VTK Designer 2

User Manual, Version 1.0 (Beta)

http://www.vcreatelogic.com/oss/vtkdesigner

VTK Designer 2.0, User Manual Version 1.0

Questions and Comments

If you have questions and comments you can write to info@vcreatelogic.com or support@vcreatelogic.com or drop a snail mail to:

VCreate Logic Private Limited,
#177, 5th Cross, 50 Feet Road,
Opposite BDA Park, Avalahalli,
Banashankari III Stage,
Bengalooru - 560 085
INDIA

Preface

This guide explains the usage aspects of "VTK Designer 2" from VCreate Logic. The document will help you understand the user interface of VTK Designer, create visualization networks, execute them, script them, import and script custom UI forms.

What is VTK Designer?

VTK Designer is a platform for creating, configuring and developing VTK applications to perform scientific/data visualization. VTK or Visualization Toolkit is an open source 3D scientific C++ visualization framework founded and managed by Kitware, and developed by a world wide team of visualization experts. While VTK requires you to program visualization networks in C++, VTK Designer helps you to graphically assemble VTK components.

History

VTK Designer started off as an open source project in the year 2004. The first release of VTK Designer was made in October of 2004. Since then five new revisions of VTK Designer were made, each revision only added a few bug fixes and features.

From January 2007, the core team of VTK Designer began working on a new version that would enhance the visualization experience for its users. As a result of all these efforts, today we have VTK Designer 2.

Purpose

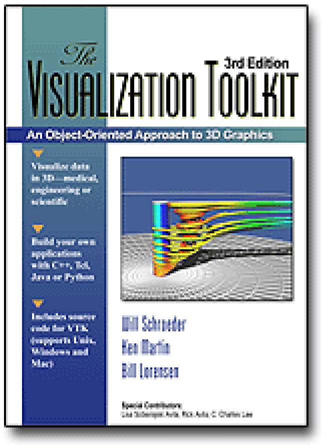
VTK is a fantastic C++ framework for performing 3D visualization. Several software systems have been written to visualize data for different domains. Kitware has Paraview which is a powerful, general purpose data visualization system. While these systems make it easy to visualize data, it can be tricky to assemble a custom visualization scenario.

The founders of the "VTK Designer" project wanted to develop a system that would help its users to custom assemble a visualization scenario and program it to achieve their goals. In many ways "VTK Designer" is like Visual Basic, but for the 3D Visualization world. It provides simple drag and drop mechanisms to create visualization networks and execute them. It also provides an inbuilt script editor and script evaluator to program the network for achieving custom results.

Audience

This document is targeted at people who have had some experience in 3D programming. While knowledge of VTK is surely a plus point, it is not however absolutely necessary to understand the content of this manual. However this manual does not dwell into the science of visualization or VTK itself. Related concepts are introduced as and when required, but an in depth description is beyond the scope of this manual.

If you have any questions on VTK Designer, you can always write an email to us at info@vcreatelogic.com or support@vcreatelogic.com



We encourage to learn VTK to truly capture the power of VTK Designer. We encourage you to read "The *Visualization Toolkit An Object-Oriented Approach To 3D Graphics, 4th Edition"* to learn about VTK.

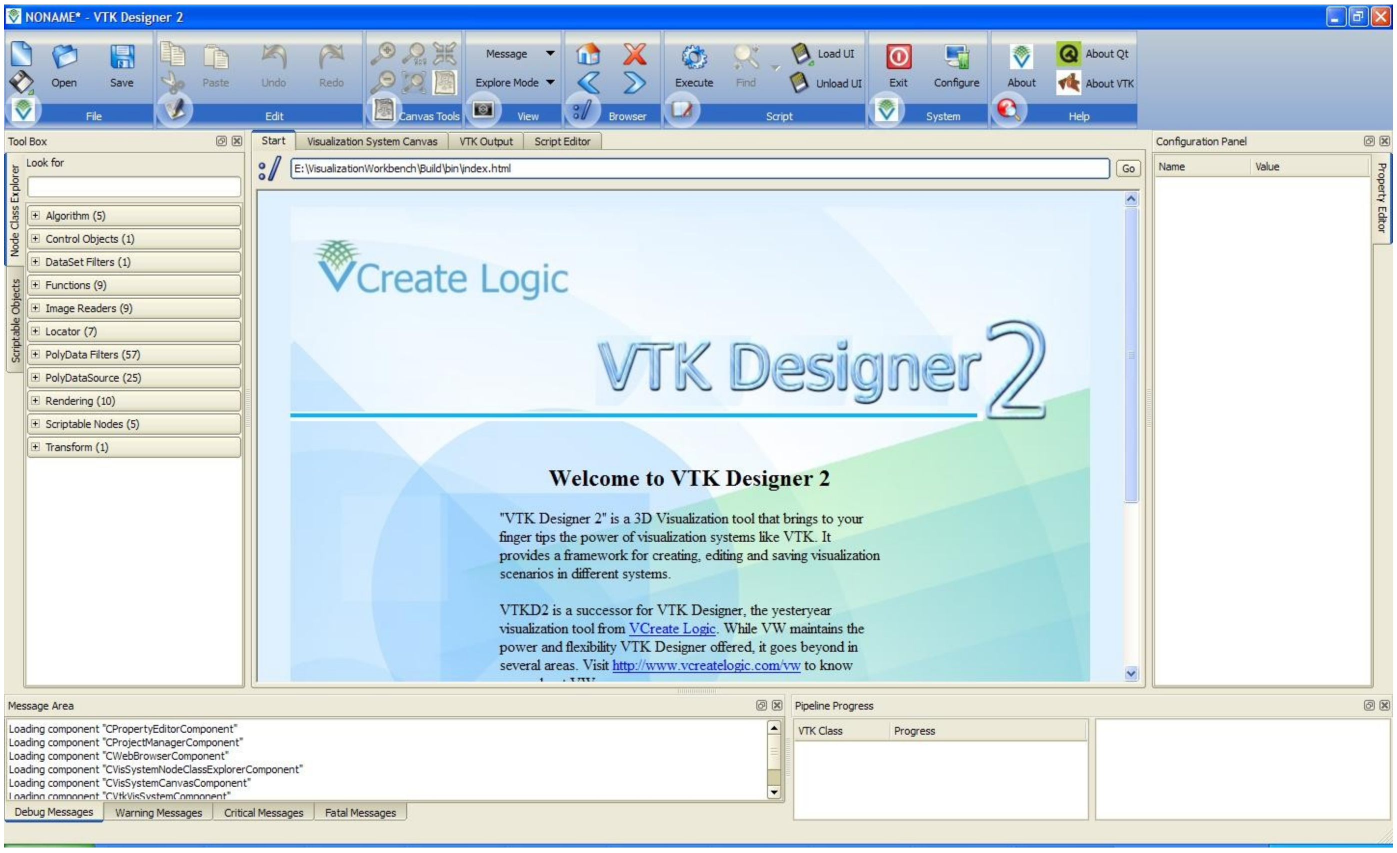
This book is written by Will Schroeder, Ken Martin and Bill Lorensen and is available from Kitware itself. The book comes with a CDROM media that has the complete source code and API documentation for VTK 5.0.2.

