを押した場合は部材に関する情報を、コントロールキィを押した場合は節点に関する情報をダイアログで提示する。

```
//      ――――――――――――――――――――――――――――――――
//      ●    OnRButtonDown 右ボタンダウン処理
//      ――――――――――――――――――――――――――――――――
```

```
//
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;
//    ――――――――――――――――――――――――――――――――
//      ●   動作を選択する変数 F_hantei を設定
//    ――――――――――――――――――――――――――――――――
          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){
//    ――――――――――――――――――――――――――――――――
//      ●   部材に関する検索と表示
//    ――――――――――――――――――――――――――――――――
case 0:
        if(fno_xy != 1 ) return;
        pikpoint(point,F_hantei);
        break;
//    ――――――――――――――――――――――――――――――――
//      ●   節点に関する検索と表示
//    ――――――――――――――――――――――――――――――――
case 1:
        if(fno_xy != 1 ) return;
        pikpoint(point,F_hantei);
        break;
case 3:
        pikpoint_mem(point);
        break;
case 4:
        pikpoint_nod(point);
        break;
case 5:
        delpoint_mem(point);
        break;
case 6:
        delpoint_nod(point);
        break;
case 7:
        stop_input_memory();
        break;
case 8:
```

処理オプション F_hantei を決定する。節点と部材の情報を提示するため、シフトキィと同時に押すと F_hantei を０に、コントロールキィと同時に押すと F_hantei を１にする。

処理オプション F_hantei が０の場合部材に関する情報を提示する。この画面が構造透視図でない場合はこの関数から抜ける。

処理オプション F_hantei が１の場合、節点に関する情報を提示する。この画面が構造透視図でない場合はこの関数から抜ける。

```
        Section.pikpoint_sec(pDC,point);
        break;
case 9:
        Section.delpoint_sec(pDC,point);
        break;
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);
}
```

次に、実際にマウスクリック位置を用いて、透視図の中から最も近い節点を選び出す関数 pikpoint()について説明する。まず、この関数を示す。

```
//    ――――――――――――――――――――――――――――――――
//    ●    pikpoint：マウスによる透視図の位置（節点番号）取得
//    ――――――――――――――――――――――――――――――――
//
void CSf31View::pikpoint(CPoint point,int ihantei)
{
//   ――――――――――――――――――――――――――――――――
//    ●    構造データを透視データに変換するための準備
//   ――――――――――――――――――――――――――――――――
        CDC* pDC = GetDC();
        CDC* pMemDC = new CDC;
        HWND hWnd=WindowFromDC(pDC->m_hDC);
        CRect rcFrame;
        ::GetClientRect(hWnd ,&rcFrame);
        pMemDC->CreateCompatibleDC(pDC);
        CBitmap *pMemBitmap  = new CBitmap;
        pMemBitmap -> CreateCompatibleBitmap(pDC,rcFrame.right,rcFrame.bottom);
        CBitmap *poldBitmap =(CBitmap*)pMemDC -> SelectObject(pMemBitmap);
        int iww,iwh;
        iww = rcFrame.right;
        iwh = rcFrame.bottom;
//    ――――――――――――――――――――――――――――――――
//    ●    ウインドウの大きさを設定
//    ――――――――――――――――――――――――――――――――
        F_pos[0][0]=0;
```

