6 Header-Datei cgiw.h, Beschreibung aller Funktionen

Die gesamte **CGIW**-Klassen-Bibliothek steht im Quellcode zur Verfügung. Die Dateinamen der Quellcode-Dateien sind mit den Funktionsnamen identisch (Ausnahmen: Konstruktoren und Destruktoren, die in Dateien stehen, die dem Klassennamen entsprechen), so daß diese leicht zu finden sind.

Jede Funktion wird mit allen Parametern, die sie erwartet, und den Ergebnissen, die sie liefert, ausführlich im Quelltext beschrieben, so daß der Programmierer, der sie verwenden will, dort die komplette Information findet.

Jede Programm-Datei, die die CGIW-Klassen-Bibliothek verwendet, muß die Header-Datei cgiw.h einbinden. Da diese die Deklarationen aller Methoden aller Klassen der Bibliothek enthält und damit eine Schnell-Orientierung über die verfügbaren Methoden ermöglicht, wird sie nachfolgend komplett gelistet:

```
// Deklarationen der CGIW-Klassen

#include <afxwin.h>
#ifndef CGIW_GI
   #define CGIW_GI
   #define DOUBLE_EPS_GI  1.e-12
   #define INV_EPS_GI     1.e-10
   #define GI_PI          3.141592653589793
   #define MARDIV_GI   20  // Zur Festlegung des Standard-Marker-Offsets:
                           // m_maroff = "1 Logical Inch" / MARDIV_GI

class CGBas
  {
     public:
       enum GI_COLORS     {GI_TRANSPARENT = - 1 ,
                           GI_BLACK   ,
                           GI_BLUE    ,
                           GI_GREEN   ,
                           GI_CYAN    ,
                           GI_RED     ,
                           GI_MAGENTA ,
                           GI_YELLOW  ,
                           GI_WHITE   } ;
     public:
        CGBas () { }
       ~CGBas () { }
       //  Mittelwert und Grenzen der Abmessungen der "World coordinates":
       double   avsiz_gi (int    npoin          , double *xyzw          ,
                          double *xmin_p = NULL , double *xmax_p = NULL  ,
                          double *ymin_p = NULL , double *ymax_p = NULL  ,
                          double *zmin_p = NULL , double *zmax_p = NULL) ;
       //  "Verkuerzen einer 3D-Linie":
       void     dmlen_gi (double *x1_p , double *y1_p , double *z1_p ,
                          double *x2_p , double *y2_p , double *z2_p ,
                          double d1    , double d2) ;
       //  Mittelwerte der Koordinaten eines Polygons ("World coordinates"):
       void     plcen_gi (double *xc_p , double *yc_p   , double *zc_p ,
                          int    npoin , double *xyzw_p ,
                          int *points_p = NULL) ;
       //  Berechnen eines koordinatenabhaengigen Farbwertes:
       COLORREF cdcol_gi (double xyz , double xyzmin , double xyzmax) ;
```

```
      //  RGB-Werte fuer Grundfarben (siehe GI_COLORS oben):
      COLORREF mkrgb_gi (int color) ;
  } ;

class CPT : public CGBas
{
    public:
      enum GI_PROJECTION {GI_CENTRAL  ,
                          GI_PARALLEL } ;
      enum GI_AXIS       {GI_AXISX ,
                          GI_AXISY ,
                          GI_AXISZ } ;
    private:
      double  m_tplane [ 9] ,
              m_tspace [16] ;
      int     m_prjtyp ;
      double  m_eyept  [3] ,
              m_rptr0  [3] ,
              m_normn  [3] ,
              m_upvec  [3] ,
              m_dparp  [3] ,
              m_pmtinv [9] ;
      double  m_scalsx ,
              m_scalsy ,
              m_scalsz ;
      double  m_vpyang ;
    public:
       CPT  (int prjtyp = GI_PARALLEL) ;
      ~CPT  () { }
      // Inline-Funktionen:
      int    prgtp_pt ()             { return m_prjtyp   ; }
      double prgta_pt ()             { return m_vpyang   ; }
      void   prsta_pt (double vpyang) { m_vpyang = vpyang ; }
      // 2D-Transformationen initialisieren, setzen, aendern:
      void   t2gtm_pt (double *t2d_p)  ;
      void   tgttm_pt (double *tm_p) ;
      void   tinit_pt (void) ;
      void   tmamu_pt (double *tnew_p) ;
      void   tmirx_pt () ;
      void   tmiry_pt () ;
      void   trota_pt (double xm , double ym , double phi) ;
      void   tru2t_pt (double xu     , double yu    ,
                       double *txu_p , double *tyu_p) ;
