

8/25/2010

# VTK Debug Leaks

Diana Wald



GERMAN  
CANCER RESEARCH CENTER  
IN THE HELMHOLTZ ASSOCIATION

- Memory leaks occur when a section of code allocates a block of memory that is never reclaimed.

```
vtkRenderWindow* vtkRenWin = vtkRenderWindow::New();  
  
mitk::VtkPropRenderer::Pointer br = mitk::VtkPropRenderer::New( "testingBR" , vtkRenWin,  
myRenderingManager);  
  
mitk::BaseRenderer::AddInstance (vtkRenWin,br);  
  
myRenderingManager->AddRenderWindow (vtkRenWin);  
  
. . . .  
  
myRenderingManager->RemoveRenderWindow (vtkRenWin);  
  
vtkRenWin->Delete();
```



One way to create a VTK object is:

```
vtkObject* myObject = new vtkObject;
```

**But** you must manually delete this object, otherwise it leads to a memory leak.

```
myObject->Delete();
```

- All objects in VTK are reference counted
- VTK automatically delete an object when it is no longer needed (reference count of an object falls to zero)
- VTK does with via calls to `New()` / `Delete()` and `Register()` / `UnRegister()` to increase and decrease the number of references to an object

Advantage of reference counting in VTK is that it allows data to be shared instead of duplicated.

- With SmartPointers, the reference count of an object is automatically increased when assigned to a SmartPointer and automatically decreased when unassigned from a SmartPointer.
- Objects are unassigned from a SmartPointer whenever that SmartPointer is assigned to another object or to 0. The latter occurs automatically whenever a SmartPointer goes out of scope.

```
#include <vtkSmartPointer>  
vtkSmartPointer<vtkTransform> transform = vtkSmartPointer<vtkTransform>::New();
```

No Delete() needs to be called. However, be careful and note that we are allocating vtkObjects and NOT just vtkSmartPointer.

Never assign a raw object pointer to a smart pointer:

```
vtkSmartPointer<vtkTransform> transform = vtkTransform::New();
```



Reference count is increment and requiring an explicit Delete later

- Compile VTK with VTK\_DEBUG\_LEAKS switched on

```

VLI_LIBRARY_FOR_VP1000      VLI_LIBRARY_FOR_VP1000-NOTFOUND
VTK_DATA_ROOT              VTK_DATA_ROOT-NOTFOUND
VTK_DEBUG_LEAKS            
VTK_GLEXT_FILE             V:/windows/source/VTK542/Utilities/ParseOGLExt/headers/glxt.h
VTK_GLXEXT_FILE            V:/windows/source/VTK542/Utilities/ParseOGLExt/headers/glxext.h
VTK_INSTALL_QT_PLUGIN_DIR  ${CMAKE_INSTALL_PREFIX}${VTK_INSTALL_QT_DIR}
VTK_LEGACY_REMOVE          
VTK_LEGACY_SILENT          
VTK_MATERIALS_DIRS         V:/windows/x64/VTK-5.4.2_Qt460_VC9.0_DebugLeaks_Bin/Utilities/MaterialLibrary/Repository

```

- Start / Run Application or Test
- Output of all objects which were not deleted including information about the name of the vtkObject and the number of instances.

```

Class "vtkCellData" has 2 instances still around.
Class "vtkInformationIntegerVectorValue" has 3 instances still around.
Class "vtkInformationVector" has 9 instances still around.
Class "vtkPointData" has 2 instances still around.
Class "vtkTrivialProducer" has 1 instance still around.
Class "vtkPoints" has 9 instances still around.
Class "vtkInformation" has 15 instances still around.
Class "vtkLine" has 4 instances still around.
Class "vtkInformationStringVectorValue" has 1 instance still around.
Class "vtkInformationIntegerPointerValue" has 1 instance still around.
Class "vtkPolyData" has 1 instance still around.
Class "vtkPixel" has 2 instances still around.
Class "vtkIdList" has 8 instances still around.
Class "vtkDoubleArray" has 9 instances still around.
Class "vtkAlgorithmOutput" has 2 instances still around.
Class "vtkInformationIntegerValue" has 42 instances still around.
Class "vtkExtentTranslator" has 1 instance still around.
Class "vtkCellArray" has 2 instances still around.
Class "vtkVertex" has 1 instance still around.
Class "vtkImageData" has 1 instance still around.
Class "vtkInformationStringValue" has 2 instances still around.
Class "vtkFloatArray" has 3 instances still around.
Class "vtkInformationExecutivePortVectorValue" has 2 instances still around.
Class "vtkContourValues" has 1 instance still around.
Class "vtkIdTypeArray" has 2 instances still around.
Class "vtkShortArray" has 1 instance still around.
Class "vtkVoxel" has 1 instance still around.
Class "vtkFieldData" has 2 instances still around.
Class "vtkInformationDoubleVectorValue" has 2 instances still around.
Class "vtkStreamingDemandDrivenPipeline" has 2 instances still around.
Class "vtkMergePoints" has 1 instance still around.
Class "vtkMarchingCubes" has 1 instance still around.
Class "vtkInformationExecutivePortValue" has 2 instances still around.