ウインドウと同じメモリー領域を設定する。

```
        F_pos[1][0]=0;
        F_pos[0][1]=iww;
        F_pos[1][1]=iwh;
        float pk[2];
        pk[0]=point.x;
        pk[1]=point.y;
        int mempik,pointpik;
//      ―――――――――――――――――――――――――――――――
//      ●　メモリー領域を背景色で塗りつぶす
//      ―――――――――――――――――――――――――――――――
        disp_frame_w(pMemDC);
//      ―――――――――――――――――――――――――――――――
//      ●　透視変換を行うと同時に、指定した位置を検索
//      ―――――――――――――――――――――――――――――――
        TOSHPK(posit,&node,&m_rr[0][0], F_bz,&F_pos[0][0],&F_amx,&F_amy,
                iconsb,&mnbsb,imeme_line,&pk[0],&mempik,&pointpik,&m_base[0],
                                m_pst_line,imeme_line,&m_pst_options);
//      ―――――――――――――――――――――――――――――――
//      ●　透視図と指定した節点もしくは、部材を赤で表示
//      ―――――――――――――――――――――――――――――――
          disp_graph_3(pMemDC,mempik,pointpik,ihantei);
//      ―――――――――――――――――――――――――――――――
//      ●　情報を VRAM に高速転送
//      ―――――――――――――――――――――――――――――――
        pDC->BitBlt(0,0,iww,iwh,pMemDC,0,0,SRCCOPY);
//      ―――――――――――――――――――――――――――――――
//      ●　リソースを解放
//      ―――――――――――――――――――――――――――――――
        delete pMemDC->SelectObject(poldBitmap);
        delete pMemDC;
        ReleaseDC (pDC);
//      ―――――――――――――――――――――――――――――――
//      ●　ダイアログで情報を表示
//      ―――――――――――――――――――――――――――――――
        char buffer[100];
        if(ihantei == 0){
          if(F_read_spring == 0){
          sprintf(buffer," Number of Member : %d ",mempik);
          MessageBox(buffer);
          }
          if(F_read_spring == 1){
//      ―――――――――――――――――――――――――――――――
//      ●　部材情報を表示
//      ―――――――――――――――――――――――――――――――
            mem_pik.datset(mempik,m_nstep,F_time_ii,F_Time);
            mem_pik.DoModal();
          }
        }else{
         if(F_read_disp == 0){
            sprintf(buffer," Number of Node ：%d  ",pointpik);
            MessageBox(buffer);
          }
        if(F_read_disp == 1){
//      ―――――――――――――――――――――――――――――――
```

ウインドウ領域を配列に設定し、これを利用してメモリー領域を関数 disp_frame_w を用いて背景を描く。

関数 TOSHPK()では、構造を透視変換すると共に、マウスがクリックした位置 posit を用いて、最も近い節点を検索する。

メモリー領域に透視図を表示すると共に、検索した節点を赤い丸印で表示する。

メモリー領域のデータを VRAM に高速転送し、画面に透視図を表示する。

処理オプション ihantei が０の場合部材情報を表示する。この時、部材応力データが読み込まれていない場合は、簡単な出力となる。読み込まれている場合は、関数 mem_pik.datset を用いてダイアログ表示する。

処理オプション ihantei が１の場合節点情報を表示する。この時、節点変位データが読み込まれていない場合は、簡単な出力となる。読み込まれている場合は、nod_pik.datset 関数を用いてダイアログ表示する。

```
//     ●   節点情報を表示
//     ―――――――――――――――――――――――――――――――――
           nod_pik.datset(pointpik,m_nstep,F_time_ii,F_Time);
           nod_pik.DoModal();
      }
   }
}
```

次に、マウスのクリック位置から実際に構造データの節点を求める処理について説明する。ここでのコードは、FORTRAN を用いている。

```
C     ―――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE / toshpk
C     ―――――――――――――――――――――――――――――――――
C     ●   マウスで指定した位置より、透視図から節点番号を取り出す
C     ―――――――――――――――――――――――――――――――――
      subroutine toshpk(bzz,iset,rr,gxy,
     *                  idpos,amx,amy,ij,memb,jic,pk,mempk,iptpk,pxx,
     *                  ipmem,itypm,ipkmem)

      dimension bzz(3,iset),igpn(iset),ij(2,memb),jic(memb)
      dimension pk(2),ipp(2,3),pxx(3),itypm(*),ipmem(*)
      dimension idpos(2,2),gxy(2,iset),rr(4,4)

      BX=(idpos(1,1)+idpos(2,1))*0.5+amx                              !1
      BY=(idpos(1,2)+idpos(2,2))*0.5+amy
C     ―――――――――――――――――――――――――――――――――
C     ●   マウスで指定した位置より、透視図から節点番号を取り出す
C     ―――――――――――――――――――――――――――――――――
       call dispgr(bzz,iset,rr,gxy,Bx,By,pxx)                         !2
C     ―――――――――――――――――――――――――――――――――
C     ●   節点と部材の検索
C     ―――――――――――――――――――――――――――――――――
       xd=1000000.                                                    !3
       iptpk=1
       if(ipkmem.eq.0) then                                           !4
       do 10 i=1,iset                                                 !5
       xdd=(pk(1)-gxy(1,i))**2+(pk(2)-gxy(2,i))**2                    !6
       if(xd.gt.xdd)then                                              !7
       iptpk=i
       xd=xdd
       endif
   10 continue
      xd=1000000.                                                    !8
       mempk=1
      do 20 i=1,memb                                                 !9
       i1=ij(1,i)                                                    !10
       i2=ij(2,i)
       xdd=(pk(1)-(gxy(1,i1)+gxy(1,i2))*0.5)**2+                     !11
     *    (pk(2)-(gxy(2,i1)+gxy(2,i2))*0.5)**2
       if(xd.gt.xdd)then                                             !12
       mempk=i
       xd=xdd
```

```
        endif
   20 continue                                                    !13
C     ―――――――――――――――――――――――――――――――――
C     ●　部材の検索（表示パラメータが指定されている場合）
C     ―――――――――――――――――――――――――――――――――
        else                                                      !14
        iptpk=1
       xd=1000000.
        mempk=1
       do 30 i=1,memb                                             !15
        itp=itypm(i)                                              !16
        if(ipmem(itp).eq.0) goto 30                               !17
        i1=ij(1,i)
        i2=ij(2,i)
        xdd=(pk(1)-(gxy(1,i1)+gxy(1,i2))*0.5)**2+
     *    (pk(2)-(gxy(2,i1)+gxy(2,i2))*0.5)**2
        if(xd.gt.xdd)then
        mempk=i
        xd=xdd
        endif
   30 continue
        i1=ij(1,mempk)
        i2=ij(2,mempk)
        iptpk=i1
        xdd=(pk(1)-gxy(1,i1))**2+(pk(2)-gxy(2,i1))**2
        xd= (pk(1)-gxy(1,i2))**2+(pk(2)-gxy(2,i2))**2
        if(xdd.gt.xd)iptpk=i2
        endif
       return
       end
```