TABLE OF CONTENTS

Introducing the VTK Designer User Interface

The user interface for VTK Designer is designed to help its users develop complex visualization scenarios and to allow deployment of custom visualization applications on it. Towards this the UI looks both like an integrated development environment and like a application shell. In this chapter we introduce the user interface of VTK Designer.

When VTK Designer is started for the first time, it would look like the window shown below.



Welcome to VTK Designer 2

"VTK Designer 2" is a 3D Visualization tool that brings to your finger tips the power of visualization systems like VTK. It provides a framework for creating, editing and saving visualization scenarios in different systems.

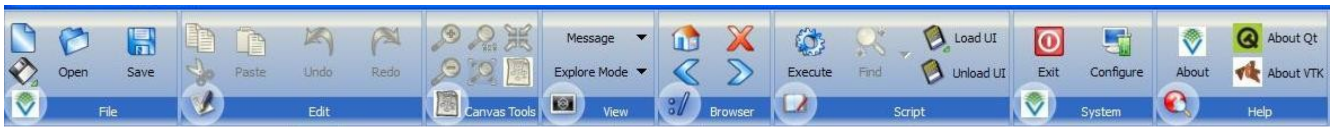
VTKD2 is a successor for VTK Designer, the yesteryear visualization tool from VCreate Logic. While VW maintains the power and flexibility VTK Designer offered, it goes beyond in several areas. Visit http://www.vcreatelogic.com/vw to know

```
Loading component "CPropertyEditorComponent"
Loading component "CProjectManagerComponent"
Loading component "CWebBrowserComponent"
Loading component "CVisSystemNodeClassExplorerComponent"
Loading component "CVisSystemCanvasComponent"
Loading component "CVtkVisSystemComponent"
```

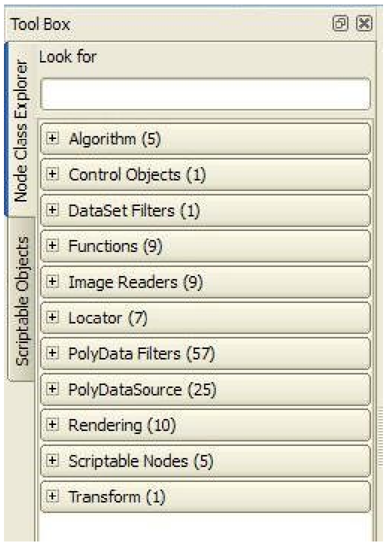
Elements of the user interface.

The User interface consists of

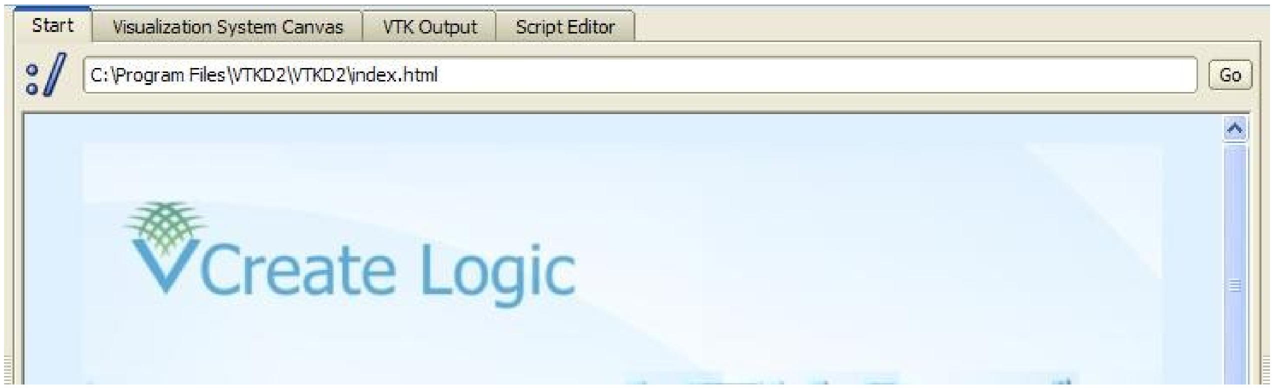
1. A Menu Strip on which all UI actions are attractively placed.



