



Medical Images

Session 10

Programming Volume Rendering

Dani Tost

Programming with vtk

Visualization Toolkit

The **Visualization Toolkit (VTK)** is an open-source, freely available software system for 3D computer graphics, image processing and visualization. VTK consists of a C++ class library and several interpreted interface layers including Tcl/Tk, Java, and Python. **Kitware**, whose team created and continues to extend the toolkit, offers *professional support and consulting services* for VTK. VTK supports a wide variety of visualization algorithms including: scalar, vector, tensor, texture, and volumetric methods; and advanced modeling techniques such as: implicit modeling, polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangulation. VTK has an extensive information visualization framework, has a suite of 3D interaction widgets, supports parallel processing, and integrates with various databases on GUI toolkits such as Qt and Tk. VTK is cross-platform and runs on Linux, Windows, Mac and Unix platforms.

News [More News >](#)

- 11.27.2013 Kitware Books now Available on Additional Amazon Websites
- 11.14.2013 Kitware to Exhibit Advancements in HPC and Visualization Technolo...
- 10.16.2013 Kitware and Elsevier Collaborate to Bring Interactive, Web-based ...
- 10.01.2013 Mastering CMake, Sixth Edition Now Available!
- 09.24.2013 Kitware and Velodyne announce the release of next generation, ope...

Kitware receives HPCwire's Editors' Choice Award for VTK.

[Learn More >](#)

Logos: NLM, itk, Sandia National Laboratories

(cc) BY-ND This website is licensed under a [Creative Commons Attribution-NoDerivs 3.0 Unported License](#). Additional information on this license can be found [here](#).



