MITK

MEDICAL IMAGING TOOLKIT

MITK TUTORIAL

MITK Tutorial

Medical Image Processing Group, Institute of Automation, Chinese Academy of Sciences
Web site: http://www.mitk.net
Corresponding person: Professor Jie Tian
E-mail: tian@doctor.com

Chapter 1

Surface Reconstruction Using MITK

This chapter will use an example to demonstrate how to use MITK to proceed surface reconstruction. This example takes Microsoft Visual Studio 6.0 as development environment.

Classes Used in Surface Reconstruction

mitkVolume： mitkVolume is the most important data object in MITK. It represents a volume composed by a series of slices. All the image data need to be processed must be converted into the form of mitkVolume first. Include mitkVolume.h to use this class and see MITK help documents to get detailed information about this class.

mitkMesh: mitkMesh is also a data object which represent a 3-dimensional geometrical data object, i.e. the surface representation of the object generated by surface reconstruction. Include mitkMesh.h to use this class and see MITK help documents to get detailed information about this class.

mitkSurfaceModel: mitkSurfaceModel is a data model which represents the rendering model of the data object in a 3-dimensional scene. mitkSurfaceModel represents the rendering model of mitkMesh. Include mitkSurfaceModel.h to use this class and see MITK help documents to get detailed information about this class.

mitkRawReader: mitkRawReader is a volume reader object. It reads data from a raw file and fill it into a volume. The reconstruction process will get volume data from it. Include mitkRawReader.h to use this class and see MITK help documents to get detailed information about this class.

mitkMarchingCubes: mitkMarchingCubes is a filter which represents an algorithm object. It encapsulates the algorithm object for surface reconstruction. We will use this algorithm called “Marching Cubes” to reconstruct the surface from the volume data. Include mitkMarchingCubes.h to use this class and see MITK help documents to get detailed information about this class.

mitkView: mitkView is a view encapsulates a window on screen. It has the ability of displaying multi-models and supports the interactive manipulation of the mouse. Include mitkView.h to use this class and see MITK help documents to get detailed information about this class.

Example for Surface Reconstruction

First let's see the functions achieved by this example. The main user interface is shown as follows.