      void   tsttm_pt (double *tm_p) ;
      void   ttabs_pt (double tx , double ty , double phi ,
                       double sx , double sy) ;
      // 3D-Transformationen initialisieren, setzen, aendern:
      void   t3abs_pt (double tx  = 0. , double ty = 0. , double tz = 0. ,
                       double phi = 0. , int    axis = GI_AXISX ,
                       double sx  = 1. , double sy = 1. ,
                       double sz = 1.) ;
      void   t3gtm_pt (double *t3d_p)  ;
      void   t3mam_pt (double *tnew_p) ;
      void   t3rot_pt (double phi , int axis) ;
      void   t3trn_pt (double tx , double ty , double tz) ;
      void   t3w2w_pt (double  xw    , double  yw    , double  zw ,
                       double *txw_p , double *tyw_p , double *tzw_p) ;
      void   taw2w_pt (int np , double *xyzw_p , double *txyzw_p) ;
      // Projektionen:
      int    pmkpi_pt (void) ;
      void   prgip_pt (double *pmtinv_p) ;
```

```
      void   prgtd_pt (double *xwd_p , double *ywd_p , double *zwd_p) ;
      void   prgte_pt (double *xwe_p , double *ywe_p , double *zwe_p) ;
      void   prgtn_pt (double *xwn_p , double *ywn_p , double *zwn_p) ;
      void   prgtr_pt (double *xwr_p , double *ywr_p , double *zwr_p) ;
      void   prgtu_pt (double *xwu_p , double *ywu_p , double *zwu_p) ;
      void   prini_pt (int prjtyp = GI_PARALLEL) ;
      int    prinv_pt (double *a_p , double eps) ;
      int    projn_pt (int ptyp) ;
      int    prstc_pt (double xwe , double ywe , double zwe ,
                       double xwr , double ywr , double zwr) ;
      int    prstp_pt (double xwd , double ywd , double zwd ,
                       double xwr , double ywr , double zwr) ;
      int    prw2u_pt (double xw , double  yw   , double  zw ,
                                   double *xu_p , double *yu_p) ;
      int    prwul_pt (double xw  , double  yw , double  zw ,
                       double *xu , double *yu , double *lambda) ;
      // Projektion und 3D-Transformation:
      double ptdis_pt (double xw , double yw , double zw) ;
      double ptdis_pt (double *xyzw_p) ;
      int    ptmx3_pt (int    npoin    , double xyzw []   ,
                       double *xumin_p , double *xumax_p  ,
                       double *yumin_p , double *yumax_p) ;
      void   ptsta_pt (CPT *cpt_p) ;
      int    ptw2u_pt (double xw    , double  yw , double  zw ,
                       double *xu_p , double *yu_p) ;
      int    ptwul_pt (double xw  , double  yw , double  zw ,
                       double *xu , double *yu , double *lambda) ;
} ;

class CGI : public CPT
{
   public:
     enum GI_ORIGIN     {GI_XYBOTTOMLEFT  ,
                         GI_XYCENTER      ,
                         GI_XYTOPLEFT     ,
                         GI_XYTOPRIGHT    ,
                         GI_XYBOTTOMRIGHT } ;
     enum GI_MARKER     {GI_MKNOMARKER ,
                         GI_MKCIRCLE   ,
                         GI_MKSQUARE   ,
                         GI_MKCROSS    ,
                         GI_MKVBAR     ,
                         GI_MKHBAR     ,
                         GI_MKFCIRCLE  ,
                         GI_MKFSQUARE  } ;
     enum GI_OFFSET     {GI_NOOFFSET  ,
                         GI_UPLEFT    ,
                         GI_LEFT      ,
                         GI_DOWNLEFT  ,
                         GI_UP        ,
                         GI_CENTER    ,
                         GI_DOWN      ,
                         GI_UPRIGHT   ,
                         GI_RIGHT     ,
                         GI_DOWNRIGHT } ;
     enum GI_VPCLIP     {GI_NOCLIPPING ,
                         GI_CLIPPING   } ;
     enum GI_CURSOR     {GI_NOCURSOR      ,
                         GI_CUBIGCROSS    ,
                         GI_CUSMALLCROSS  ,
                         GI_CURECTANGLE } ;
   private:
     CDC*    m_dc_p  ;
     CRgn*   m_rgn_p ;
```

```
  int     m_nvppdh , // Pixel-Differenz (horizontal) der Randpunkte im
                     // "Current viewport"
          m_nvppdv , // Pixel-Differenz (vertikal) der Randpunkte im
                     // "Current viewport"