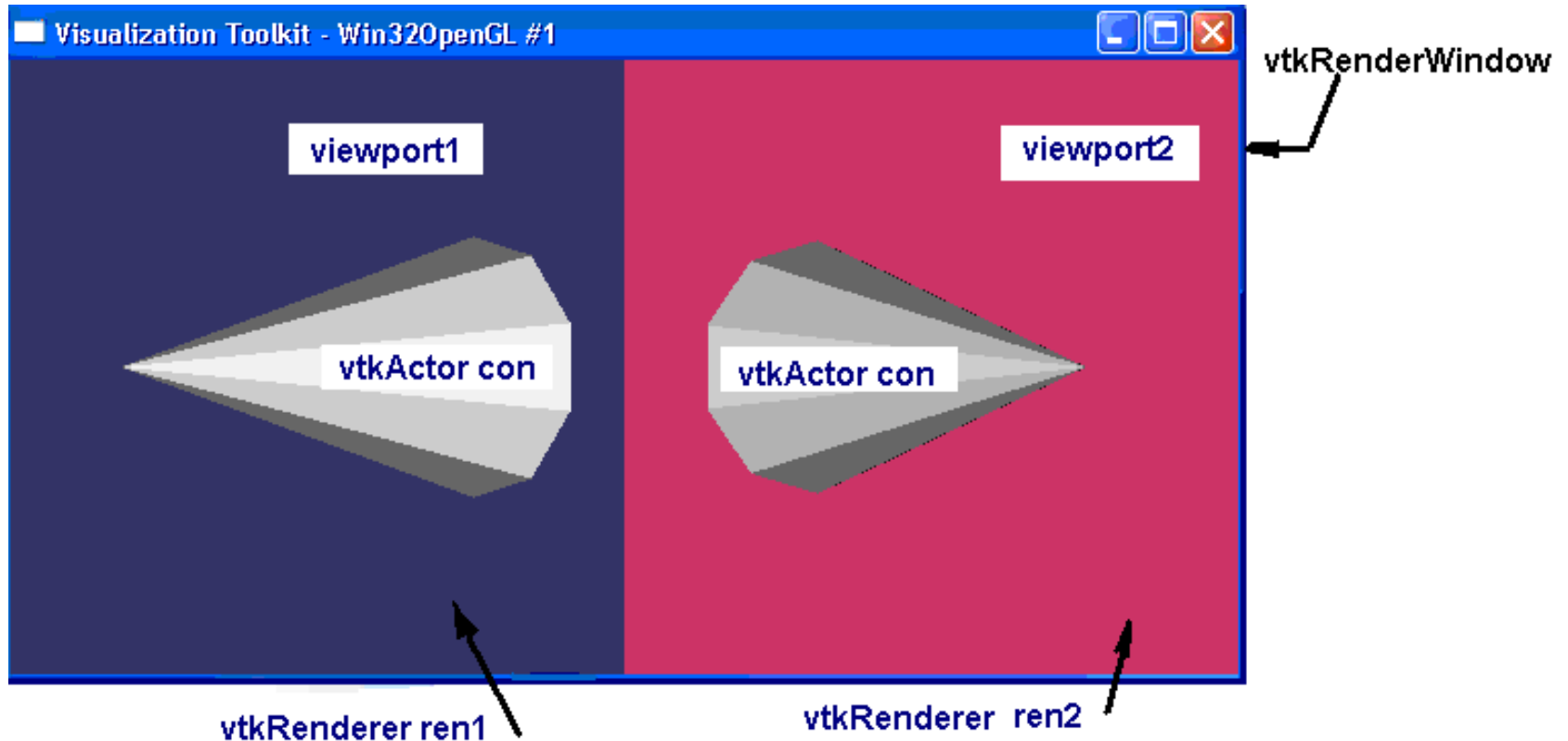
VTK

Interaction layer
Tcl/Qt/ python and java wrappers

VTK
C++

```
$ python  
>>> import vtk  
>>> dir(vtk)  
>>> help(vtk.vtkActor)  
Q
```

Visualization



RenderWindow: graphical window in which render happens

Viewport: viewport in the render window

Renderer: object in charge of rendering

Actors and volumes: objects that are rendered

Interactor: object in charge of user interaction



Class rendering

```
import vtk
```

```
class Rendering :
```

```
    def __init__(self, size, color) :
```

```
        self.renderer = vtk.vtkRenderer()
```

← Renderer

```
        self.renderer.SetBackground(color[0], color[1], color[2])
```

```
        self.render_window = vtk.vtkRenderWindow()
```

← Render window

```
        self.render_window.SetSize(size[0], size[1])
```

```
        self.render_window.AddRenderer(self.renderer)
```

```
        self.interactor = vtk.vtkRenderWindowInteractor()
```

← Interactor

```
        self.interactor.SetRenderWindow(self.render_window)
```

```
        self.interactor.Initialize()
```

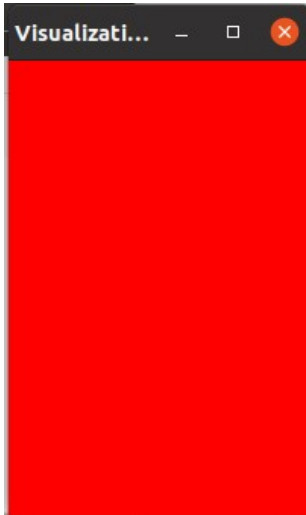
```
if __name__ == '__main__':
```

```
    ren = Rendering((900, 800), (1, 0, 0))
```

```
    ren.interactor.Start()
```



Class Rendering



Run:

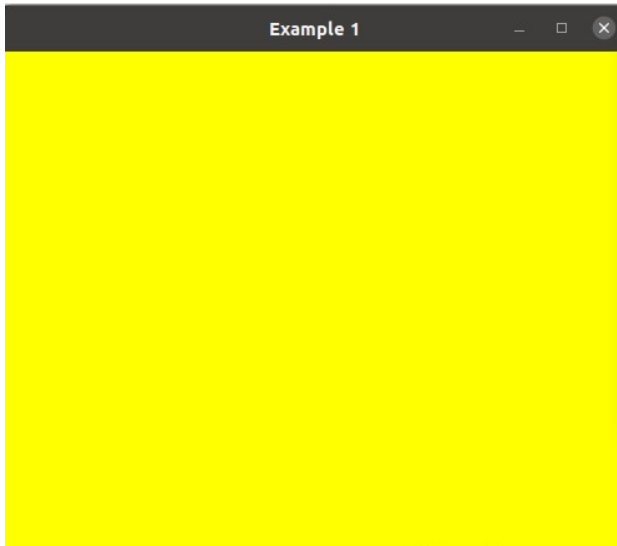
```
python3 rendering.py
```

Try to modify the size of the RenderWindow

Try to change the background color

In which range are color specified?

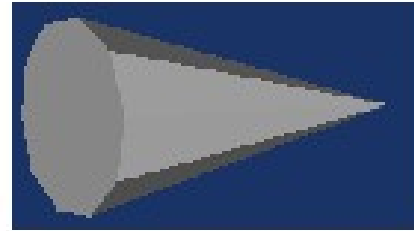
Try to change its name. Look in the documentation of `vtk` classes or do `dir(vtk.vtkRenderWindow)` to look for available methods.



