STL file format MeshIO class for ITK

Release 1.00

Luis Ibanez[1]

January 5, 2014

[1]Kitware Inc. Clifton Park, NY

Abstract

This document describes the implementation of an ITK class to support the reading and writing of Meshes in STL file format. The Meshes are assumed to contain 2D manifolds embedded in a 3D space. In practice, it would be desirable to use this class mostly to read and write QuadEdgeMeshes.

This paper is accompanied with the source code, input data, parameters and output data that the authors used for validating the algorithm described in this paper. This adheres to the fundamental principle that scientific publications must facilitate reproducibility of the reported results.

Contents

1 Introduction **2**

2 Software Requirements **2**

3 Infrastructure **2**

4 How to Use **3**

5 Examples **4**
5.1 ASCII 5
5.2 BINARY 5
5.3 Screenshots 5

6 Conclusion **8**

1 Introduction

The STL file format is a very common standards for the transmission of Mesh data. Originally designed for Stereo Lithography, the STL format has become a widely used standard for storing and sharing mesh data.

http://en.wikipedia.org/wiki/STL_(file_format)

In recent years, the STL file format has been adopted as the standard way of preparing shapes as input to 3D printing. Given that ITK is a natural choice for segmenting anatomical structures from 3D medical data, it is desirable to be able to generate meshes from such structures, and then store them in STL files suitable for 3D printing. The class described in this article will make that possible, using a pure ITK pipeline.

2 Software Requirements

You need to have the following software installed:

- Insight Toolkit 4.5
- CMake 2.8

For convenience, the code has been configured as an External module. This means that you can

- Take the source tree of this article
- Copy it into the ITK/Modules/External directory
- Reconfigure ITK with CMake
- Compile
- Start using the new STLMeshIO class for reading and writing

The External modules in ITK facilitate the process of adding customized classes to the toolkit in local installations [1].

3 Infrastructure

This STLMeshIO class build upon the infrastructure of the classes

- MeshIOBase
- MeshFileReader
- MeshFileWriter

and uses the IO factory mechanism.

Therefore, we register the STLMeshIOFactory class as an available provider of MeshIO classes, and with this, depending on the filenames (particularly the filename extension) that are requested for reading or writing, this specific MeshIO class may be internally selected for the task by ITK.

For more details on the Mesh IO Framework, please see this article [2]:

- http://www.insight-journal.org/browse/publication/761
- http://hdl.handle.net/10380/3212

4 How to Use

The following code example illustrates how to use the STLMeshIO class to read and write Meshes.

We start with including the headers of fundamental classes:

```
#include "itkQuadEdgeMesh.h"
#include "itkSTLMeshIOFactory.h"
#include "itkSTLMeshIO.h"
#include "itkMeshFileReader.h"
#include "itkMeshFileWriter.h"
```

We declare the type of the new Mesh.

```
  const unsigned int Dimension = 3;
  typedef float PixelType;

  typedef itk::QuadEdgeMesh<PixelType, Dimension>    QEMeshType;
```

The QuadEdgeMeshe is probably the best type of Mesh to use to host the content of STL files, given that it represents a 2D surface embedded into a 3D space.

We register the STLMeshIO factory in the Mesh IO framework with the line:

```
  itk::STLMeshIOFactory::RegisterOneFactory();
```

The STL MeshIO class will respond to requests for reading or writing any mesh whose filename uses the extensions

- .stl
- .STL

The factory framework will take care of selecting the appropriate MeshIO class.

We can now declare and instantiate a MeshFileReader and a MeshFileWriter

```
typedef itk::MeshFileReader< QEMeshType >     ReaderType;
typedef itk::MeshFileWriter< QEMeshType >     WriterType;

ReaderType::Pointer reader = ReaderType::New();
WriterType::Pointer writer = WriterType::New();
```

Set the filenames to be read and to be written

```
reader->SetFileName( argv[1] );
writer->SetFileName( argv[2] );
```

Select whether we are going to write out an STL file in ASCII or BINARY format

```
int fileMode = atoi( argv[3] );

if( fileMode == 0 )
  {
  writer->SetFileTypeAsASCII();
  }
else if( fileMode == 1 )
  {
  writer->SetFileTypeAsBINARY();
  }
```

In this particular example, we connect them together

```
writer->SetInput( reader->GetOutput() );
```

and trigger the reading and writing by calling the Update() method in the writer

```
try
  {
  writer->Update();
  }
catch( itk::ExceptionObject & excp )
  {
  std::cerr << excp << std::endl;
  result = EXIT_FAILURE;
  }
```

5 Examples

STL files can use either an ASCII representation or a BINARY representation.