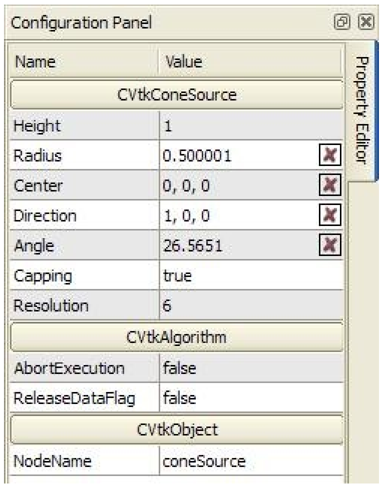
2. A Tool Box on which you can find tools and options exposed by different components of VTK Designer.



3. A Workspace on which you can find all sorts of editors and output viewers.

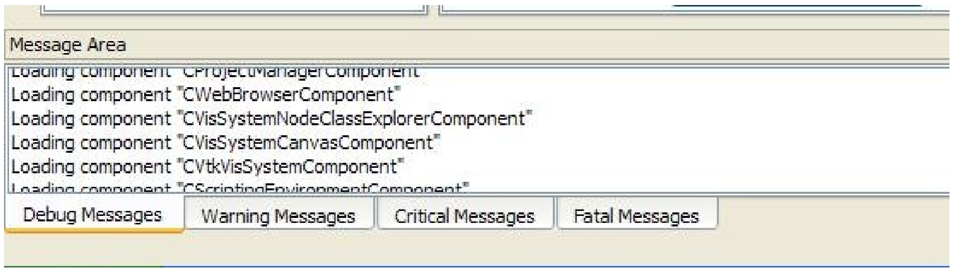

Start
Visualization System Canvas
VTK Output
Script Editor
C:\Program Files\VTKD2\VTKD2\index.html
Go
VCreate Logic

4. A Configuration Panel where you can configure objects chosen in the "Visualization System Canvas"



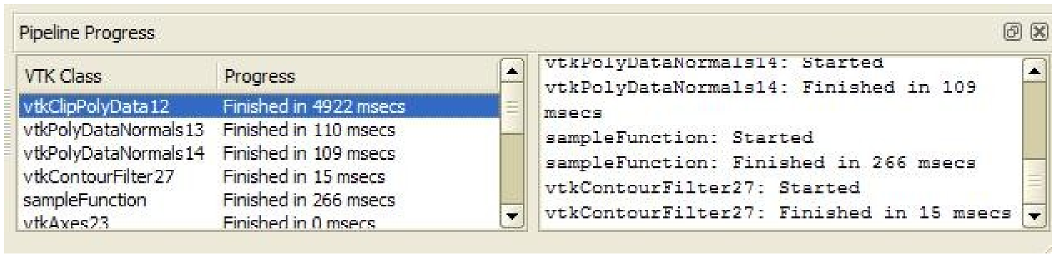
Name	Value
CVtkConeSource	
Height	1
Radius	0.500001
Center	0, 0, 0
Direction	1, 0, 0
Angle	26.5651
Capping	true
Resolution	6
CVtkAlgorithm	
AbortExecution	false
ReleaseDataFlag	false
CVtkObject	
NodeName	coneSource

5. A Message Area where debug, warning and critical messages from all components in VTK Designer are shown.



```
Loading component "CProjectManagerComponent"
Loading component "CWebBrowserComponent"
Loading component "CVisSystemNodeClassExplorerComponent"
Loading component "CVisSystemCanvasComponent"
Loading component "CVtkVisSystemComponent"
Loading component "CScriptingEnvironmentComponent"
```

6. A pipeline progress monitor panel that shows the processing time consumed by different algorithms in the visualization network.



VTK Class	Progress
vtkClipPolyData12	Finished in 4922 msecs
vtkPolyDataNormals13	Finished in 110 msecs
vtkPolyDataNormals14	Finished in 109 msecs
vtkContourFilter27	Finished in 15 msecs
sampleFunction	Finished in 266 msecs
vtkAxes23	Finished in 0 msecs

```
vtkPolyDataNormals14: Started
vtkPolyDataNormals14: Finished in 109
msecs
sampleFunction: Started
sampleFunction: Finished in 266 msecs
vtkContourFilter27: Started
vtkContourFilter27: Finished in 15 msecs
```

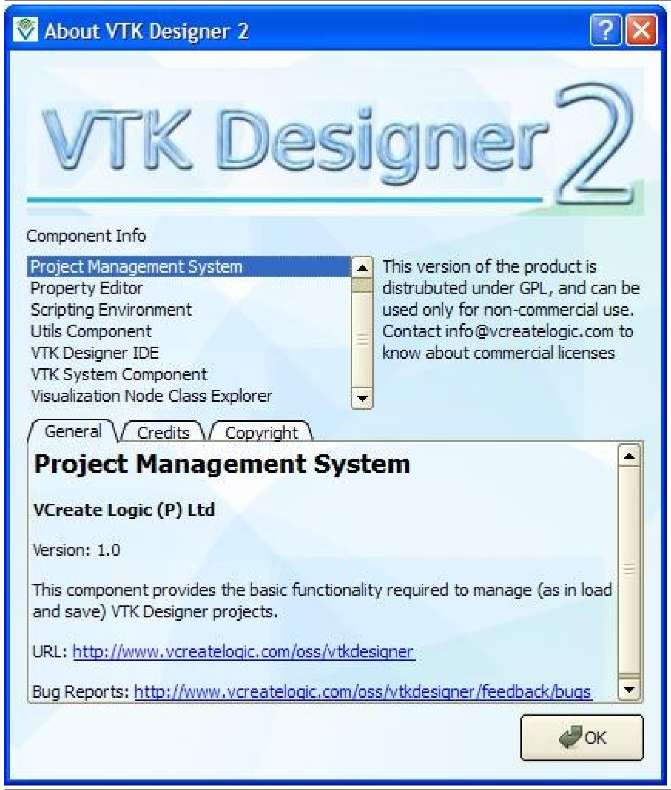
VTK Designer Components

VTK Designer is composed of several components. The Generic Component Framework (or GCF)[1] provides the necessary software infrastructure to assemble VTK Designer from its components. The basic VTK Designer comes with the following components