```

- Comment part of the code to find the position of the vtkDebugLeak

```
m_ResultNode->SetProperty("volumerendering",  
                          mitk::BoolProperty::New(false));  
  
vtkMarchingCubes* surfaceCreator = vtkMarchingCubes::New();  
surfaceCreator->SetInput(m_ResultImage->GetVtkImageData());  
surfaceCreator->SetValue(0, 1);  
  
mitk::Surface::Pointer surface = mitk::Surface::New();  
surface->SetVtkPolyData(surfaceCreator->GetOutput());  
  
mitk::DataNode::Pointer surfaceNode = mitk::DataNode::New();  
surfaceNode->SetData(surface);  
  
m_DataStorage->Add(surfaceNode);  
  
mitk::RenderingManager::GetInstance()->RequestUpdateAll();
```



```
m_ResultNode->SetProperty("volumerendering",  
                          mitk::BoolProperty::New(false));  
  
vtkMarchingCubes* surfaceCreator = vtkMarchingCubes::New();  
// surfaceCreator->SetInput(m_ResultImage->GetVtkImageData());  
// surfaceCreator->SetValue(0, 1);  
  
// mitk::Surface::Pointer surface = mitk::Surface::New();  
// surface->SetVtkPolyData(surfaceCreator->GetOutput());  
  
// mitk::DataNode::Pointer surfaceNode = mitk::DataNode::New();  
// surfaceNode->SetData(surface);  
//  
// m_DataStorage->Add(surfaceNode);  
  
//mitk::RenderingManager::GetInstance()->RequestUpdateAll();
```



```
m_ResultNode->SetProperty("volumerendering",  
                          mitk::BoolProperty::New(false));  
  
// vtkMarchingCubes* surfaceCreator = vtkMarchingCubes::New();  
// surfaceCreator->SetInput(m_ResultImage->GetVtkImageData());  
// surfaceCreator->SetValue(0, 1);  
  
// mitk::Surface::Pointer surface = mitk::Surface::New();  
// surface->SetVtkPolyData(surfaceCreator->GetOutput());  
  
// mitk::DataNode::Pointer surfaceNode = mitk::DataNode::New();  
// surfaceNode->SetData(surface);  
//  
// m_DataStorage->Add(surfaceNode);  
  
//mitk::RenderingManager::GetInstance()->RequestUpdateAll()
```



```
m_ResultNode->SetProperty("volumerendering",  
                          mitk::BoolProperty::New(false));  
  
vtkMarchingCubes* surfaceCreator = vtkMarchingCubes::New();  
surfaceCreator->SetInput(m_ResultImage->GetVtkImageData());  
surfaceCreator->SetValue(0, 1);  
  
mitk::Surface::Pointer surface = mitk::Surface::New();  
surface->SetVtkPolyData(surfaceCreator->GetOutput());  
  
mitk::DataNode::Pointer surfaceNode = mitk::DataNode::New();  
surfaceNode->SetData(surface);  
  
m_DataStorage->Add(surfaceNode);  
  
mitk::RenderingManager::GetInstance()->RequestUpdateAll();  
surfaceCreator->Delete();
```



- Check reference count of an object

The screenshot shows a debugger's Autos window with a tree view on the left and a table of variables on the right. The tree view shows a hierarchy starting with 'surfaceCreator', which contains 'vtkPolyDataAlgorithm', which contains 'ContourValues', which contains 'vtkObject'. The 'vtkObject' node is expanded to show 'vtkObjectBase', which in turn has 'ReferenceCount' highlighted in blue. The table on the right shows the following data:

Name	Value	Type
surfaceCreator	0x0000000008204370 {ContourValues=0x0000000008281480 ComputeNormals=1 ComputeGradi...	vtkMarch
vtkPolyDataAlgorithm	{...}	vtkPolyDi
ContourValues	0x0000000008281480 {Contours=0x000000000822cfa0 }	vtkConto
vtkObject	{Debug=0 MTime={...} SubjectHelper=0x0000000000000000 }	vtkObjec
vtkObjectBase	{ReferenceCount=1 }	vtkObjec
__vfptr	0x000007fef68c1048 const vtkContourValues::`vftable'	*
ReferenceCount	1	int
Debug	0	unsigned
MTime	{ModifiedTime=43252 }	vtkTimeS
SubjectHelper	0x0000000000000000	vtkSubje
Contours	0x000000000822cfa0	vtkDoubl
ComputeNormals	1	int
ComputeGradients	0	int
ComputeScalars	1	int
Locator	0x0000000000000000	vtkPointL
this	0x0000000000aff8e0	Step7 * c

- Use Valgrind (<http://valgrind.org/>)

- VTK SmartPointer:

<http://www.itk.org/Wiki/VTK/Tutorials/SmartPointers>