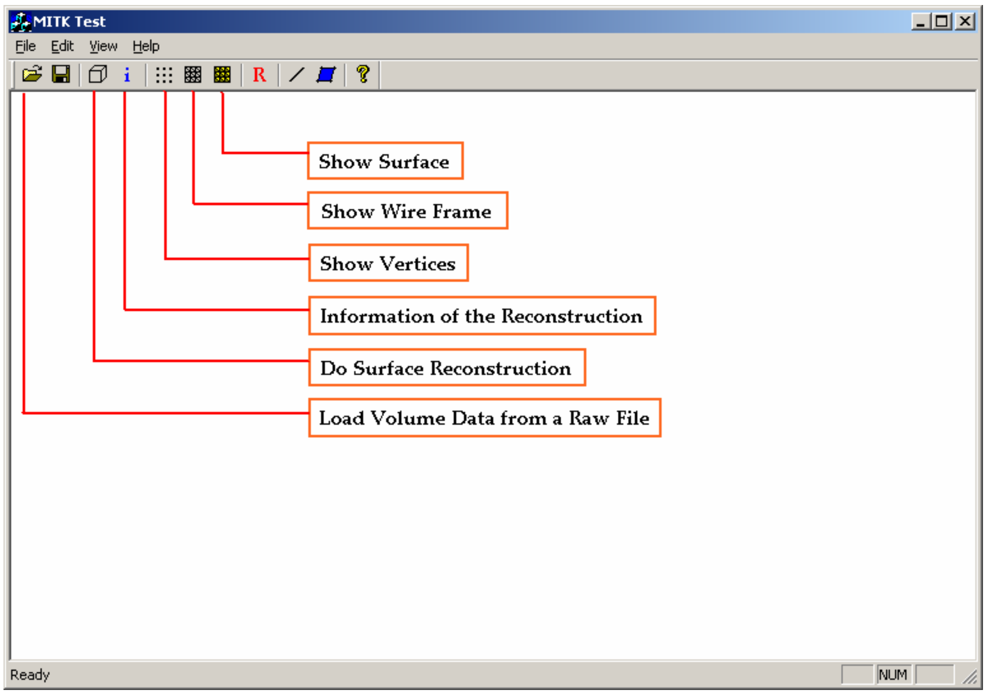

MITK Test
File
Edit
View
Help
Show Surface
Show Wire Frame
Show Vertices
Information of the Reconstruction
Do Surface Reconstruction
Load Volume Data from a Raw File
Ready
NUM

Fig. 1 Main User Interface

Now let's do surface reconstruction step by step.

Step 1: Load volume data from a raw file.

Click 📂.

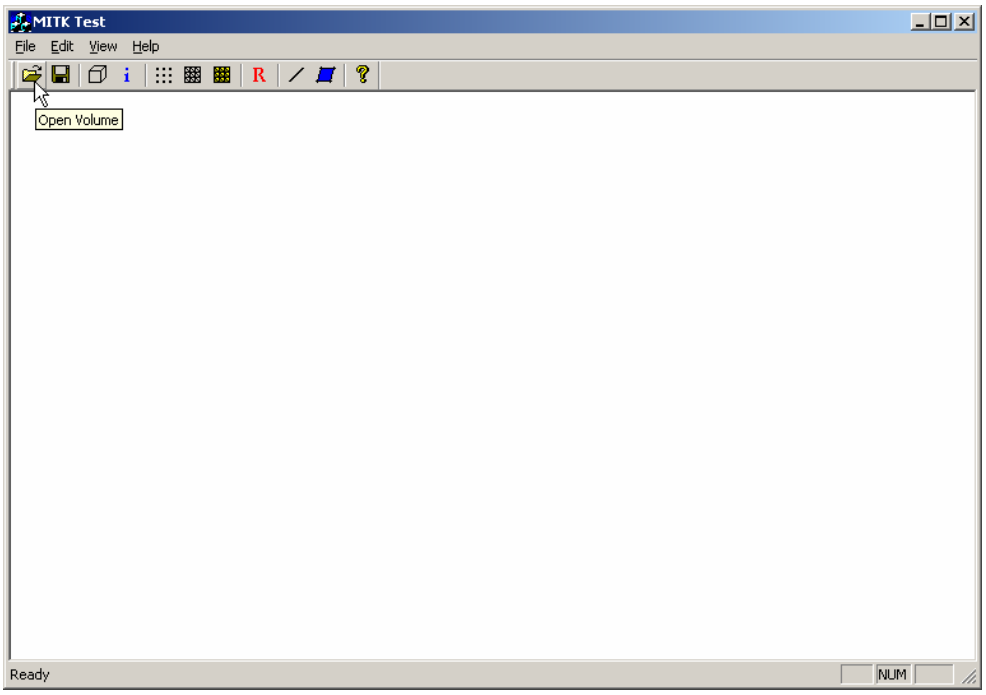
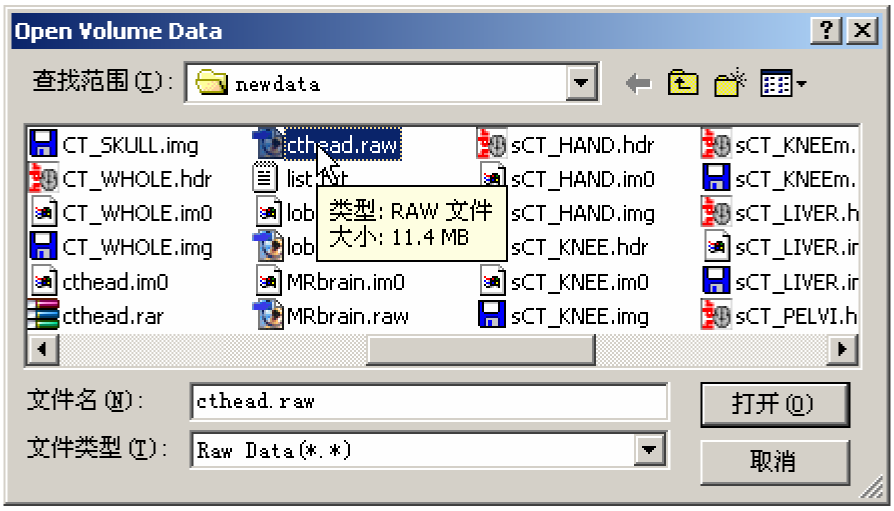