1. VTK Designer IDE
2. Class Explorer
3. Visualization System Canvas
4. VTK Class Factory
5. Property Editor
6. Message Log Window
7. Configuration Dialog Box.
8. Project Manager.
9. Scripting Environment

VTK Designer's About Box shows you the list of available components

1 GCF is an award winning Qt based UI component framework from VCreate Logic. GCF is available under the GNU/GPL v2 License. You can download it from http://www.vcreatelogic.com/oss/gcf.



Project Management System
Property Editor
Scripting Environment
Utils Component
VTK Designer IDE
VTK System Component
Visualization Node Class Explorer

This version of the product is distrubuted under GPL, and can be used only for non-commercial use. Contact info@vcreatelogic.com to know about commercial licenses

Project Management System

VCreate Logic (P) Ltd

Version: 1.0

This component provides the basic functionality required to manage (as in load and save) VTK Designer projects.

URL: http://www.vcreatelogic.com/oss/vtkdesigner

Bug Reports: http://www.vcreatelogic.com/oss/vtkdesigner/feedback/bugs

We also have custom components for that will help enhance your experience with VTK Designer and also allows you to do more than just assembling visualization networks.

1. Reporting Framework (not for release yet)

   Using the reporting framework, you can create PDF reports consisting of detailed technical information from your visualization experiment. *More about this in a later chapter.*

2. OpenHaptics Integration (not for release yet)

   Using the OpenHaptics Integration plugin, you will be able to use SensAble Technology's Haptic Devices with their OpenHaptics toolkit to haptically render your visualization on to the device. This means that you will be able to touch and feel your visualization and also perform meaningful haptics visualization. *More about this in a later chapter.*

3. Distributed/Parallel Visualization (not for release yet)

   Using the Distributed/Parallel Visualization plugin, you will be able to split your visualization network into several nodes in your cluster or into several processes on your local system to speed your visualization processes.

4. Volumetric Visualization Support (not for release yet)

   Using the Volumetric Visualization plugin, you will be able to perform volumetric visualization using specialized ray tracing algorithms from VTK.

Cone Source – Your First Pipeline in VTK Designer

The primary purpose of VTK Designer is to help construct complex visualization networks or pipelines consisting of VTK algorithms to perform a visualization task. A pipeline, in VTK, consists of the following elements

1. **Data Source:** The source forms the starting point of visualization. As the name suggests the source provides data that needs to be visualized. The data source can provide this data by reading a file, socket or any input device; or by mathematically evaluating an equation. The function of the source it to simply provide raw data.
2. **Data Filter:** Data sources may provide unwanted data at times. A data filter takes the data provided by the source, and filters out the unwanted data in it. The filtered data can then be passed on to the next stages in the pipeline.
3. **Mapper:** A mapper is the geometric representation of an actor.
4. **Actor:** An actor represents an object rendered in the scene, along with its properties and position. It can be treated as logical entity in the scene.
5. **Renderer:** A renderer coordinates the rendering process involving lights, cameras and actors.
6. **Render Window:** Manages a window on the display device, where the rendered graphics will be draw. The above elements when connected in sequence forms a pipeline.

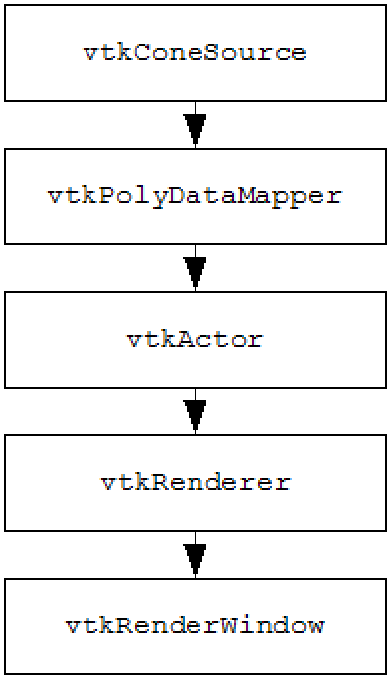
When elements of the type described above are brought together, connected appropriately and executed, we can perform some meaningful visualization.

