Journal of Computers Vol. 29 No. 2, 2018, pp. 249-260
doi:10.3966/199115992018042902024

3D Reconstruction of Medical Image Based on OpenGL

Zhen-Huan Zhou[1*], Xiao-Jun Wen[1]

School of Computer Engineering, Shenzhen Polytechnic, Shenzhen, China, 518055
{zhouzhenhuan, wxjun}@szpt.edu.cn

Received 24 July 2017; Revised 19 September 2017; Accepted 19 October 2017

Abstract. 3D reconstruction of medical image based on OpenGL is a complex process, in which 3D surface rendering is the most widely used in medical diagnosis and treatment. In this paper, we introduce the whole process of 3D surface rendering in detail. 1. Reading in medical sequence image files; 2. Interpolating based on layer thickness and scale factor; 3. Adjusting window width/center and converting 16bit raw data to 8bit display data. 4.Making three orthogonal planes(axial, sagittal and coronal). 5. Making 3D skin points and computing their normal vectors. 6. 3D surface rendering based on OpenGL. 7. Dragging with the mouse, 3D image can rotate the x, y, z axis.

Keywords: 3D reconstruction, medical image, MFC, OpenGL, suface rendering

1 Introduction

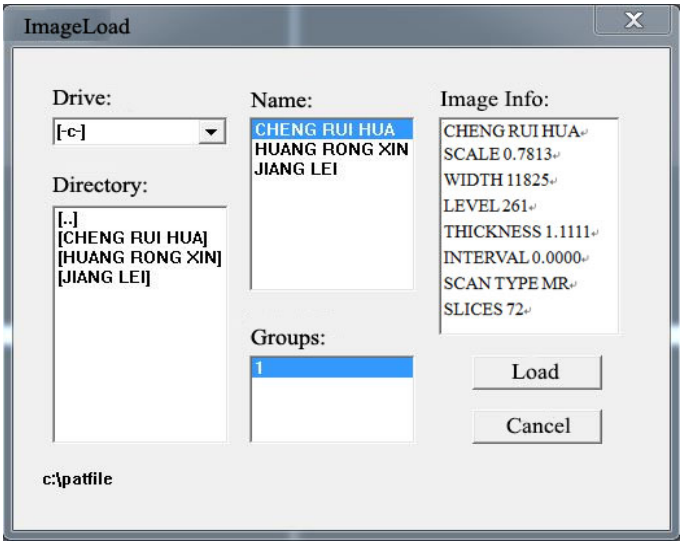
OpenGL was originally Silicon Graphics Incorporated (SGI) to develop high-quality image interface on their graphics workstations. Later became a very good open three-dimensional graphics interface. In fact it is the graphical software and hardware interface, which includes more than 120 graphics functions, "GL" is "GRAPHIC LIBRARY" acronym, meaning "graphics library." The emergence of OpenGL allows most programmers to develop complex 3D graphics in C on a PC. Microsoft has provided three OpenGL libraries (glu32.lib, glau.lib, OpenGL32.lib) in Visual C ++ 6.0 that allow us to easily program and quickly generate 3D medical images.

The use of OpenGL to draw three-dimensional images, mainly through the following steps: OpenGL settings, image loading, OpenGL initialization, generate drawing points, three-dimensional drawing. The use of OpenGL to draw three-dimensional objects is a complex process, to a number of OpenGL settings, not only to know the location of the three-dimensional point, but also to calculate the three-dimensional point normal, set the three-dimensional point of light and material, and finally, On the screen is displayed. Three-dimensional image of a large amount of data, the core issue is to show the speed and effectiveness.

3D reconstruction of medical image based on OpenGL is a basic and core technology, Especially 3D surface rendering is widely used to medical imaging equipment [1-3], such as CT, MRI and so on. 3D surface rendering of medical image is a complex process [4-6], and needs seven steps to complete at least [7-10].

First, image information should be confirmed before reading, such as drives, directory, name, scan type, the images slices, thickness, interval, width, level etc. At this time, raw image data is 16 bits sequence images. In general, scanning interval is bigger, if you want reconstruct a real 3d image, you must have an image interpolation between slices and fill the missing image. In order to display 2D images, 16 bits raw data muse be converted to 8 bits displaying data, window width and window level may be adjusted as well. 2D images are usually three orthogonal planes: transverse, sagittal and coronal plane. 3D reconstruction by OpenGL must find skin dots over the threshold, compute their normal vectors and render 3D surface directly. 3D image may rotate around its center in real-time.