マウスで指定した位置の節点あるいは部材を検索する場合、マウス位置が２次元の透視画面で設定しているため、構造物の３次元位置を直接検索することはできない。そこで、ここでは、透視図画面上で節点位置とマウスのクリック位置が最も近い節点や部材を検索することになる。ただし、奥行きの情報が抜けているため、得られた結果はユーザーが意図した節点や部材でない場合がある。この場合は、視点位置を変えることによって求めることが可能であるとして、この方法を用いることにする。以下にコメント番号に従って、プログラムの説明を行う。

１．透視図の中心位置を設定する。変数 amx、amy は図形中心位置を移動するパラメータである。
２．サブルーチン dispgr()を用いて、構造データを透視変換する。このサブルーチンについては、第４章のアニメーション技術で説明する。
３．節点を検索するために、初期設定を行う。
４．部材表示パラメータ ipkmem が 0 の場合、以下の処理を行う。

５．全節点について以下の処理を行う。
６．透視画面上の節点位置 pk()とマウスのクリック位置 gxy()間の長さの 2 乗 xdd を計算する。
７．以前の最小値と比較し、最新の節点位置が以前の最小値より小さ場合、その節点番号を変数 iptpk に、その長さの 2 乗を xd に保存する。
８．部材を検索するために、初期設定を行う。
９．全部材について以下の処理を行う。
10. 部材両端の節点番号をセットする。
11. 部材中央位置とマウスのクリック位置間の長さの 2 乗 xdd を計算する。
12. 以前の最小値と比較し、最新の部材位置が以前の最小値より小さ場合、その部材番号を変数 mempk に、その長さの 2 乗を xd に保存する。
13. 全部材について以上の処理が終了すると、マウスクリック位置から最も近い部材が検索される。
14. 部材表示パラメータ ipkmem が 2 以外の場合以下の処理を行う。
15. 全部材についてチェックする。
16. 部材のタイプを取得する。
17. 表示部材かどうかチェックし、表示部材である場合は以下の検索処理を行う。後は、else 以前の処理と同様である。