Fig. 2 Load Volume Data

Select the raw file to open.



newdata

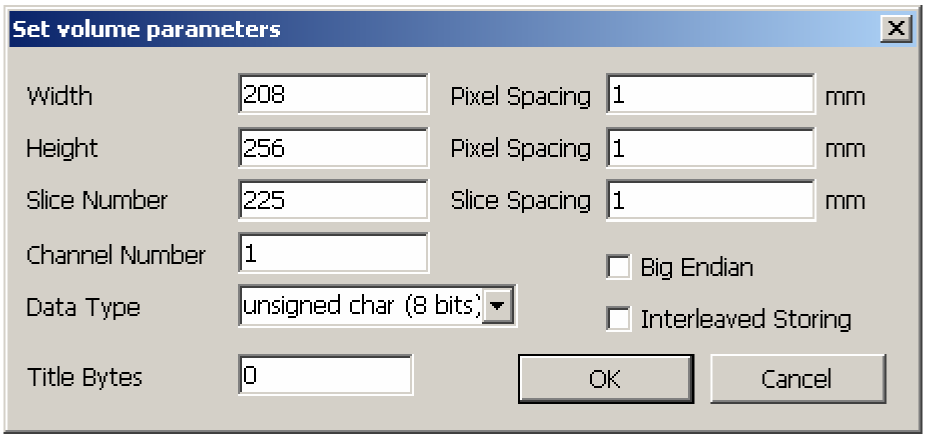
CT_SKULL.img
CT_WHOLE.hdr
CT_WHOLE.im0
CT_WHOLE.img
cthead.im0
cthead.rar
cthead.raw
list
lob
lob
MRbrain.im0
MRbrain.raw
sCT_HAND.hdr
sCT_HAND.im0
sCT_HAND.img
sCT_KNEE.hdr
sCT_KNEE.im0
sCT_KNEE.img
sCT_KNEEm.
sCT_KNEEm.
sCT_LIVER.h
sCT_LIVER.ir
sCT_LIVER.ir
sCT_PELVI.h

类型: RAW 文件
大小: 11.4 MB

文件名 (N): cthead.raw
文件类型 (T): Raw Data (*.*)

Fig. 3 Select Raw File

Because the raw file has no information about the data it contains, you must enter the volume parameters into the dialog shown after you clicked the “Open” button in open file dialog as follows.


Set volume parameters

Width	208	Pixel Spacing	1	mm
Height	256	Pixel Spacing	1	mm
Slice Number	225	Slice Spacing	1	mm
Channel Number	1		☐ Big Endian	
Data Type	unsigned char (8 bits)		☐ Interleaved Storing	
Title Bytes	0	OK	Cancel	

Fig. 4 Set Volume Parameters

When you have entered all the information, click OK to load the volume. If no error occurs, the volume data will be loaded to memory successfully after a while.

Step 2: Surface reconstruction.

Click .