* Corresponding Author



2 Reading in Medical Sequence Image Files

```
////////////////////////////////////////////////////////////////////////////
CFile fImg;
CFileException fe;
image_header imgHeader;
String strImgPath, nImgNo;
// Read header file information
CString strFileName;
strFileName.Format("%s%s%d%s", strImgPath,"\\", nImgNo, "_1.img");
if(fImg.Open(strFileName, CFile::modeRead|CFile::typeBinary|CFile::shareExclusive, &fe)==NULL)
{
      AfxMessageBox("Failed to read file header!");
      return;
}
fImg.Read(&imgHeader,sizeof(image_header));
fImg.Close();
m_listboxpatinfo.ResetContent();
CString strListBoxPatInfo;
strListBoxPatInfo.Format("%s%s", "name:", imgHeader.name);
m_listboxpatinfo.AddString(strListBoxPatInfo);
strListBoxPatInfo.Format("%s%s", "birthday:", imgHeader.birth);
m_listboxpatinfo.AddString(strListBoxPatInfo);
strListBoxPatInfo.Format("%s%s", "sex:", imgHeader.sex);
m_listboxpatinfo.AddString(strListBoxPatInfo);
strListBoxPatInfo.Format("%s%s", "hospital:", imgHeader.original_hospital);
m_listboxpatinfo.AddString(strListBoxPatInfo);
strListBoxPatInfo.Format("%s%s", "machine type:", imgHeader.machine_type);
m_listboxpatinfo.AddString(strListBoxPatInfo);
//read 16bit raw image data
for(int i = 1;i <= nImgNum;i++)
{
 strFileName.Format("%s%s%d%s%d%s", strImgPath,"\\", nImgNo,"_",i,".img");
  if(fImg.Open(strFileName, CFile::modeRead|CFile::typeBinary|CFile::shareExclusive,   &fe)==NULL)
  {
      AfxMessageBox("Failed to read raw data!");
      return;
  }
```

```
    fImg.Seek(512,CFile::begin);
    for(int j = 0;j < 256;j++)
    {
        fImg.Read(usOrigData16[i-1][j],256*2);
    }
    fImg.Close();
  }
  ////////////////////////////////////////////////////////////////////////////
```

3 Interpolating between Two Successive Slices

We cannot directly overlay the sequence slices together to form a 3D image, because each scanning slice has a certain thickness, called slice thickness, for example, slice thickness is 3 mm. Between slices there is a slice interval, for example, slice interval is 2 mm. Scale factor indicates how many millimeters per pixel. So to form a real 3D image, we have to interpolate another slice between the two slices. Interpolating codes are as follows:

```
////////////////////////////////////////////////////////////////////////////
  void CImageData::ZoomInZLinear()
  {
    int frames ;
    double sh, sl ;
    int i,j,slice ;
    unsigned short x, y;
    imageSlices = iH.slices;
    frames=(int)((iH.slice_thickness*(float)imageSlices)/iH.scale_factor);//Theoretical frames
    if(frames > IMAGESIZE)  frames = IMAGESIZE ;
    for(int m = 0 ; m < frames ; m++)
    {
      sh = (double)(((double)(m)*iH.scale_factor)\
  /(double)iH.slice_thickness) ;
      slice = (int)sh ;
      if(slice >= imageSlices-1)
        return ;
      sl = sh - (int)sh ;
      sh = 1.0 - sl ;
      for(i = 0 ; i < iH.img_height ; i++)
        for(j = 0 ; j < iH.img_width ; j++)
        {
        x = unsigned short(float(m_usOrigData16[slice][i][j]&0x0fff)*sh +
float(m_usOrigData16[slice+1][i][j]&0x0fff)*sl);
        y = (unsigned short(float(m_usOrigData16[slice][i][j]>>12)*sh +
float(m_usOrigData16[slice+1][i][j]>>12)*sl + 0.5))<<12;
        m_usData3D16[m][i][j] = (x + y);
         }
        imageFrames = m ;//real frames
     }
     return;
  }
  ////////////////////////////////////////////////////////////////////////////
```

4 Adjusting Window Width/Center and Converting 16bit Raw Data to 8bit Display Data

Medical images cannot be displayed directly on the screen. To display them, you must map 12-bit raw

data into 8-bit display data, this mapping called window transformation.