次に、関数 mem_pik.datset()、mem_pik.DoModal()を用いて、マウスで指定した節点あるいは部材の情報をダイアログで画面に表示する。この 2 つは、クラス CDIG_nodpik のメンバー関数である。このクラスの全記述を次に示すことにする。ただし、初期設定と DoDataExchange 部分には省略がある。関数 mem_pik.datset()では、検索した節点番号から、その節点の情報であるその時刻の変位や最大変位、あるいは、速度や加速度が読み込まれていれば、その値が所定メンバー変数にセットされる。その後、DoModal 文でダイアログが表示されることになる。内容は非常に単純などで、理解することは容易であろう。

```
//    ――――――――――――――――――――――――――――――――
//    ●    DIG_nodpik.cpp : インプリメンテーション ファイル
//    ――――――――――――――――――――――――――――――――
//
#include "stdafx.h"
#include "sf31.h"
#include "DIG_nodpik.h"
```

```
#include "sf31datex.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//     ――――――――――――――――――――――――――――――――――
//     ●    CDIG_nodpik ダイアログ
//     ――――――――――――――――――――――――――――――――――
//
CDIG_nodpik::CDIG_nodpik(CWnd* pParent /*=NULL*/)
       : CDialog(CDIG_nodpik::IDD, pParent)
{
       //{{AFX_DATA_INIT(CDIG_nodpik)
       m_edit_dis1 = 0.0f;
       m_edit_dis10 = 0.0f;
       m_edit_dis11 = 0.0f;
       ・
       ・
       m_edit_p_chek1 = _T("");
       m_edit_p_chek2 = _T("");
       m_edit_p_chek3 = _T("");
       m_edit_p_aacc1 = 0.0f;
       //}}AFX_DATA_INIT
}
//     ――――――――――――――――――――――――――――――――――
//     ●    DoDataExchange
//     ――――――――――――――――――――――――――――――――――
//
void CDIG_nodpik::DoDataExchange(CDataExchange* pDX)
{
       CDialog::DoDataExchange(pDX);
       //{{AFX_DATA_MAP(CDIG_nodpik)
       DDX_Text(pDX, IDC_EDIT_DIS1, m_edit_dis1);
       DDX_Text(pDX, IDC_EDIT_DIS10, m_edit_dis10);
       DDX_Text(pDX, IDC_EDIT_DIS11, m_edit_dis11);
       ・
       ・
       DDX_Text(pDX, IDC_EDIT1_TIME, m_edit_time);
       DDX_Text(pDX, IDC_EDIT_CHEK4, m_edit_p_chek1);
       DDX_Text(pDX, IDC_EDIT_CHEK5, m_edit_p_chek2);
       DDX_Text(pDX, IDC_EDIT_CHEK6, m_edit_p_chek3);
       DDX_Text(pDX, IDC_EDIT_AACC7, m_edit_p_aacc1);
       //}}AFX_DATA_MAP
}

//     ――――――――――――――――――――――――――――――――――
//     ●    CDIG_nodpik ダイアログ
//     ――――――――――――――――――――――――――――――――――
//
BEGIN_MESSAGE_MAP(CDIG_nodpik, CDialog)
       //{{AFX_MSG_MAP(CDIG_nodpik)
       //}}AFX_MSG_MAP
```

```
END_MESSAGE_MAP()
//     ――――――――――――――――――――――――――――――――――
//     ●    datset：節点情報をメンバー変数にセットする
//     ――――――――――――――――――――――――――――――――――
//
void CDIG_nodpik::datset(int pointpik,int m_nstep, int itime, float stime)
{
float m_edit_disx[30],axx;
     m_edit_no_node = pointpik;
int ipoint = 6*(pointpik -1);
       m_edit_f1 = _T("Free");
       if( F_free[ipoint] == -1)   m_edit_f1 = _T("Fix");
       m_edit_f2 = _T("Free");
       if( F_free[ipoint+1] == -1) m_edit_f2 = _T("Fix");
       m_edit_f3 = _T("Free");
       if( F_free[ipoint+2] == -1) m_edit_f3 = _T("Fix");
       m_edit_f4 = _T("Free");
       if( F_free[ipoint+3] == -1) m_edit_f4 = _T("Fix");
       m_edit_f5 = _T("Free");
       if( F_free[ipoint+4] == -1) m_edit_f5 = _T("Fix");
       m_edit_f6 = _T("Free");
       if( F_free[ipoint+5] == -1) m_edit_f6 = _T("Fix");
       m_edit_chek1 = _T("X");
       m_edit_chek2 = _T("X");
       m_edit_chek3 = _T("X");
    for ( int i=0;i<30;i++) m_edit_disx[i]=0.;
       for (   i=0;i<m_nstep;i++){
         ipoint=(3*node)*i + 3*(pointpik-1);
         for (int j=0;j<3;j++){
         axx=F_disp[ipoint+j];
         if(axx < m_edit_disx[j]) m_edit_disx[j] = axx;
         if(axx > m_edit_disx[j+6]) m_edit_disx[j+6] = axx;
          }
     }