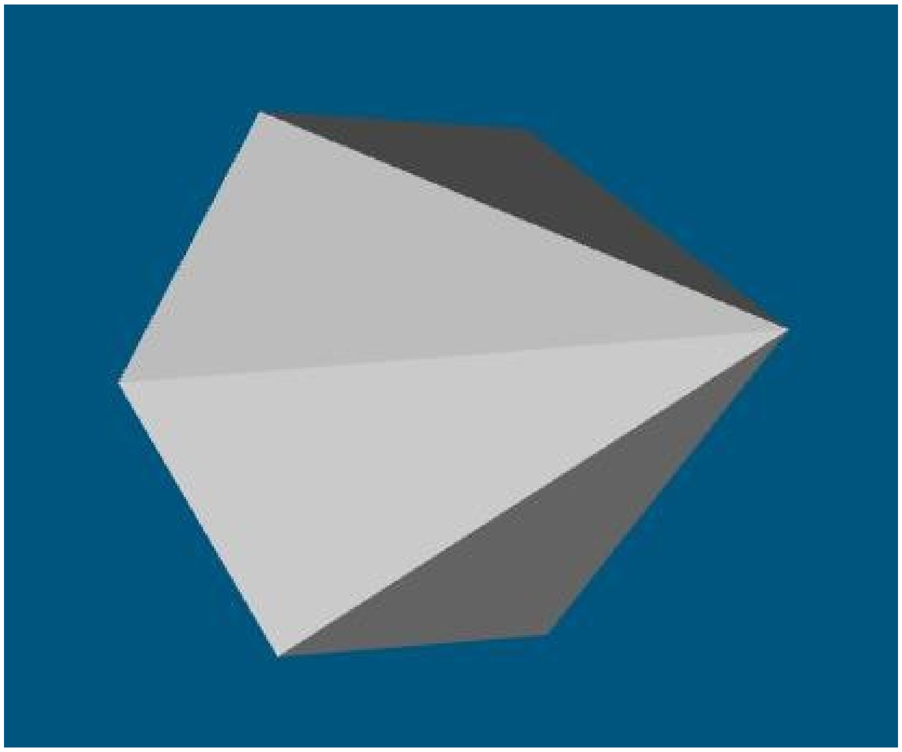
Visualizing a Cone in 3D

Using the concepts learned so far lets visualize a 3D cone. To do this we need the following algorithms.

1. `vtkConeSource` : This algorithm in VTK generates the data required to visualize a cone.
2. `vtkPolyDataMapper` : This algorithm accepts the data generated by `vtkConeSource` and renders it into OpenGL.
3. `vtkActor` : This algorithm sets the material properties for the cone object.
4. `vtkRenderer` : This algorithm prepares the OpenGL viewport into which the actor would be rendered. Essentially it sets up the lights and camera.
5. `vtkRenderWindow` : This algorithm provides a rectangular space on to which all the output is drawn.

The above algorithms must be connected in the following way to make a visualization pipeline.