Let x be a value of the original range, y is the corresponding value in the display range, ww and wl represents window width and window center, ym represents the maximum value in the display range. Window conversion formula is commonly as follows:

$$y(x)=\begin{cases} 0 & x < wl - ww/2 \\ \dfrac{ym}{ww}(x+\dfrac{ww}{2}-wl) & wl-\dfrac{ww}{2} < x < wl+\dfrac{ww}{2} \\ ym & x > wl+\dfrac{ww}{2} \end{cases} \quad \textbf{(1)}$$

Let top be the windows top then $top = wl + ww/2$, $bottom$ be the window bottom, $bottom = wl - ww/2$. Mapping from the raw data to display data can be expressed by the following (Fig. 1):

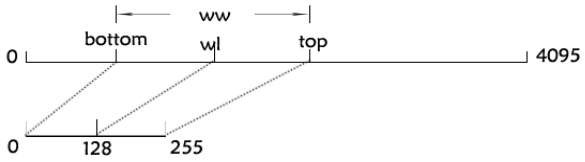


Fig. 1. Mapping from the raw data to display data

```
////////////////////////////////////////////////////////////////////////////
#define PIX_BITS                    12
#define D_HIGH                     255
#define D_LOW                      0
#define D_MID                  ((D_HIGH+D_LOW+1)/2)
#define FACTOR_H
      ((D_HIGH+1-D_MID) << PIX_BITS)
#define FACTOR_L
      ((D_MID-D_LOW) << PIX_BITS)
#define MIN_BOTTOM                      0
#define MAX_TOP                    4095
#define set_window_h(bot,top,fact,x) (x >= top) ? D_HIGH : \( (x <= bot) ? D_MID : ( (fact * (x -
bot)) >> PIX_BITS) + D_MID )
#define set_window_l(bot,top,fact,x) (x >= top) ? D_MID : \( (x <= bot) ? D_LOW : ( (fact * (x - bot))
>> PIX_BITS) + D_LOW )
void WindowTransform(int ww,int wl,unsigned short *win_table)
{
   int top = wl + ww/2 ;
   int bottom = wl - ww/2 ;
   if(top>MAX_TOP)
   {
      top = MAX_TOP ;
      bottom = MAX_TOP - ww;
   }
   if(bottom<MIN_BOTTOM)
   {
      bottom = MIN_BOTTOM ;
      top = ww ;
   }
   int scale_factorh, scale_factorl;
   if(top == wl && bottom == wl)
```

```
    {
      scale_factorh = FACTOR_H;
      scale_factorl = FACTOR_L;
    }
    else if(top == wl)
    {
      scale_factorh = FACTOR_H;
      scale_factorl = FACTOR_L / (wl - bottom) ;
    }
    else if(bottom == wl)
    {
      scale_factorh = FACTOR_H / (top - wl);
      scale_factorl = FACTOR_L;
    }
    else
    {
      scale_factorh = FACTOR_H / (top - wl);
      scale_factorl = FACTOR_L / (wl - bottom) ;
    }
    for(int i = 0 ; i <= wl ; i++)
      vi_win_table[i] = set_window_l(bottom, wl, scale_factorl, i);
    for(int i = wl ; i <= MAX_TOP ; i++)
    vi_win_table[i] = set_window_h(wl, top, scale_factorh, i);
}
////////////////////////////////////////////////////////////////////////////////
```