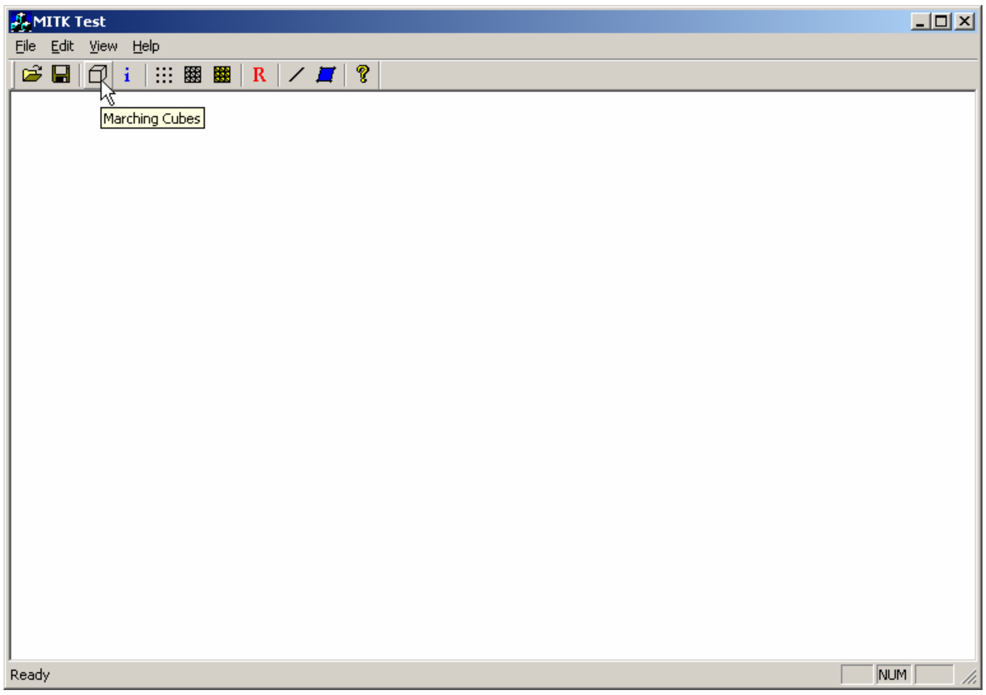
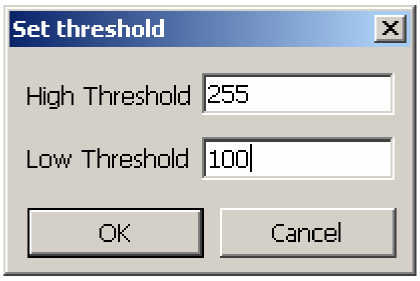


Fig. 5 Surface Reconstruction

Enter the low threshold and high threshold.


Set threshold

High Threshold 255

Low Threshold 100

Fig. 6 Enter Thresholds

The choice of thresholds lies on the physical characteristics of the object whose surface is to be extracted. It needs some priori knowledge. Click OK after the proper thresholds have been entered. If no error occurs, the result will be shown in the client area of the main window after a while. You can use mouse to manipulate the surface model:

Keep left button down and drag on the view to rotate the model;

Keep middle button down and drag on the view to move the model;

Keep right button down and drag on the view to zoom in or zoom out.