                     // (die Anzahl der Pixel des Viewports ist jeweils
                     // um 1 groesser als m_nvppdh bzw. m_nvppdv)
          m_vppulx , // "Upper-left"-x-Koordinate des "Current viewport"
          m_vppuly , // "Upper-left"-y-Koordinate des "Current viewport"
          m_vcoord , // Typ des Viewport-Koordinatensystems:
                     // 0 - Linke untere Ecke,  x nach rechts,
                     //                         y nach oben
                     // 1 - Viewport-Mitte,     x nach rechts,
                     //                         y nach oben
                     // 2 - Linke obere Ecke,   x nach rechts,
                     //                         y nach unten
                     // 3 - Rechte obere Ecke,  x nach links,
                     //                         y nach unten
                     // 4 - Rechte untere Ecke, x nach links,
                     //                         y nach oben
          m_cpsset , // "Current position" ...
                     // 0 - ist undefinert,
                     // 1 - ist definiert (m_cuposx,m_cuposy),
                     //     aber noch nicht gesetzt
                     // 2 - ist definiert (m_cuposx,m_cuposy)
                     //     und bereits gesetzt
          m_cuposx , // "Window coordinates" ...
          m_cuposy , // ... der "current Position"
                     // ("Window Coordinates" beziehen sich immer auf
                     // ein Koordinatensystem in der linken oberen Ecke
                     // des "Current viewport", Umrechnungen auf die
                     // fuer den Viewport geltenden "Viewport
                     // voordinates" werden von den Zeichenroutinen
                     // ausgefuehrt)
          m_maroff , // Standard-Marker-Offset (halbe Kantenlaenge des
                     // umschliessenden Quadrats)
          m_vpclip , // 1 - Viewports arbeiten mit Clipping
                     // 0 - Viewports arbeiten ohne Clipping
          m_curpox , // Aktuelle Cursorposition eines
          m_curpoy , // Spezialcursors
          m_curvis , // Typ des Spezialcursors
          m_curfix , // Festposition des Rechteckcursors,
          m_curfiy ; // gespeichert in "Window coordinates"
  double  m_p1xwin ,
          m_p1ywin ,
          m_p2xwin ,
          m_p2ywin , // "User coordinates" der Viewport-Eckpunkte
          m_x1u    ,
          m_y1u    ; // Erster Punkt bei Rechteck-Pick
public:
  // Konstruktoren, Destruktor und Initialisierung:
  CGI  (CDC* dc_p , int cxClient , int cyClient ,
                                   int vpclip = GI_CLIPPING) ;
  CGI  (CDC* dc_p , int cxClient , int cyClient , CPT *cpt_p) ;
  CGI  (CWnd *wnd_p = NULL , CDC* dc_p   = NULL ,
                                  int vpclip = GI_CLIPPING) ;
  CGI  (CWnd *wnd_p , CDC *dc_p , CPT *cpt_p ,
                                  int vpclip = GI_CLIPPING) ;
  ~CGI () ;
  void     init_gi  (CDC* dc_p , int cxClient , int cyClient ,
                                 int vpclip = GI_CLIPPING) ;
  void     init_gi  (CWnd *wnd_p , CDC *dc_p ,
                                   int  vpclip = GI_CLIPPING) ;
  void     init_gi  (CWnd *wnd_p , CDC *dc_p ,
                     CPT *cpt_p  , int  vpclip = GI_CLIPPING) ;
```

```
// Inline-Funktionen:
CDC*     getdc_gi ()          { return m_dc_p   ; }
void     remdc_gi ()          { m_dc_p = NULL   ; }
void     setdc_gi (CDC* dc_p) { m_dc_p = dc_p   ; }
int      gtcxv_gi ()          { return m_nvppdh + 1 ; }
int      gtcyv_gi ()          { return m_nvppdv + 1 ; }
double   gtp1x_gi ()          { return m_p1xwin ; }
double   gtp1y_gi ()          { return m_p1ywin ; }
double   gtp2x_gi ()          { return m_p2xwin ; }
double   gtp2y_gi ()          { return m_p2ywin ; }
// Viewport, "Viewport coordinates", "User coordinates":
void     stcvp_gi (int pulx  , int puly   ,
                   int width , int height , int cstype = GI_XYTOPLEFT) ;
void     stuca_gi (double p1xu , double p1yu ,
                   double p2xu , double p2yu , double pmarg = 0.) ;
void     stuci_gi (double p1xu , double p1yu ,
                   double p2xu , double p2yu , double pmarg = 0.) ;