5 Making Three Orthogonal Planes (Transverse, Sagittal and Coronal)

Although we can get a real 3D image by interpolation of transverse planes, but doctors prefer to sagittal and coronal planes, these planes seem easier to understand (Fig. 2). So, we need multi-planner reformation (MPR), that is reconstructing sagittal and coronal planes from transverse plane (Fig. 3).

```
////////////////////////////////////////////////////////////////////////////
void CImageData::DisplayImage(CDC *pDC, int width, int height, int type,BOOL bCross)
{
  CImageData *pDataIF=CImageData::Instance();
  CRgn     rgn;
  int i ;
  unsigned char bmpBuf[IMAGESIZE*IMAGESIZE*3];
  unsigned char *ip ;
  ip = pDataIF->iBuf8 ;
  int x,y;
  if(type == TRAN)
  {
    ip = pDataIF->tranViewBuf8 ;
    x = iTop.y * width/256.0;
    y = iTop.x * width/256.0;
  }
  if(type == CORO)
  {
    ip = pDataIF->coroViewBuf8 ;
    x = iTop.y * width/256.0;
    y = (iTop.z+(256-imageFrames)/2) * width/256.0;
  }
  if(type == SAGI)
```

```
    {
      ip = pDataIF->sagiViewBuf8 ;
      x = iTop.x * width/256.0;
      y = (iTop.z+(256-imageFrames)/2) * width/256.0;
    }
    if(!bWindowColor)
      for(i = 0 ; i < IMAGESIZE*IMAGESIZE ; i++)
        bmpBuf[3*i] = bmpBuf[3*i+1] = bmpBuf[3*i+2] = ip[i] ;
    else
     for(i = 0 ; i < IMAGESIZE*IMAGESIZE ; i++)
      {
        bmpBuf[3*i] = pDataIF->psB[ip[i]] ;
        bmpBuf[3*i+1] = pDataIF->psG[ip[i]] ;
        bmpBuf[3*i+2] = pDataIF->psR[ip[i]] ;
      }
      long lImgLen;
      lImgLen = (long)IMAGESIZE*IMAGESIZE;
      LPBITMAPINFO Pbmi = (LPBITMAPINFO)new BYTE[sizeof(BITMAPINFOHEADER) + 256
* sizeof(RGBQUAD)];
      Pbmi->bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
      Pbmi->bmiHeader.biWidth = IMAGESIZE;
      Pbmi->bmiHeader.biHeight = (-1)*IMAGESIZE;
      Pbmi->bmiHeader.biPlanes = 1;
      Pbmi->bmiHeader.biBitCount = 24;
      Pbmi->bmiHeader.biCompression = 0;
      Pbmi->bmiHeader.biSizeImage = 0;
      Pbmi->bmiHeader.biXPelsPerMeter = 0;
      Pbmi->bmiHeader.biYPelsPerMeter = 0;
      Pbmi->bmiHeader.biClrUsed = 0;
      Pbmi->bmiHeader.biClrImportant = 0;
      ::StretchDIBits(pDC->GetSafeHdc(), 0,0, width,height, 0, 0, IMAGESIZE, IMAGESIZE,
bmpBuf, Pbmi, DIB_RGB_COLORS, SRCCOPY);
      delete []Pbmi;
      rgn.DeleteObject();
      if(bCross)
      {
      CPen MyNewPen0;
      MyNewPen0.CreatePen(PS_SOLID,1,RGB(0,255,0));
      CPen* pOriginalPen=pDC->SelectObject(&MyNewPen0);
       pDC->MoveTo(0,y);
       pDC->LineTo(height,y);
       pDC->MoveTo(x,0);
       pDC->LineTo(x,width);
       pDC->SelectObject(&pOriginalPen);
       }
  }
  ////////////////////////////////////////////////////////////////////////////
```