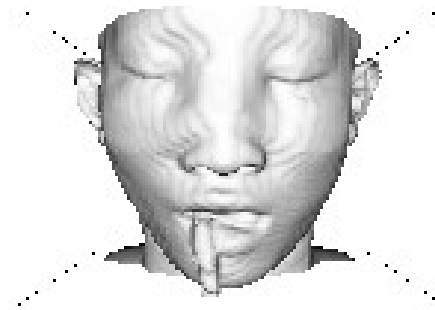
Main objects

vtkProp3D

vtkActor



vtkVolume



Surfaces either parametric or read from a file or created from volumes

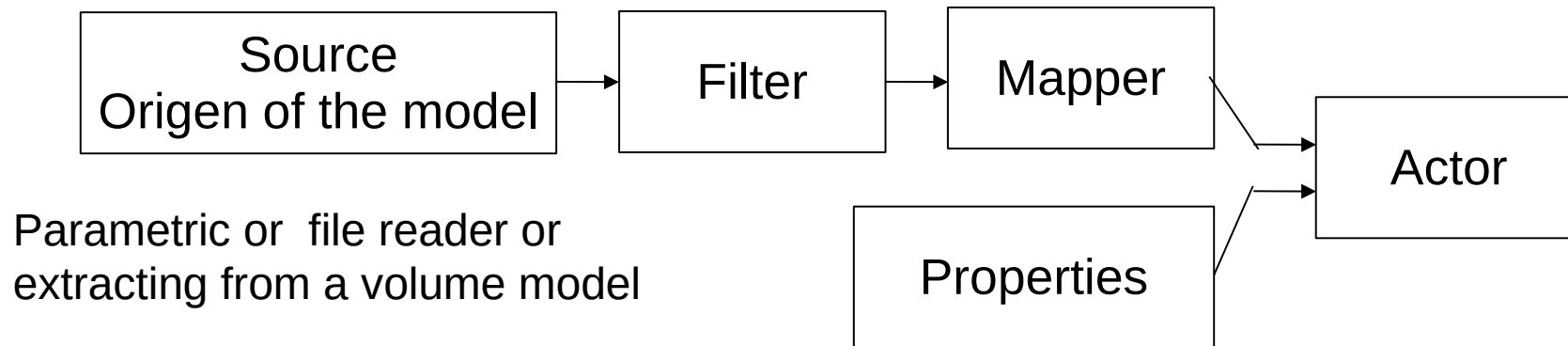
Volumes read from a file or created from other volumes

```
>>> a = vtk.vtkActor()  
>>> a  
(vtkOpenGLActor)0x2d67310  
>>> b = vtk.vtkVolume()  
(vtkVolume)0x2d6675
```



Surface models

Data processing flow



Actors need to be added to the `renderer` to be rendered:

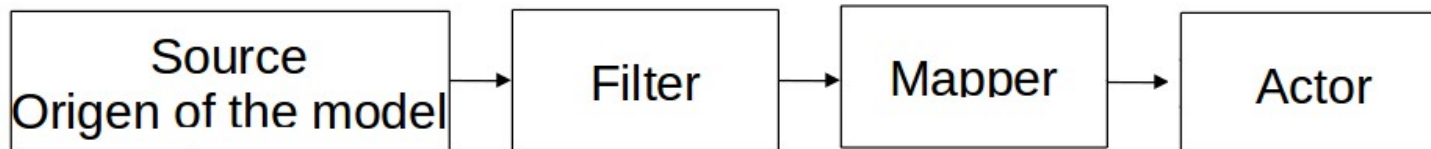
```
renderer.AddActor(actor)
```

Class Surface

Create the module `surface.py` with the class `Surface` that inherits from `vtkActor`

```
import vtk
class Surface(vtk.vtkActor) :
    def __init__(self, name, source, mat= None):
        super().__init__()
        self.name = name
        mapper = vtk.vtkPolyDataMapper()
        mapper.SetInputConnection(source.GetOutputPort())
        mapper.ScalarVisibilityOff()
        self.SetMapper(mapper)
        self.SetProperty(mat)
```

To render the surface with its own material



Here, we don't apply filters.