// 2D-Zeichnen mit "Viewport coordinates":
void     vback_gi (CBrush &brush , CPen &pen , int offset = 0) ;
void     vfram_gi (int offset = 0) ;
void     vfrec_gi (int xv1 , int yv1 , int xv2 , int yv2) ;
void     vline_gi (int xv , int yv) ;
void     vmove_gi (int xv , int yv) ;
void     vrect_gi (int xv1 , int yv1 , int xv2 , int yv2) ;
// 2D-Zeichnen mit "User coordinates":
int      mksiz_gi (double msize) ;
void     udarc_gi (double xc , double yc , double r  ,
                   double xs , double ys , double xe , double ye) ;
void     uearc_gi (double xc , double yc , double a  , double b ,
                   double xs , double ys , double xe , double ye) ;
void     ufcir_gi (double xc , double yc , double r) ;
void     ufell_gi (double xu1 , double yu1 , double xu2 , double yu2) ;
void     ufpie_gi (double x1 , double y1 , double x2 , double y2 ,
                   double xs , double ys , double xe , double ye) ;
void     ufpol_gi (int npoin , double xu [] , double yu []) ;
void     ufrec_gi (double xu1 , double yu1 , double xu2 , double yu2) ;
void     ufsec_gi (double xc , double yc , double r  ,
                   double xs , double ys , double xe , double ye) ;
void     uline_gi (double xu , double yu) ;
void     umark_gi (int mtype , double msize ,
                   double xu , double yu    , int offset = 0) ;
void     umove_gi (double xu , double yu) ;
void     uwidl_gi (double xu1     , double yu1 ,
                   double xu2     , double yu2 ,
                   CPen   *ipen_p , CPen   *wpen_p) ;
// "Zeichnende t...-Funktionen":
void     tdarc_gi (double xc , double yc , double r  ,
                   double xs , double ys , double xe , double ye) ;
void     tfcir_gi (double xc , double yc , double r) ;
void     tline_gi (double xu , double yu) ;
void     tmove_gi (double xu , double yu) ;
// Punkte picken und Zoom:
int      picvp_gi (int pxd , int pyd , int *pxv_p = NULL  ,
                                       int *pyv_p = NULL) ;
int      picid_gi () ;
int      rpick_gi (CWnd   *wnd_p , POINT point    ,
                   double *x1u_p , double *y1u_p  ,
                   double *x2u_p , double *y2u_p) ;
void     scapp_gi (CWnd *wnd_p , int xv , int yv , int ctype) ;
void     scapp_gi (CWnd *wnd_p , double xu , double yu , int ctype) ;
void     scdap_gi (CWnd *wnd_p) ;
void     scdrw_gi (CDC *dc_p) ;
void     scupd_gi (CWnd *wnd_p , POINT point) ;
int      umpos_gi (POINT point , double *xu_p , double *yu_p) ;
```

```
     // Umrechnung von Koordinaten:
     int      xyu2v_gi (double xu , double yu , int *xv_p , int *yv_p) ;
     int      xyu2w_gi (double xu , double yu , int *xw_p , int *yw_p) ;
     void     xyv2w_gi (int xv  , int yv  , int *xw_p , int *yw_p) ;
     void     xyv2u_gi (int xv , int yv , double *xu_p , double *yu_p) ;
     // Funktionen, die die aktuelle Projektion auswerten:
     int      prlin_gi (double xw , double yw , double zw) ;
     int      prmov_gi (double xw , double yw , double zw) ;
     // Funktionen, die die aktuelle Projektion und die
     // aktuelle Transformation auswerten:
     int      ptbll_gi (double xw  , double  yw , double  zw ,
                                                  double  radw) ;
     int      ptfpl_gi (int    npoin , double *xyzw_p ,
                                       int    *points_p = NULL) ;
     int      ptlin_gi (double xw , double yw , double zw) ;
     int      ptmov_gi (double xw , double yw , double zw) ;
     int      ptmrk_gi (int    mtype , double msize , double xw ,
                        double yw    , double zw    , int offset) ;
     int      ptwdl_gi (double x1w  , double y1w , double z1w ,
                        double x2w  , double y2w , double z2w ,
                        CPen   *ipen_p  , CPen   *wpen_p ,
                        double d1w = 0. , double d2w = 0.) ;
} ;

// Deklaration der Klassen fuer die Verwaltung von
// Graphik-Modellen:

class CGElem : public CGBas
{
     private:
       int      m_nop      ;      // Anzahl der int-Werte im m_param-Array
       int      *m_param_p ;      // ... zur Elementbeschreibung (i. a.