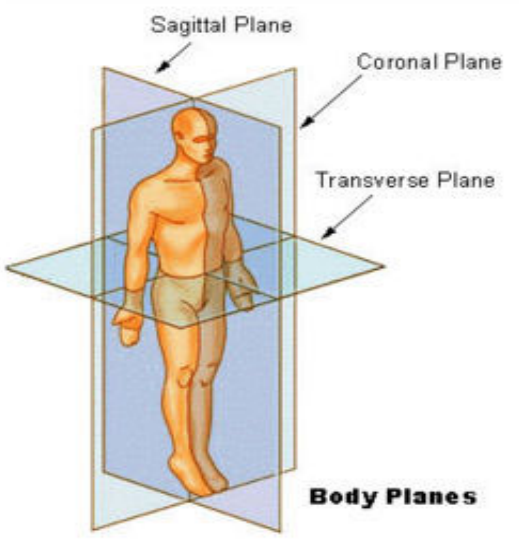


Fig. 2. Transverse, sagittal and coronal plane

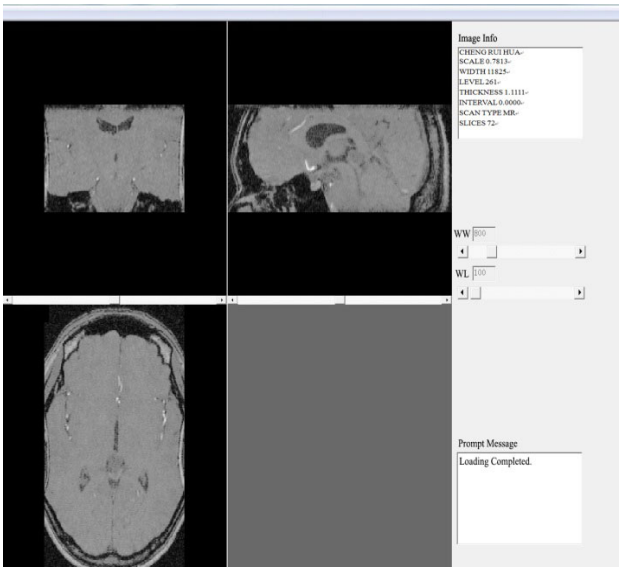


Fig. 3. Multi-planner reformation (MPR)