    if(F_read_vel == 1){
       m_edit_chek3 = _T("0");
       for (   i=0;i<m_nstep;i++){
         ipoint=(3*node)*i + 3*(pointpik-1);
         for (int j=0;j<3;j++){
            axx=F_vel[ipoint+j];
            if(axx < 0.) axx=-axx;
            if(axx > m_edit_disx[j+24]) m_edit_disx[j+24] = axx;
         }
   }}
   if(F_read_acc == 1){
       m_edit_chek1 = _T("0");
        for (   i=0;i<m_nstep;i++){
         ipoint=(3*node)*i + 3*(pointpik-1);
         for (int j=0;j<3;j++){
            axx=F_acc[ipoint+j];
            if(axx < 0.) axx=-axx;
            if(axx > m_edit_disx[j+12]) m_edit_disx[j+12] = axx;
         }
  }}
```

```
    if(F_read_accab == 1){
           m_edit_chek2 = _T("O");
           for (   i=0;i<m_nstep;i++){
               ipoint=(3*node)*i + 3*(pointpik-1);
               for (int j=0;j<3;j++){
                 axx=F_accab[ipoint+j];
                 if(axx < 0.) axx=-axx;
                 if(axx > m_edit_disx[j+18]) m_edit_disx[j+18] = axx;
               }
    }}
          float ff1 = 0.0001;
          float ff2 = -0.0001;
          for( i=0;i<30;i++){
            if(m_edit_disx[i] < ff1 && m_edit_disx[i] > ff2 ) m_edit_disx[i]=0.;
          }
          int iit = itime -1;
          m_edit_p_aacc1 = 0.0f;
          m_edit_p_aacc2 = 0.0f;
          m_edit_p_aacc3 = 0.0f;
          m_edit_p_dis1 = 0.0f;
          m_edit_p_dis2 = 0.0f;
          m_edit_p_dis3 = 0.0f;
          m_edit_p_racc1 = 0.0f;
          m_edit_p_racc2 = 0.0f;
          m_edit_p_racc3 = 0.0f;
          m_edit_p_vel1 = 0.0f;
          m_edit_p_vel2 = 0.0f;
          m_edit_p_vel3 = 0.0f;
          m_edit_time = 0.0f;
          m_edit_p_chek1 = _T("X");
          m_edit_p_chek2 = _T("X");
          m_edit_p_chek3 = _T("X");

          if(iit >= 0 && iit < m_nstep){
            ipoint=(3*node)*iit + 3*(pointpik-1);
            m_edit_p_dis1 = F_disp[ipoint];
            m_edit_p_dis2 = F_disp[ipoint+1];
            m_edit_p_dis3 = F_disp[ipoint+2];
            m_edit_time = stime;
            if(F_read_vel == 1){
              m_edit_p_chek3 = _T("O");
              m_edit_p_vel1 = F_vel[ipoint];
              m_edit_p_vel2 = F_vel[ipoint+1];
              m_edit_p_vel3 = F_vel[ipoint+2];
            }
            if(F_read_acc == 1){
               m_edit_p_chek1 = _T("O");
              m_edit_p_aacc1 = F_acc[ipoint];
              m_edit_p_aacc2 = F_acc[ipoint+1];
              m_edit_p_aacc3 = F_acc[ipoint+2];
            }
            if(F_read_accab == 1){
               m_edit_p_chek2 = _T("O");
              m_edit_p_racc1 = F_accab[ipoint];
```

```
            m_edit_p_racc2 = F_accab[ipoint+1];
            m_edit_p_racc3 = F_accab[ipoint+2];
        }
        }

        m_edit_dis1 = m_edit_disx[0];
        m_edit_dis2 = m_edit_disx[1];
        m_edit_dis3 = m_edit_disx[2];
        m_edit_dis7 = m_edit_disx[6];
        m_edit_dis8 = m_edit_disx[7];
        m_edit_dis9 = m_edit_disx[8];
        m_edit_racc1 = m_edit_disx[12];
        m_edit_racc2 = m_edit_disx[13];
        m_edit_racc3 = m_edit_disx[14];
        m_edit_aacc1 = m_edit_disx[18];
        m_edit_aacc2 = m_edit_disx[19];
        m_edit_aacc3 = m_edit_disx[20];
        m_edit_vel1 = m_edit_disx[24];
        m_edit_vel2 = m_edit_disx[25];
        m_edit_vel3 = m_edit_disx[26];
    }
```