                                  // Knotennummern, deren Koordinaten im
                                  // CGMod-Objekt gespeichert sind)
       COLORREF m_color1   ;
       COLORREF m_color2   ;
       CGElem   *m_next_p  ;      // ... fuer Verkettung in der Liste
     public:
        CGElem  (void) ;
       ~CGElem  (void) ;
       // Inline-Funktionen:
       void stnop_md (int nop)        { m_nop = nop ; }
       void stpap_md (int *param_p)   { m_param_p = param_p ; }
       void stcl1_md (COLORREF color) { m_color1 = color ; }
       void stcl2_md (COLORREF color) { m_color2 = color ; }
       void stnxp_md (CGElem *next_p) { m_next_p = next_p ; }
       int        gtnop_md () const   { return m_nop ; }
       int*       gtpap_md () const   { return m_param_p ; }
       COLORREF   gtcl1_md () const   { return m_color1 ; }
       COLORREF   gtcl2_md () const   { return m_color2 ; }
       CGElem*    gtnxp_md () const   { return m_next_p ; }
} ;

class CGMod : public CGBas
{
     private:
       CGElem   *m_root_p ;             // Pointer fuer die Verwaltung
       CGElem   *m_last_p ;             // der verketteten Liste der
       CGElem   *m_next_p ;             // der CGElem-Objekte
       int      m_ne      ;             // Anzahl der Elemente
       int      m_nk      ;             // Anzahl der Knoten
       int      m_ke      ;             // Anzahl der Knoten pro Element
       int      m_kx      ;             // "Dimensionalitaet"
```

```
       double   *m_xy_p   ;              // Feld der Knotenkoordinaten
                                         // (m_nk * m_kx double-Werte)
     public:
       // Konstruktor, Destruktor:
        CGMod  (CGElem *root_p = NULL) ;
       ~CGMod  (void) ;
       // Inline-Funktionen:
       int     gtmne_md ()          const { return m_ne ; }
       int     gtmnk_md ()          const { return m_nk ; }
       int     gtmke_md ()          const { return m_ke ; }
       int     gtmkx_md ()          const { return m_kx ; }
       double* gtxyp_md (int i = 1) const { return m_xy_p + (i - 1) * m_kx ; }
       // Modellbeschreibung vom File lesen:
       void    appel_md (CGElem *elem_p) ;
       int     rddfl_md (char const *pathnm_p) ;
       // Elementliste bearbeiten:
       CGElem* gtfel_md (int *nop_p = NULL) ;
       CGElem* gtnel_md (int *nop_p = NULL) ;
       // Extremwerte und Mittelwert der Laengen eines Stab-Modells:
       double   avlen_md (double *minlen_p , double *maxlen_p) ;
} ;

class CGObj : public CGBas     // Abstrakte Klasse zur Unterstuetzung
{                              // des Anlegens und Abarbeitens eines
     private:                  // binaeren Baums fuer das "Zeichnen von
                               // Objekten in der Reihenfolge ihres
       double   m_dist    ;    // Abstands vom Betrachter"
       CGObj   *m_left_p  ;
       CGObj   *m_right_p ;
     public:
        CGObj  (double dist = 0.) ;
       ~CGObj  (void) ;
       // Inline-Funktion:
       double  gtdis_md () { return m_dist ; }
       // Bearbeiten des binaeren Baumes:
       void    dromd_md (CGI   *cgi_p) ;
       void    drrec_md (CGObj *anchor_p) ;
       CGObj*  insot_md (CGObj *root_p , double dist) ;
       void    updbt_md (CGObj *anchor_p) ;