6 Making 3D Skin Points and Computing Their Normal Vectors

First, we look for the 3D points whose gray value is greater than the bottom of the window, then compute their normal vectors.

```
////////////////////////////////////////////////////////////////////////////
void CImageData::MakeSkinPoints()
{
 FILE *fp ;
 float sp, sm ;
 int i,j,k;
 int tP = 0 ;
 float nx,ny,nz,nl ;
 if(NULL!=skinPointsBuf)
  delete [] skinPointsBuf;
 skinPointsBuf = NULL;
 skinPoints = 0 ;
 for(i = 2 ; i < imageFrames-2 ; i++)
   for(k = 2 ; k < iH.img_width-2 ; k++)
     for(j = 2 ; j < iH.img_height-2 ; j++)
     {
        if(m_usData3D16[i][j][k] >= windowBottom+1)
        skinPoints ++ ;
     }
 skinPointsBuf = new DOT3D[skinPoints] ;
 fp = fopen("skpnew.txt", "w") ;
 fprintf(fp, "Real : %d\n", skinPoints) ;
 for(k = 2 ; k < imageFrames-2 ; k++)
   for(i = 2 ; i < iH.img_width-2 ; i++)
      for(j = 2 ; j < iH.img_height-2 ; j++)
      {
      if(m_usData3D16[k][j][i] >= windowBottom+1)
      {
         skinPointsBuf[tP].xp = (float)j;
```

```
            skinPointsBuf[tP].yp = (float)i;
            skinPointsBuf[tP].zp = (float)k ;
                 sp = m_usData3D16[k][j+1][i];
            sm = m_usData3D16[k][j-1][i];
            nx = 0.5 * (sm - sp);
            sp = m_usData3D16[k][j][i+1];
            sm = m_usData3D16[k][j][i-1];
            ny = 0.5 * (sm - sp);
            sp = m_usData3D16[k+1][j][i];
            sm = m_usData3D16[k-1][j][i];
            nz = 0.5 * (sm - sp);
            nl=(float)sqrt(nx*nx+ny*ny+nz*nz);
            if(nl > 0.01)
            {
               skinPointsBuf[tP].xn=(float)(nx/(1.0*nl));
               skinPointsBuf[tP].yn=(float)(ny/(1.0*nl));
               skinPointsBuf[tP].zn=(float)(nz/(1.0*nl));
            }
            else
            {
               skinPointsBuf[tP].xn=0.0;
               skinPointsBuf[tP].yn=0.0;
               skinPointsBuf[tP].zn=0.0;
            }
            tP++ ;
            if(tP >= skinPoints)
            {
               fprintf(fp,"break : %d\n",skinPoints) ;
               break ;
            }
             }
         }
         fclose(fp) ;
}

void CImageData::OpenGlDisplay(int mode,int dataset)
{
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        glClearColor(0.11f,0.63f,0.7f,0.78f);
        OpenGlPrepair(mode,dataset);
        glFinish();
        HDC hDC = wglGetCurrentDC();
        SwapBuffers(wglGetCurrentDC());
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
}

#define MINSZIE 16
void CImageData::OpenGlPrepair(int mode,int dataset)
{
        int i ;
    glDepthFunc(GL_LESS);
        glEnable(GL_DEPTH_TEST);
        glPushMatrix();
        glRotatef((GLfloat)glRotateX,1.0,0.0,0.0);
        glRotatef((GLfloat)glRotateY,0.0,1.0,0.0);
```

```
        glRotatef((GLfloat)glRotateZ,0.0,0.0,1.0);
        glTransZ = (-1)*imageFrames/2;
      glTranslatef((GLfloat)glTransX,(GLfloat)glTransY,(GLfloat)glTransZ) ;
        glGetDoublev(GL_MODELVIEW_MATRIX, modelMatrix);
      glGetDoublev(GL_PROJECTION_MATRIX, projMatrix);
      glGetIntegerv(GL_VIEWPORT, viewport);
        if(mode == 1)
          glPointSize((GLfloat)1.0*(GLfloat)(1.5+(iH.slice_thickness/iH.scale_factor)));
        if(mode == 2)
          glPointSize((GLfloat)1.5*(GLfloat)(1.5+(iH.slice_thickness/iH.scale_factor)));
        glBegin(GL_POINTS);
        glShadeModel(GL_FLAT);
        glColor4f(0.88f,0.68f,0.13f,0.28f) ;
        if(1 == dataset)
        {
          if(mode == 1)
                for(i=0;i<skinPoints;i++)
                {
                      glNormal3f(skinPointsBuf[i].xn,skinPointsBuf[i].yn,skinPointsBuf[i].zn);
                      glVertex3f( skinPointsBuf[i].xp,skinPointsBuf[i].yp,skinPointsBuf[i].zp);
                }
               if(mode == 2)
                for(i=0;i<skinPoints/MINSZIE;i++)
                {
  glNormal3f(skinPointsBuf[i*MINSZIE].xn,skinPointsBuf[i*MINSZIE].yn,skinPointsBuf[i*MINSZIE].zn);
                      glVertex3f(skinPointsBuf[i*MINSZIE].xp,
                      skinPointsBuf[i*MINSZIE].yp,
                      skinPointsBuf[i*MINSZIE].zp);
                 }
        }
        glEnd();
        glPopMatrix();
        glFinish();
  }

  void CDlgImage4::OnLButtonDown(UINT nFlags, CPoint point)
  {
        m_StartPoint = point;
        m_3DState = FREEHAND_3DSTATE;
        CDialog::OnLButtonDown(nFlags, point);
  }
  void CDlgImage4::OnMouseMove(UINT nFlags, CPoint point)
  {
        CImageData *pDataIF=CImageData::Instance();
        if(m_3DState == FREEHAND_3DSTATE)
        {
        Update3DWindowFromPoint(point,m_StartPoint,m_EndPoint,pDataIF->glRotateX,pDataIF-
>glRotateY,pDataIF->glRotateZ);
        pDataIF->OpenGlDisplay(2,1);
        }
        CDialog::OnMouseMove(nFlags, point);
  }
  BOOL CDlgImage4::Update3DWindowFromPoint(CPoint point, CPoint &m_StartPoint, CPoint
&m_EndPoint, double &X_angle, double &Y_angle, double &Z_angle)
  {
```

```
        m_EndPoint = point;
        CSize m_sizeOffset=m_EndPoint-m_StartPoint;
        m_StartPoint=m_EndPoint;
        CRect rect;
        GetClientRect(&rect);
        int nAngle;
        nAngle = ((CFushNavDlg*)AfxGetMainWnd())->GetRadio3DXYZ() + 1;
        if(1==nAngle)
        {
                if(abs(m_sizeOffset.cy)>abs(m_sizeOffset.cx))
                        X_angle+=double(m_sizeOffset.cy)/3.6;
                else if(abs(m_sizeOffset.cx)>abs(m_sizeOffset.cy))
                        if(point.y>(rect.Height()/2))
                                Y_angle+=double(m_sizeOffset.cx)/3.6;
                        else
                                Y_angle+=double(m_sizeOffset.cx*(-1.0))/3.6;
                else
                        Z_angle+=double(m_sizeOffset.cx)/3.6;
        }
        else if(2==nAngle)
        {
                if(abs(m_sizeOffset.cy)>abs(m_sizeOffset.cx))
                        if(point.x<(rect.Width()/2))
                                Y_angle+=double(m_sizeOffset.cy)/3.6;
                        else
                                Y_angle+=double(m_sizeOffset.cy*(-1.0))/3.6;
                else if(abs(m_sizeOffset.cx)>abs(m_sizeOffset.cy))
                        Z_angle+=double(m_sizeOffset.cx*(-1.0))/3.6;
                else
                        X_angle+=double(m_sizeOffset.cx)/3.6;
        }
        else if(3==nAngle)
        {
                if(abs(m_sizeOffset.cy)>abs(m_sizeOffset.cx))
                        X_angle+=double(m_sizeOffset.cy)/3.6;
                else if(abs(m_sizeOffset.cx)>abs(m_sizeOffset.cy))
                        Z_angle+=double(m_sizeOffset.cx*(-1.0))/3.6;
                else
                        Y_angle+=double(m_sizeOffset.cx)/3.6;
        }
        else
                ;
        return TRUE;
}
void CDlgImage4::OnLButtonUp(UINT nFlags, CPoint point)
{
        m_3DState = -1;
        CImageData *pDataIF=CImageData::Instance();
        Update3DWindowFromPoint(point,m_StartPoint,m_EndPoint,pDataIF->glRotateX,pDataIF-
>glRotateY,pDataIF->glRotateZ);
        pDataIF->OpenGlDisplay(1,1);
        CDialog::OnLButtonUp(nFlags, point);
}
////////////////////////////////////////////////////////////////////////////////
```