Class Surface

In the same module, create the class method to create a sphere

```
@classmethod
def from_sphere(cls, name, center, radius, res, mat=None):
    source = vtk.vtkSphereSource()
    source.SetCenter(center)
    source.SetRadius(radius)
    source.SetThetaResolution(res[0])
    source.SetPhiResolution(res[1])
    return cls(name, source, mat)
```





Class Rendering

In the class Rendering, a method to add a surface to the renderer:

```
import vtk
class Rendering :
    def __init__(self, size, color) :
        self.renderer = vtk.vtkRenderer()
        self.renderer.SetBackground(color[0], color[1], color[2])
        self.render_window = vtk.vtkRenderWindow()
        self.render_window.SetWindowName('Example 1')
        self.render_window.SetSize(size[0], size[1])
        self.render_window.AddRenderer(self.renderer)
        self.interactor = vtk.vtkRenderWindowInteractor()
        self.interactor.SetRenderWindow(self.render_window)
        self.interactor.Initialize()
```

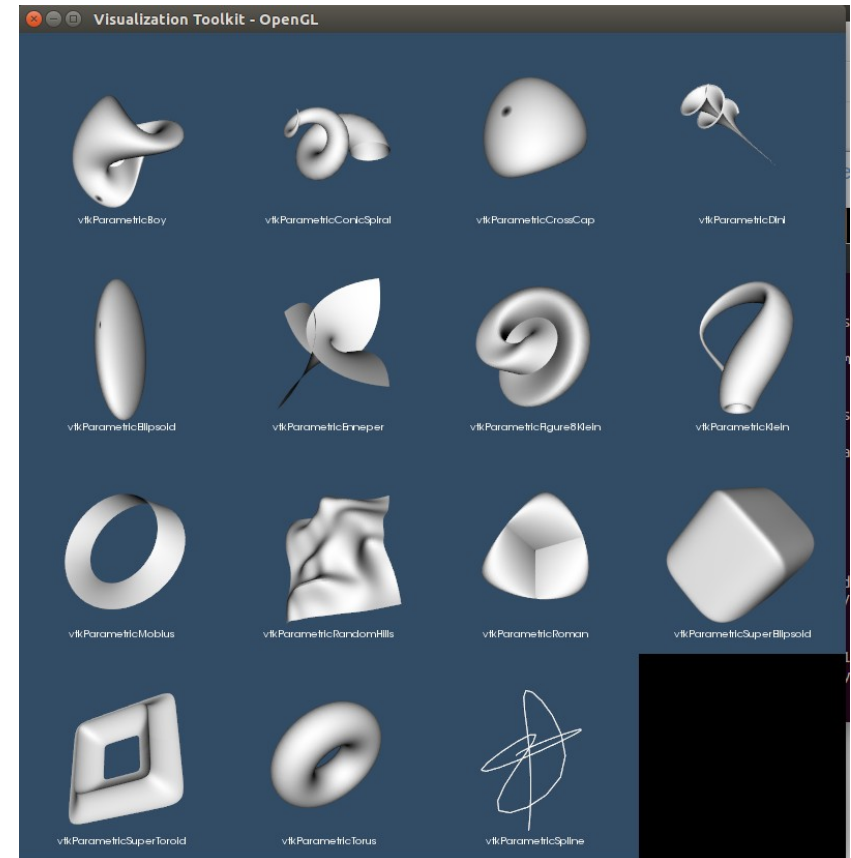
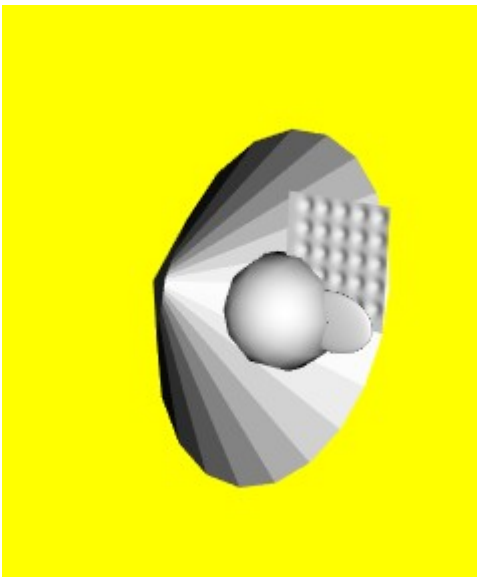
```
def add_actor(self, actor):
    self.renderer.AddActor(actor)
```