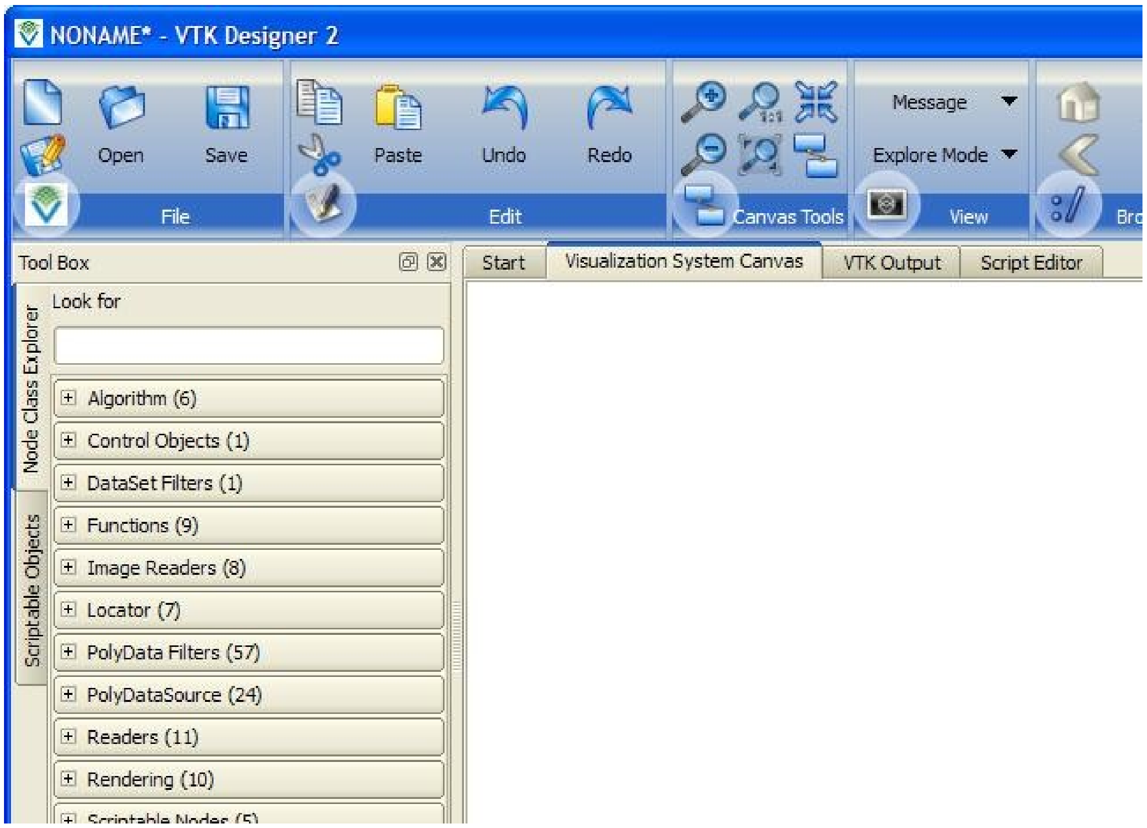




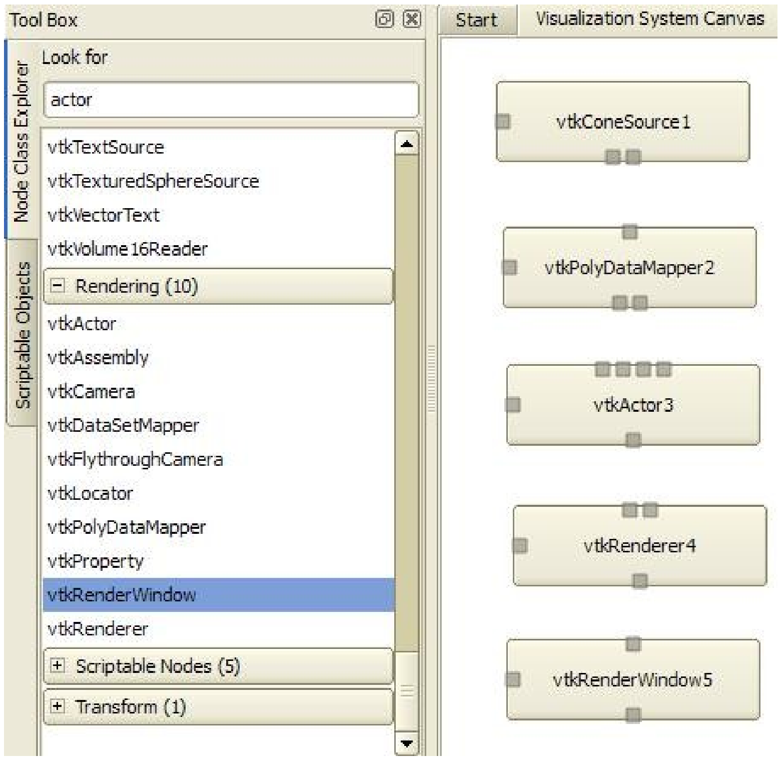
Step by step walkthrough

Lets now walk through the process of creating the above pipeline in VTK Designer and executing it.

1. Start VTK Designer and select the "Visualization System Canvas" tab in the main window.

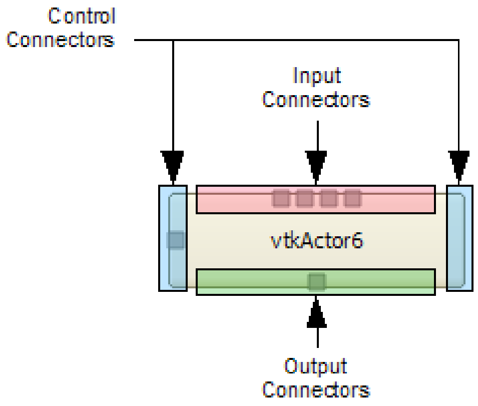


2. Drag `vtkConeSource`, `vtkPolyDataMapper`, `vtkActor`, `vtkRenderer` and `vtkRenderWindow` classes from the node class explorer on the canvas as shown below.

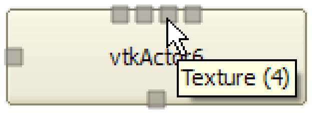


Tip: You can type a few letters from the class name into the “Look For” text box of the “Node Class Explorer” panel in the left dock area to quickly locate the class that you are looking for.

3. You can notice that each node has certain connector paths shown on it. Usually input connectors are shown on the top of the node, output connectors are shown at the bottom and control connectors are shown either on the left or right of the node.

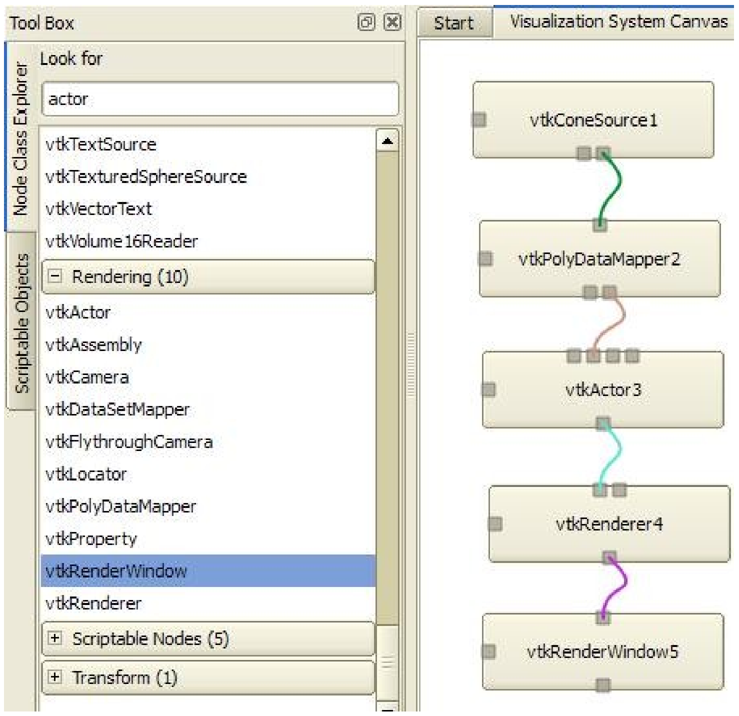


Inputs to node are given via one or more input connectors. Output provided by the node is available at output connectors. Control signals between nodes can be exchanged via control connectors. You can move the mouse over any of the connectors to view a brief description of the connector type.