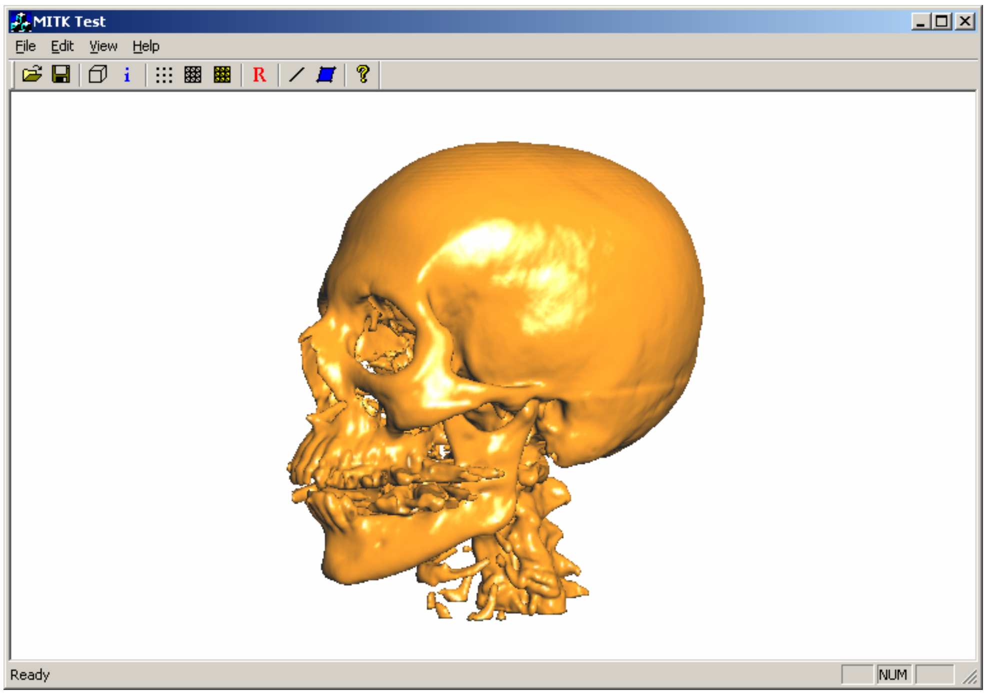


Fig. 7 The Surface Reconstruction Result

Click to show the model in points;

Click to show the model in wire frame;

Click ▦ to show the surface.

Now let's see how easy to build this example with MITK.

Step 1: New a project named "MitkTest" in Visual C++ 6.0.

Step 2: Set up the UI resources. You can copy them from the source codes of the example.

Step 3: Add two members to CMitkTestDoc class (in MitkTestDoc.h) to register the data objects and set them to NULL in constructor (in MitkTestDoc.cpp). And add a double member to record the time used by surface reconstruction and set it to 0 in constructor. Don't forget including necessary header files.

```
// MitkTestDoc.h : interface of the CMitkTestDoc class

#include "mitkVolume.h"
#include "mitkMesh.h"

class CMitkTestDoc : public CDocument
{
        … … …
        mitkVolume *m_Volume;
        mitkMesh   *m_Mesh;

        double m_TimeUsed;
        … … …
};


CMitkTestDoc::CMitkTestDoc()
{
        // TODO: add one-time construction code here
        m_Volume = NULL;
        m_Mesh = NULL;

        m_TimeUsed = 0;
}
```

Step 4: Add two members to CMitkTestView class (in MitkTestView.h). m_Surface represent the rendering model of the mesh data and set them to NULL in constructor (in MitkTestView.cpp). m_View is used to show the model. Don't forget including necessary header files.

```
// MitkTestView.h : interface of the CMitkTestView class
```

```
#include "mitkSurfaceModel.h"
#include "mitkMesh.h"
#include "mitkView.h"

class CMitkTestView : public CView
{
        … … …
        void SetMeshData(mitkMesh* mesh);

        mitkSurfaceModel *m_Surface;
        mitkView *m_View;
        … … …
};

CMitkTestView::CMitkTestView()
{
        // TODO: add construction code here
        m_View = NULL;
        m_Surface = NULL;

}
```

The function SetMeshData() just calls the SetData() function in order to encapsulate m_Surface. See it as follows.

```
void CMitkTestView::SetMeshData(mitkMesh* mesh)
{
        m_Surface->SetData (mesh);
}
```

Step 5: Add message handlers to handle the commands and two private methods to clear mesh and volume data in CMitkTestDoc.

```
afx_msg void OnFileOpenVolume();
afx_msg void OnReconstruction();
afx_msg void OnInfo();

void clearVolume();
void clearMesh();
```