       // Virtuelle Funktion, die das Zeichnen fuer ein Objekt einer
       // aus CGObj abgeleiteten Klasse ausfuert:
       virtual void drobj_md (CGI *cgi_p) = 0 ;
} ;

class CGNod : public CGObj
{
     protected:
       double   *m_xyz_p  ;
       COLORREF m_linecol ;
       COLORREF m_fillcol ;
     public:
        CGNod  (double *xyz_p , COLORREF linecol , COLORREF fillcol) ;
        CGNod  (double *xyz_p = NULL) ;
       ~CGNod  (void) { }
        void drobj_md (CGI *cgi_p) ;
} ;
```

```
class CGRod : public CGObj
{
     protected:
       double   *m_xyz1_p ;
       double   *m_xyz2_p ;
       int      m_wpix ;
       int      m_ipix ;
       COLORREF m_wcol ;
       COLORREF m_icol ;
     public:
        CGRod  (double *xyz1_p , double *xyz2_p ,
                int wpix , int ipix , COLORREF wcol , COLORREF icol) ;
        CGRod  (double *xyz1_p = NULL , double   *xyz2_p = NULL ,
                int    wpix    = 5    , int      ipix    = 1) ;
       ~CGRod  (void) { }
        void drobj_md (CGI *cgi_p) ;
} ;

class CGPoly : public CGObj
{
     protected:
       int      m_npoin     ;
       double   *m_xyz_p    ;
       int      *m_points_p ;
       COLORREF m_linecol   ;
       COLORREF m_fillcol   ;
     public:
        CGPoly  (int npoin , double   *xyz_p  , int      *points_p ,
                             COLORREF fillcol , COLORREF linecol = 0)  ;
        CGPoly  (int npoin = 0 , double *xyz_p = NULL ,
                                 int    *points_p = NULL) ;
       ~CGPoly  (void) { }
        void drobj_md (CGI *cgi_p) ;
} ;

class CArea3d : public CGBas // Abstrakte Klasse fuer die Darstellung
{                            // einer 3D-Flaeche
   private:
       double  m_xu1 ;       // Grenzen der beiden unabhaengigen Variablen
       double  m_yv1 ;       // (bei Flaechendefinition inn der Darstellung
       double  m_xu2 ;       // z = z(x,y): x1,...,x2 und y1,...,y2, bei
       double  m_yv2 ;       // Parameterdarstellung u1,...,u2 und v1,...,v2)
       int     m_xusteps ;   // Anzahl der Polygone in den beiden
       int     m_yvsteps ;   // Koordinatenrichtungen
   public:
       CArea3d (double xu1     = -1. , double yv1     = -1. ,
                double xu2     =  1. , double yv2     =  1. ,
                int    xusteps = 10  , int    yvsteps = 10) ;
       virtual ~CArea3d  () { }
       // Inline-Funktionen:
       double gtxu1_md () { return m_xu1 ; }
       double gtyv1_md () { return m_yv1 ; }
       double gtxu2_md () { return m_xu2 ; }
       double gtyv2_md () { return m_yv2 ; }
       double gtdxu_md () { return (m_xu2 - m_xu1) / m_xusteps ; }
       double gtdyv_md () { return (m_yv2 - m_yv1) / m_yvsteps ; }
       int    gtxus_md () { return m_xusteps ; }
       int    gtyvs_md () { return m_yvsteps ; }
       // Setzen der Klassen-Elemente (Funktionen befinden sich gemeinsam
       // mit dem Konstruktor in der Datei ca3d_gi.cpp):
       void   stint_md (double xu1  , double yv1   ,
                        double xu2  , double yv2)  ;
```

```
        void   ststp_md (int xusteps , int yvsteps) ;

        // Rein virtuelle Funktion, mit der in einer abgeleiteten Klasse
        // ein Punkt der Flaeche berechnet wird:
        virtual int gtptc_md (double xu , double yv , double *xyz_p) = 0 ;
} ;
#endif
```