5.1 ASCII

The ASCII representation follows the structure:

```
solid ascii
 facet normal nx ny nz
  outer loop
   vertex x1 y1 z1
   vertex x2 y2 z2
   vertex x3 y3 z3
  endloop
 endfacet
 ...
 [next facet]
 ...
endsolid
```

http://en.wikipedia.org/wiki/STL_(file_format)#ASCII_STL

5.2 BINARY

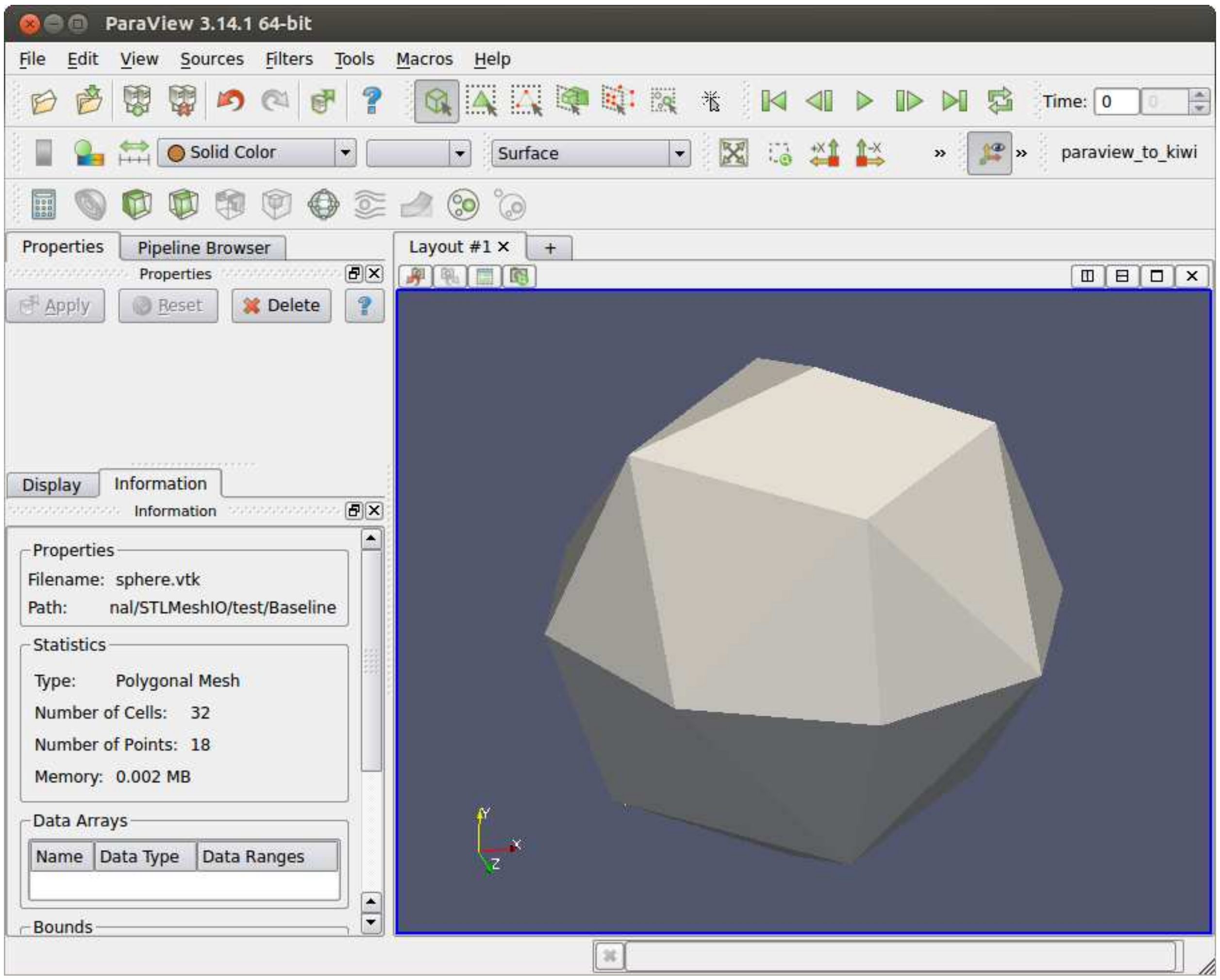
The BINARY representation follows the structure:

```
UINT8[80]     Header
UINT32     Number of triangles

foreach triangle
REAL32[3]     Normal vector
REAL32[3]     Vertex 1
REAL32[3]     Vertex 2
REAL32[3]     Vertex 3
UINT16     Attribute byte count
end
```