The codes are shown as follows.

```
// Message handler for loading volume data.
void CMitkTestDoc::OnFileOpenVolume()
```

```
{
        // TODO: Add your command handler code here
        // Show open file dialog.
        COpenFileDialog fileDialog;
        fileDialog.SetTitle("Open Volume Data");
        fileDialog.SetFilter("Raw Data(*.*)\0*.*\0\0");

        // If user select a raw file and press "OK".
        if(fileDialog.Run())
        {
                int nFilterIndex = fileDialog.GetFilterIndex();
                CString szFileName = fileDialog.GetPathName();

                // Read raw data from the file and generate a volume.
                CRawSetDlg dlg;
                if (dlg.DoModal() == IDOK)
                {
                        // Use mitkRawReader to generate volume from raw
                        // data file.
                        mitkRawReader *reader = new mitkRawReader;

                        // Set information about the raw data to the reader.
                        reader->SetWidth(dlg.m_Width);
                        reader->SetHeight(dlg.m_Height);
                        reader->SetImageNum(dlg.m_ImageNum);

                        reader->SetSpacingX(dlg.m_SpacingX);
                        reader->SetSpacingY(dlg.m_SpacingY);
                        reader->SetSpacingZ(dlg.m_SpacingZ);

                        reader->SetChannelNum(dlg.m_ChannelNum);
                        reader->SetDataType(dlg.m_DataType);
                        reader->SetEndian(dlg.m_IsBigEndian);
                        reader->SetPlanarCfg(dlg.m_IsColorByPlane);

                        // Set name of the file to read.
                        reader->AddFileName(szFileName);

                        // Run reading process.
                        reader->Run();

                        // Clear the old volume.
                        this->clearVolume();

                        // Get the volume generated by this reader.
                        m_Volume = reader->GetOutput();
                        // To keep the volume data in main program.
                        m_Volume->AddReference();

                        // Delete the mitkRawReader object.
                        reader->Delete();
                }
        }
}

// Message handler for surface reconstruction.
void CMitkTestDoc::OnReconstruction()
{
        // TODO: Add your command handler code here
        CThresholdDlg dlg;

        // Show the dialog to get two thresholds.
        if (dlg.DoModal() == IDOK)
        {
                // Use mitkMarchingCubes to proceed surface reconstruction
                mitkMarchingCubes *mc = new mitkMarchingCubes;

                // Set thresholds and input volume.
                mc->SetThreshold(dlg.m_LowValue, dlg.m_HighValue);
                mc->SetInput(m_Volume);

                // Record the start time of the reconstruction process.
                clock_t start = clock();

                // Run the process.
                mc->Run();

                // Record the finish time of the reconstruction process.
                clock_t finish = clock();

                // Calculate the time used by the reconstruction process.
                m_TimeUsed = (double)(finish - start) /
CLOCKS_PER_SEC;

                // Get the pointer of MitkTestView
                POSITION pos = this->GetFirstViewPosition();
                CMitkTestView *view = (CMitkTestView *)(this-
>GetNextView(pos));

                // Clear the old mesh data.
                view->SetMeshData(NULL);
                this->clearMesh();
                // Register the new mesh data.
                m_Mesh = mc->GetOutput();
                m_Mesh->AddReference();

                // Delete the mitkMarchingCubes object.
                mc->Delete();

                // Show information about the reconstruction result.
                this->OnInfo();

                // Send mesh to the view to display.
                view->SetMeshData(m_Mesh);

                UpdateAllViews(NULL);
        }
}

// Show information about the reconstruction result.
void CMitkTestDoc::OnInfo()
{
        // TODO: Add your command handler code here
        if (!m_Mesh) return;

        // Get total triangle number of triangles and the memory used in
        // bytes
        unsigned long triNum = m_Mesh->GetFaceNumber();
        unsigned long memSize = m_Mesh->GetActualMemorySize();

        CString msg;
        msg.Format("Time:%f s\nTriangles:%d\nTriangles per
sec.:%f\nMesh Memory Size:%d Bytes",
                    m_TimeUsed, triNum, triNum/m_TimeUsed, memSize);
        ::AfxMessageBox(msg);
}

// Clear mesh data.
void CMitkTestDoc::clearMesh()
{
        if (m_Mesh)
        {
                m_Mesh->RemoveReference();
                m_Mesh = NULL;
        }
}

// Clear volume data.
void CMitkTestDoc::clearVolume()
```

```
{
        if (m_Volume)
        {
                m_Volume->RemoveReference();
                m_Volume = NULL;
        }
}
```