7 3D Surface Rendering Based on OpenGL

We use MFC in win7 and Studio Visual 2010, and set up a 3D display system (Fig. 4). The system includes three orthogonal planes (transverse, sagittal and coronal plane) and one 3D view. Dragging the 3D image with the mouse, it can rotate round x, y and z axis.

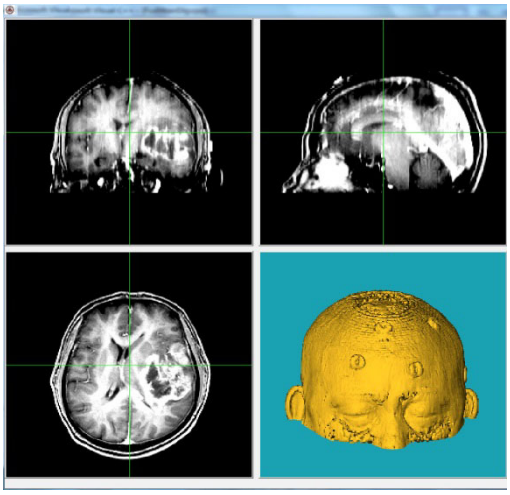


Fig. 4. 3D display system

8 Conclusions

3D construction of medical sequence image is a core technology, 3D surface rendering is complex and widely used in imaging device. In this paper, we describe 3D surface rendering based on OpenGL in detail, in order to speed up display, we reduce resolution when rotating and restore resolution when finished. So we can rotate the 3D image in real time (Fig. 5).

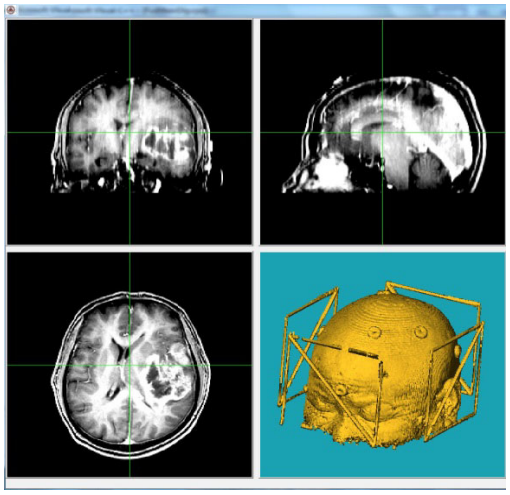


Fig. 5. 3D image rotate in real time

In the future work we will carry out 3D volume rendering, improve rendering speed, and realize 3D transparent or translucent display

References

[1] L. Liu, Z.H. Jia, J. Yang, K. Nikola, A medical image enhancement method using adaptive thresholding in NSCT domain combined unsharp masking, International Journal of Imaging Systems & Technology 25(3)(2015) 199-205.