Extend

Add other parametric objects

(see <http://www.vtk.org/Wiki/VTK/Examples/Python/GeometricObjects/ParametricObjectsDemo>)

```
@classmethod
def from_random_fields(cls, mat=None):
    random_field = vtk.vtkParametricRandomHills()
    random_field.AllowRandomGenerationOff()
    source = vtk.vtkParametricFunctionSource()
    source.SetParametricFunction(random_field)
    return cls('random_hills', source, mat)
```

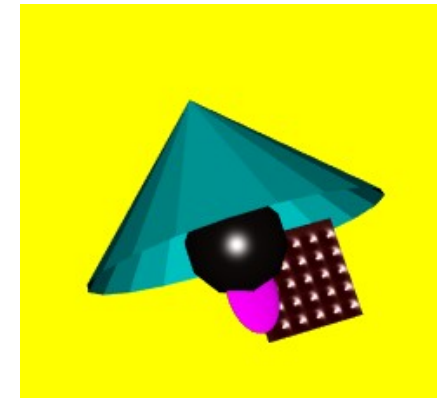




Optical properties

Optical properties for surface shading are part of the class `vtk.vtkProperty`. In the module `material.py` implement the class `SurfaceMaterial`

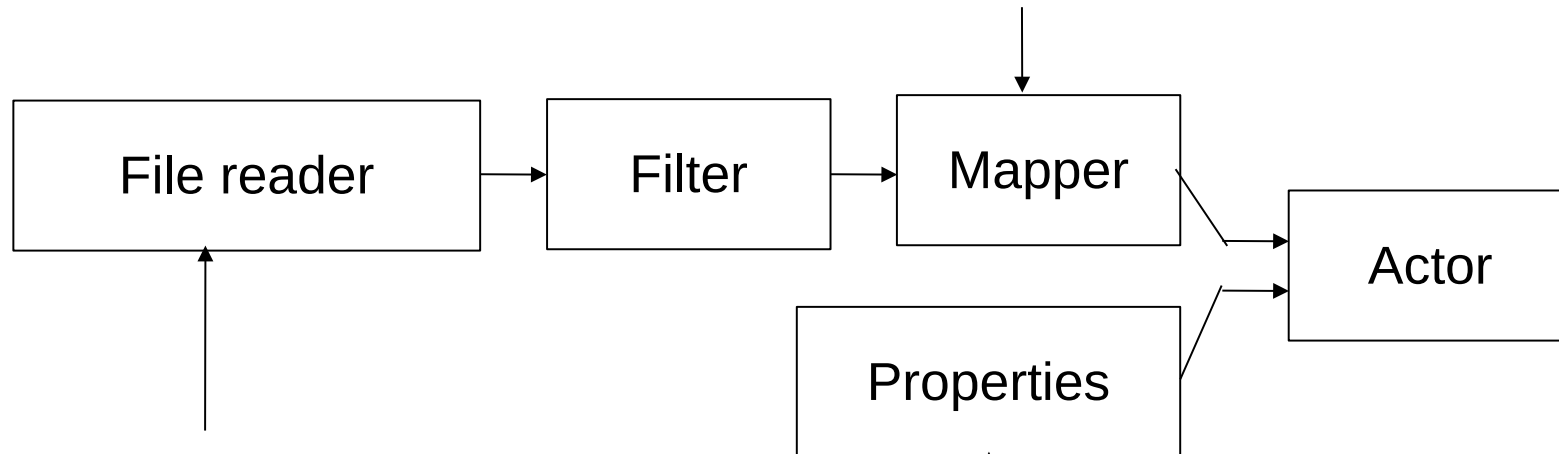
```
import vtk
class SurfaceMaterial(vtk.vtkProperty) :
    def __init__(self, name, Od, kd, Os, ks, n, alpha):
        super().__init__()
        self.SetMaterialName(name)
        self.SetDiffuseColor(Od[0], Od[1], Od[2])
        self.SetDiffuse(kd)
        self.SetSpecularColor(Os[0], Os[1], Os[2])
        self.SetSpecular(ks)
        self.SetSpecularPower(n)
        self.SetOpacity(alpha)
```



Create a material, pass it as a parameter to the sphere instantiation. Play with parameters.