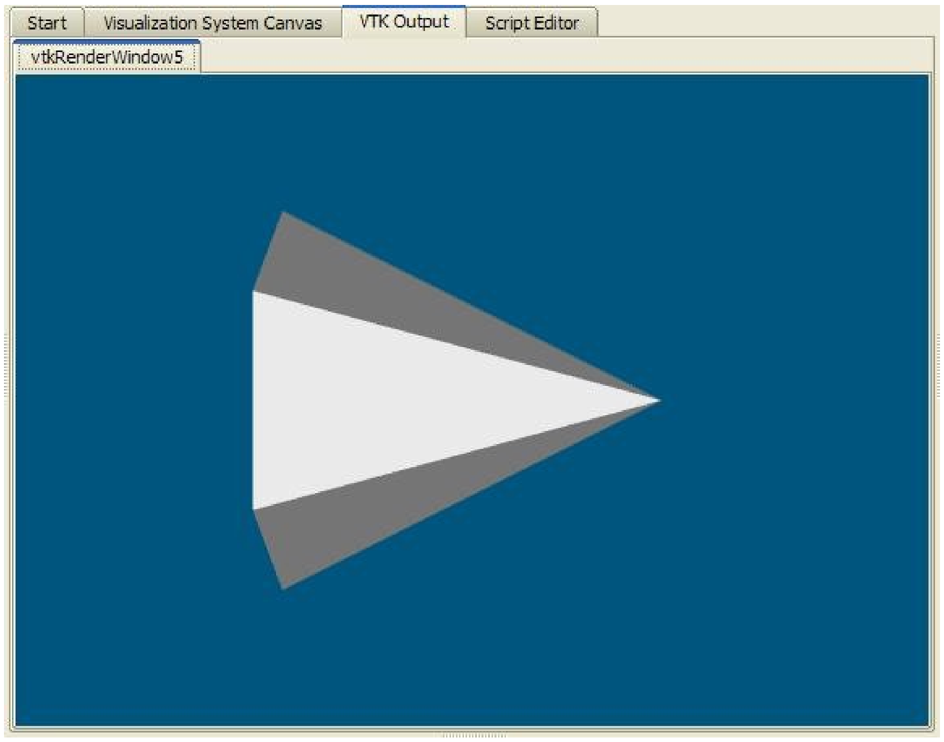


To connect two nodes you can simply drag a connection line from the output connector of one node to the input connector of another node. Of course VTK Designer 2 will validate the connection before actually making it. By validation we mean, checking whether the input/output datatypes are compatible. Sometimes it may be difficult to figure out the right kind of input and output connector combinations. In such cases you can make use of the “Auto Connect” feature in VTK Designer 2. This feature is described in the next step.

4. Hold the Ctrl key on the keyboard and use the mouse to drag connectors between `vtkConeSource1` and `vtkPolyDataMapper2` and so on until you connect all the nodes into a pipeline as shown below.



5. Now click on the “VTK Output” tab in the workspace to view the output. You can even interact with the output.



By now you would have realized that constructing and executing visualization pipelines in VTK Designer is very simple. Lets now look at what happened behind the scenes in VTK Designer while you were constructing and executing the pipeline.

Wrapper Classes and Class Factories

The concept of wrapper classes has been around in VTK Designer since its first version. Although the basic design patterns is still the same, the actual wrapper class architecture has changed significantly between VTK Designer 1.x to 2.0.

Wrapper Classes

VTK Designer accesses the functionality offered by VTK classes via a wrapper class layer, which is essentially a collection of wrapper classes that adhere to a common framework. A Wrapper class essentially manages a VTK class

1. It provides means for accessing the methods on that class, and also ensures the "health" of the class at all times.
2. It manages the life time of instances of the backend VTK class.
3. It helps interface with the front end to offer visual programming methods like performing connections between nodes, arranging nodes on the canvas etc.

Each time you drag a class from the “Node Class Explorer” on to the visualization system canvas, the front end essentially creates a wrapper class which internally creates an instance of the corresponding VTK class. The wrapper class framework is designed to allow

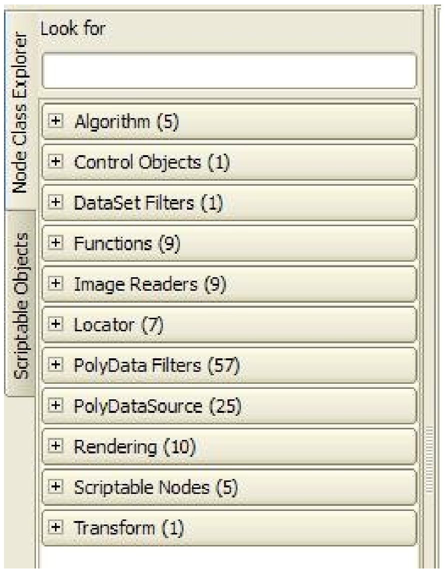
1. Dynamic querying of properties. This helps in the display and editing of properties in the “Property Editor” panel.
2. Connections between compatible input and output paths of wrappers.
3. Scripting of the node functionality.

Wrapper Class Factory

VTK Designer is designed for modularity. This means that while the basic “VTK Designer” software comes bundled with a set of wrapper classes, it is possible to deploy mode bundles with time. The technical term we use for each wrapper class bundles is: “Wrapper Class Factory”.

VTK Designer can potentially have several Wrapper Class factories. Each factory offering one or more classes[2] that can used to make a pipeline. The “Node Class Explorer” component in VTK Designer lists the names of all classes in the system grouped appropriately.

2 Classes offered by wrapper class factories need not be VTK classes. Although most classes in “VTK Designer” are VTK classes, you can have non-VTK Classes also.



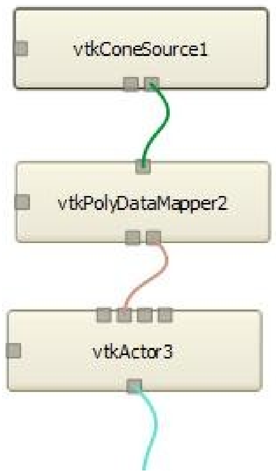
Algorithm (5)
Control Objects (1)
DataSet Filters (1)
Functions (9)
Image Readers (9)
Locator (7)
PolyData Filters (57)
PolyDataSource (25)
Rendering (10)
Scriptable Nodes (5)
Transform (1)

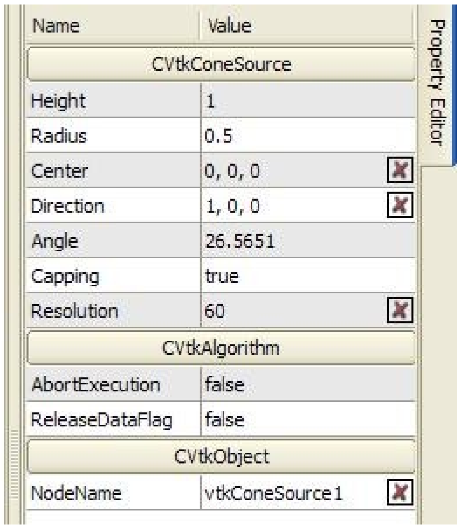
Illustration 1: Node Class Explorer showing classes organized into groups.