Note:
i. The destructors of all classes derived from mitkObject in MITK are protected. So you can not create local object in stack, and can not use delete operator to delete the object created by new operator. Please call Delete() function to delete the object if you really want to.
ii. MITK uses reference counting mechanism to track the number of references to an object. So if you create MITK objects and set them to some of the MITK processes, you do not need to delete them. MITK will take care of all the objects. But if you want to keep an object (e.g. m_Volume and m_Mesh in this example) alive during the whole running time of your program, please call AddReference() of the object immediately when the object has been created, and call RemoveReference() when you want to delete the object. In other cases (e.g. the object of mitkMarchingCubes mc in OnReconstruction() above), don't call this two functions and call Delete() to delete the object if it has nothing to do with any MITK processes.

Step 6: Add message handlers to handle WM_CREATE and WM_SIZE messages so as to make mitkView fill the client area of the main window. The codes are shown as follows.

```
int CMitkTestView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CView::OnCreate(lpCreateStruct) == -1)
                return -1;

        // TODO: Add your specialized creation code here
        // Get the size of client area.
        RECT clientRect;
        this->GetClientRect(&clientRect);
        int wWidth = clientRect.right - clientRect.left;
        int wHeight = clientRect.bottom - clientRect.top;

        // Create mitkView and set its position and size.
        m_View = new mitkView;
        m_View->SetParent(GetSafeHwnd());
        m_View->SetLeft(0);
        m_View->SetTop(0);
        m_View->SetWidth(wWidth);
        m_View->SetHeight(wHeight);
        // Show mitkView.
        m_View->Show();

        // Build surface model to contain mesh data.
        m_Surface = new mitkSurfaceModel;

        // Set the material properties of the surface model.
        // You can also use default values.
        m_Surface->GetProperty()->SetAmbientColor(0.75f, 0.75f, 0.75f, 1.0f);
        m_Surface->GetProperty()->SetDiffuseColor(1.0f, 0.57f, 0.04f, 1.0f);
        m_Surface->GetProperty()->SetSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
        m_Surface->GetProperty()->SetSpecularPower(100.0f);
       m_Surface->GetProperty()->SetEmissionColor(0.0f, 0.0f, 0.0f, 0.0f);

        // Add this surface model to the view.
        m_View->AddModel(m_Surface);

        return 0;
}

void CMitkTestView::OnSize(UINT nType, int cx, int cy)
{
        CView::OnSize(nType, cx, cy);

        // TODO: Add your message handler code here
        // Set the position and size of the view
        // to make it fill the client area.
        m_View->SetLeft(0);
        m_View->SetTop(0);
        m_View->SetWidth(cx);
        m_View->SetHeight(cy);

        // Update the view.
        m_View->Update();
}
```

Step 7: The last step is to release the memory. Call clearMesh() and clearVolume() in the destructor of the CMitkTestDoc class to delete m_Mesh and m_Volume. Call Delete() of m_View in the destructor of the CMitkTestView class to delete m_View.

The destructor of CMitkTestDoc:

```
CMitkTestDoc::~CMitkTestDoc()
{
        this->clearMesh();
```

```
	this->clearVolume();
}
```

The destructor of CMitkTestView:

```
CMitkTestView::~CMitkTestView()
{
	if (m_View)
	{
		m_View->Delete();
		m_View = NULL;
	}
}
```

Note: When you called m_View->AddModel(m_Surface) in OnCreate() function of CMitkTestView, MITK takes control of m_Surface. So it will be deleted in m_View. Don’t delete it in the destructor of CMitkTestView.