STL files

```
surfMapper = vtk.vtkPolyDataMapper()  
surfMapper.SetInputConnection(rabbit.GetOutputPort() )
```



```
rabbit = vtk.vtkSTLReader()  
rabbit.SetFileName('bunny.stl')
```

```
property = vtk.vtkProperty()  
property.SetColor(1.0, 0.3882, 0.2784)  
property.SetDiffuse(0.7)  
property.SetSpecular(0.4)  
property.SetSpecularPower(20)
```

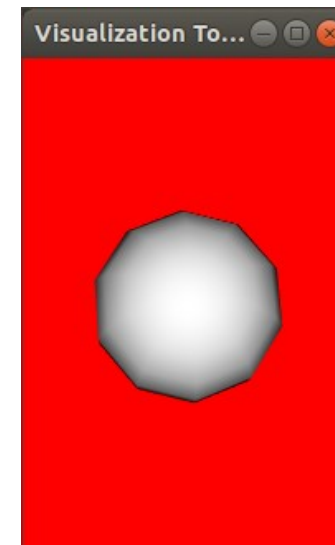
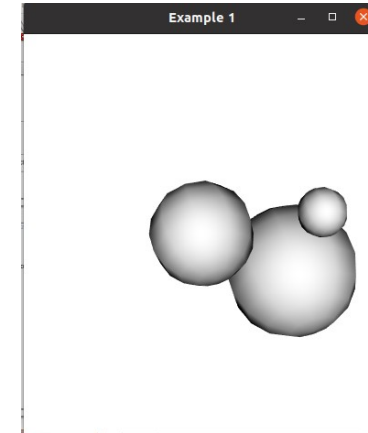


Usage

Modify the main program:

```
if __name__ == '__main__':
    ren = Rendering((900, 800), (1, 0, 0))
    from surface import Surface
    s1 = Surface.from_sphere('sphere_1', (0, 0, 0), 2, (10, 10))
    ren.add_actor(s1)
    ren.interactor.Start()
```

Try then to add new spheres.



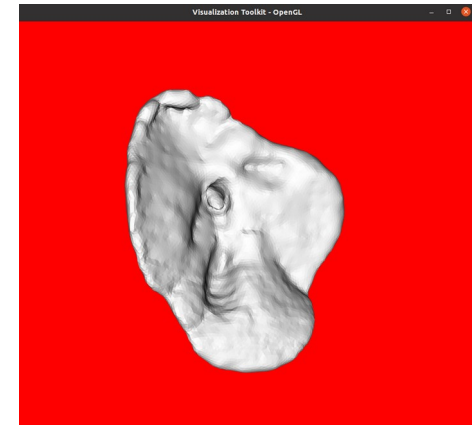


STL Files

In `surface.py` add a class method to read a stl file

In main, read an stl file and add it to the renderer

```
@classmethod
def from_stl_file(cls, filename, name='', mat=None):
    reader = vtk.vtkSTLReader()
    reader.SetFileName(filename)
    if name == '':
        base_name = os.path.basename(filename)
        name, extension = os.path.splitext(base_name)
    return cls(name, reader, mat)
```





STL Files

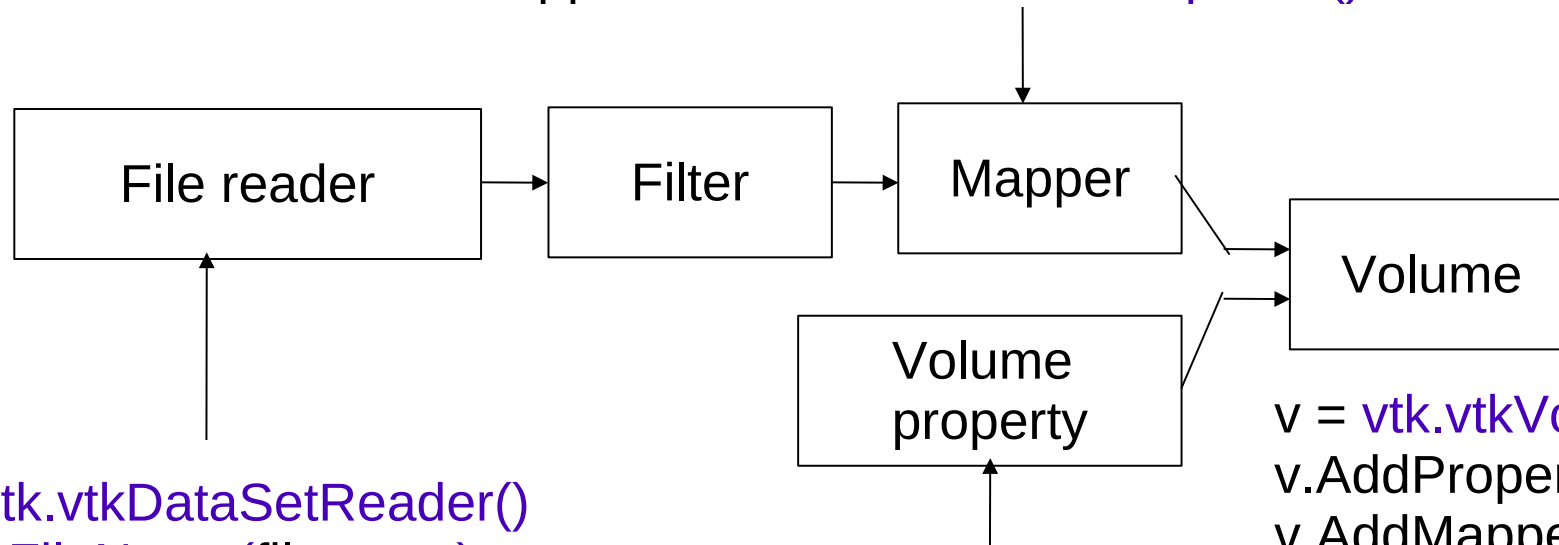
In `surface.py` add a class method to read a stl file