[2] W.Z.W. Ismail, K.S. Sim, Contrast enhancement dynamic histogram equalization for medical image processing application, International Journal of Imaging Systems & Technology 21(3)(2011) 280-289.

[3] F.N. Kiwanuka, M.H. Wilkinson, Automatic attribute threshold selection for morphological connected attribute filters, Pattern Recognition 53(2016) 59-72.

[4] M. Rodrigo, S. Örjan, Gradient-based enhancement of tubular structures in medical images, Medical Image Analysis 26(1)(2015) 19-29.

[5] H.-T. Wu, J.-W. Huang, Y.-Q. Shi, A reversible data hiding method with contrast enhancement for medical images, Journal of Visual Communication & Image Representation 31(2015) 146-153.

[6] C. Hariton, Recent trends in medical image processing, Computer Science Journal of Moldova 2(2)(2014) 147-154.

[7] J.-J. Wang, Z.-H. Jia, K. Nikola, Medical image enhancement algorithm based on NSCT and the improved fuzzy contrast, International Journal of Imaging Systems & Technology 25(1)(2015) 7-14.

[8] E. Anders, D. Paul, F. Daniel, L. Stephen, Medical image processing on the GPU-past, Medical Image Analysis 17(8)(2013) 1073-1094.

[9] K. Dagmar, V. Michal, H. Andreas, T. Thomas, Simplified implementation of medical image processing algorithms into a grid using a workflow management system, Future Generation Computer Systems 26(4)(2010) 681-684.

[10] M.S. Ehsan, A.-A. Alireza, R. Roohollah, F.-R., Shahrooz, Taimouri, Web-based interactive 2D/3D medical image processing and visualization software, Computer Methods & Programs in Biomedicine 98(2)(2010) 172-182.

[11] A. Sam, N. George, L. Sergio, Stereotactic navigation for TAMIS-TME: opening the gateway to frameless, image-guided abdominal and pelvic surgery, Surgical Endoscopy 29(1)(2015) 207-211.

[12] X.Y. Sun, Y.K. Yoon, J. Li, F.D. McKenzie, Automated image-guided surgery for common and complex dental implants, Journal of Medical Engineering & Technology 38(5)(2014) 251-259.

[13] T.P. Kingham, M.A. Scherer, B.W. Neese, L.W. Clements, J.D. Stefansic, W.R. Jarnagin, Image-guided liver surgery: intraoperative projection of computed tomography images utilizing tracked ultrasound, HPB: The Official Journal of the International Hepato Pacreato Biliary Association 14(9)(2012) 594-603.

[14] T. Elserry, H. Anwer, I.N. Esene, Image guided surgery in the management of craniocerebral gunshot injuries, Surgical Neurology International 4(6)(2013) S448-S454.

[15] M. Nau-Hermes, R. Schmitt, M. Becker, W. El-Hakimi, S. Hansen, T. Klenzner, Quality assurance of multiport image-guided minimally invasive surgery at the lateral skull base, BioMed Research International. <https://www.hindawi.com/journals/bmri/2014/904803/2014>, 2014.

[16] G. Li, H. Su, G.A. Cole, W.J. Shang, K. Harrington, A. Camilo, J.G. Pilitsis, G.S. Fischer, Robotic system for MRI-guided stereotactic neurosurgery, IEEE Transactions on Biomedical Engineering 62(4)(2015) 1077-1088.

[17] L.M. Vigneron, R.C. Boman, J.P. Ponthot, P.A. Robe, S.K. Warfield, J.G. Verly, Enhanced FEM-based modeling of brain shift deformation in image-guided neurosurgery, Journal of Computational & Applied Mathematics 234(7)(2010) 2046-2053.

[18] J.H. Yan, J.C-S. Lim, D.W. Townsend, MRI-guided brain PET image filtering and partial volume correction, Physics in Medicine & Biology 60(3)(2015) 1-7.