http://en.wikipedia.org/wiki/STL_(file_format)#Binary_STL

5.3 Screenshots



Properties
Filename: sphere.vtk
Path: nal/STLMeshIO/test/Baseline

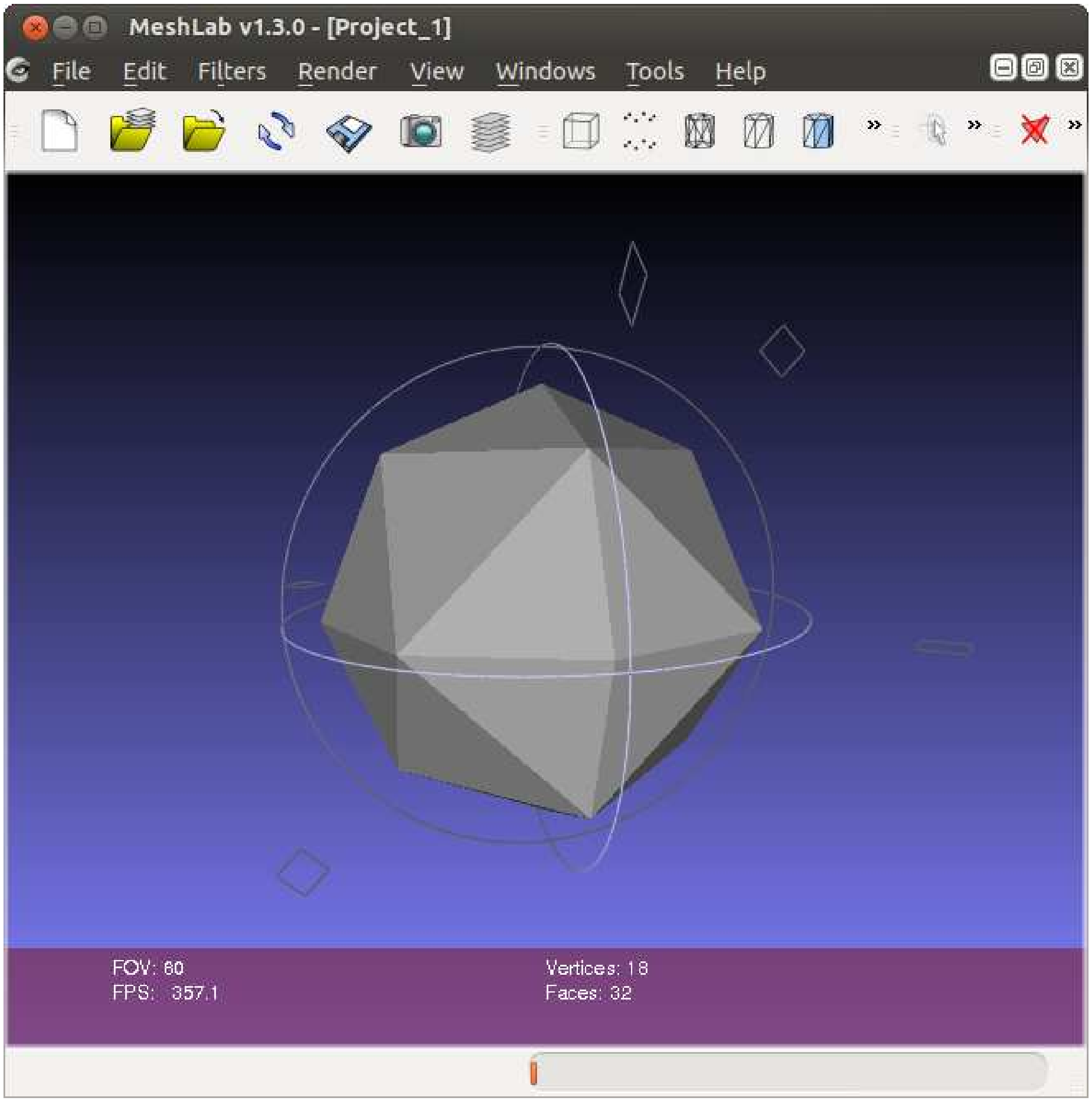
Statistics
Type: Polygonal Mesh
Number of Cells: 32
Number of Points: 18
Memory: 0.002 MB

Data Arrays

Name	Data Type	Data Ranges

Bounds

Figure 1: Input Sphere in VTK format displayed in Paraview



FOV: 60
FPS: 357.1
Vertices: 18
Faces: 32

Figure 2: Output Sphere in STL format displayed in Meshlab

6 Conclusion

It should now be possible to use the STL MeshIO class to write out directly from ITK the segmentations of anatomical structures, and feed them into a 3D printing pipeline.

References

[1] Brian Helba, Luis Ibanez, Brad King, Xiaoxiao Liu, and Matt McCormick. Advance itk with modules. *Kitware Blog*, 2013. 2

[2] W. Zhu. Itk mesh io framework. *Insight Journal*, 2010. 3