In main, read an stl file and add it to the renderer

```
@classmethod
def from_stl_file(cls, filename, name='', mat=None):
    reader = vtk.vtkSTLReader()
    reader.SetFileName(filename)
    if name == '' :
        base_name = os.path.basename(filename)
        name, extension = os.path.splitext(base_name)
    return cls(name, reader, mat)
```

DVR pipeline

```
mapper.SetVolumeRayCastFunction(raycast)
mapper = vtk.vtkGPUVolumeRayCastMapper()
mapper.SetInputConnection(source.GetOutputPort())
mapper.SetBlendModeToComposite()
```



```
v = vtk.vtkVolume()
v.AddProperty(property)
v.AddMapper(mapper)
```

```
property = vtk.vtkVolumeProperty()
```

```
data = vtk.vtkDataSetReader()
data.SetFileName(filename)
```



DVR

Create the module `volume.py` with a definition of the class `Volume`

```
class Volume(vtk.vtkVolume) :
    def __init__(self, name, reader, mat):
        self.name = name
        mapper = vtk.vtkFixedPointVolumeRayCastMapper()
        mapper.SetInputConnection(reader.GetOutputPort())
        mapper.SetBlendModeToComposite()
        self.reader = reader
        self.SetMapper(mapper)
        self.SetProperty(mat)
```



DVR

Define the transfer functions to create a new class of material:

```
class VolumeMaterial(vtk.vtkVolumeProperty) :
    def __init__(self, name, rgb, opac):
        self.name = name
        colorTF = vtk.vtkColorTransferFunction()
        for point in rgb :
            colorTF.AddRGBPoint(point[0], point[1][0],
                                point[1][1] , point[1][2])
        self.SetColor(colorTF)
        opacTF = vtk.vtkPiecewiseFunction()
        for point in opac :
            opacTF.AddPoint(point[0], point[1])
        self.SetScalarOpacity(opacTF)
```

Suggestion: investigate how to apply surface shading



DVR

Add the class method `read_vtk_file`. Create default transfer functions.

```
@classmethod
def read_vtk_file(cls, filename, mat=None):
    reader = vtk.vtkDataSetReader()
    reader.SetFileName(filename)
    reader.Update()
    if mat == None:
        minv, maxv = reader.GetOutput().GetScalarRange()
        mat = VolumeMaterial.basic(minv, maxv)

    return cls('volume', reader, mat)
```

Suggestion: change the name by the name of the file without path and extension

DVR

Try to render mummy.vtk

```
def main() :
    ren = Rendering((200, 300), (1, 0.5, 1))
    mummy = Volume.read_vtk_file('mummy.vtk')
    ren.renderer.AddVolume(mummy)
    ren.interactor.Initialize()
    ren.interactor.Start()
```

Homework: Create a suitable transfer function as a class method of VolumMaterial

```
mat_mum = VolumeMaterial.mummy()
Mummy = Volume.read_vtk_file('mummy.vtk',
                             mat_mum)
```





Thank you