Editing the "Cone Source" pipeline

Lets make use of VTK Designer's editing capabilities to change the number of sides or resolution of the vtkConeSource.

1. Click on the "vtkConeSource1" node in the pipeline to view its properties in the "Property Editor" panel. Now click on the "Resolution" property in the property editor to edit the value of Resolution. Lets change the resolution to 60.



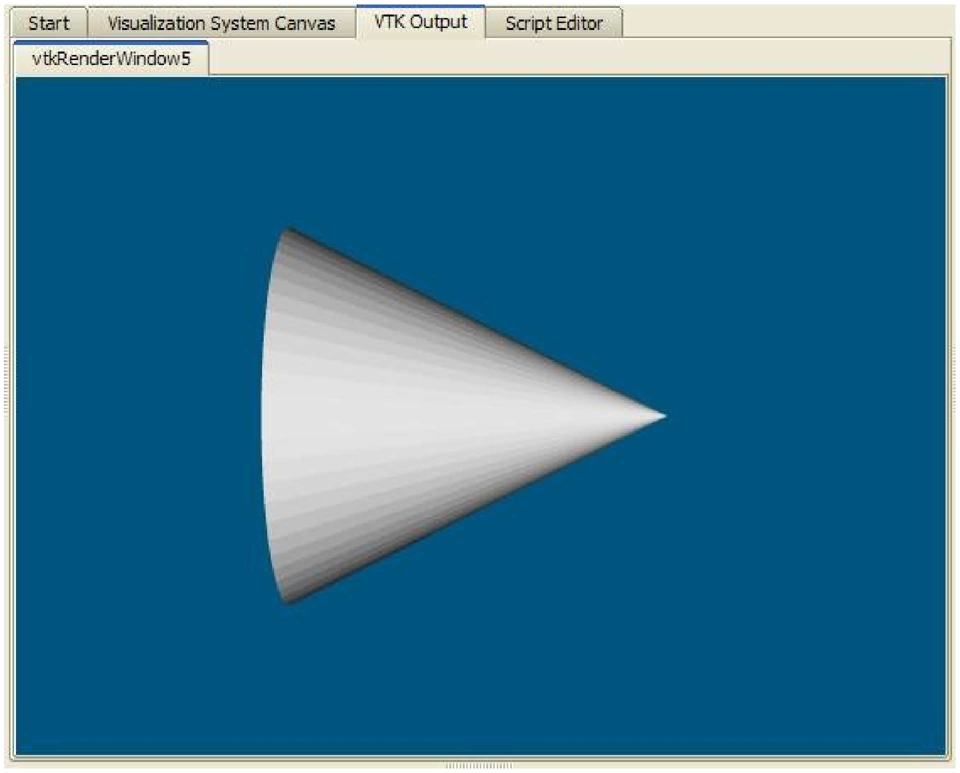


Name	Value
CVtkConeSource	
Height	1
Radius	0.5
Center	0, 0, 0
Direction	1, 0, 0
Angle	26.5651
Capping	true
Resolution	60
CVtkAlgorithm	
AbortExecution	false
ReleaseDataFlag	false
CVtkObject	
NodeName	vtkConeSource1

You will notice that a icon is shown next to the changed properties. Clicking on that icon will revert the property's to its default value.

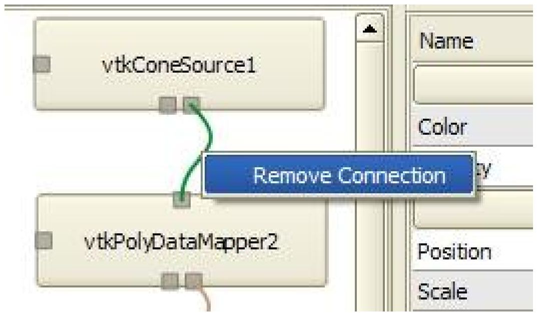
2. Now that the resolution property is changed, you can click on the "VTK Output" tab to view the modified cone source. ***Sometimes you might have to click on the render window area to view an updated***

image. This is because the render window is not updated each time a property is changed.

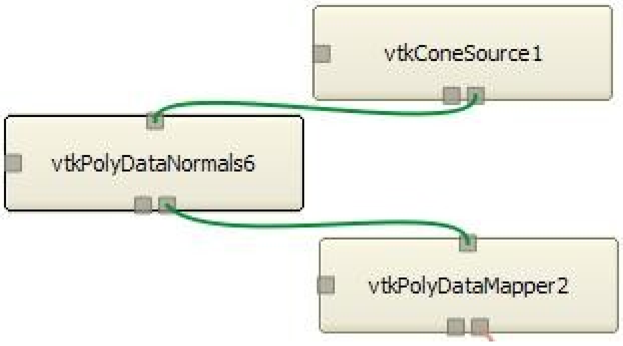


3. You will notice that the surface of the cone is still not smooth. To fix this we will have to introduce a "vtkPolyDataNormals" object between "vtkConeSource1" and "vtkPolyDataMapper2".

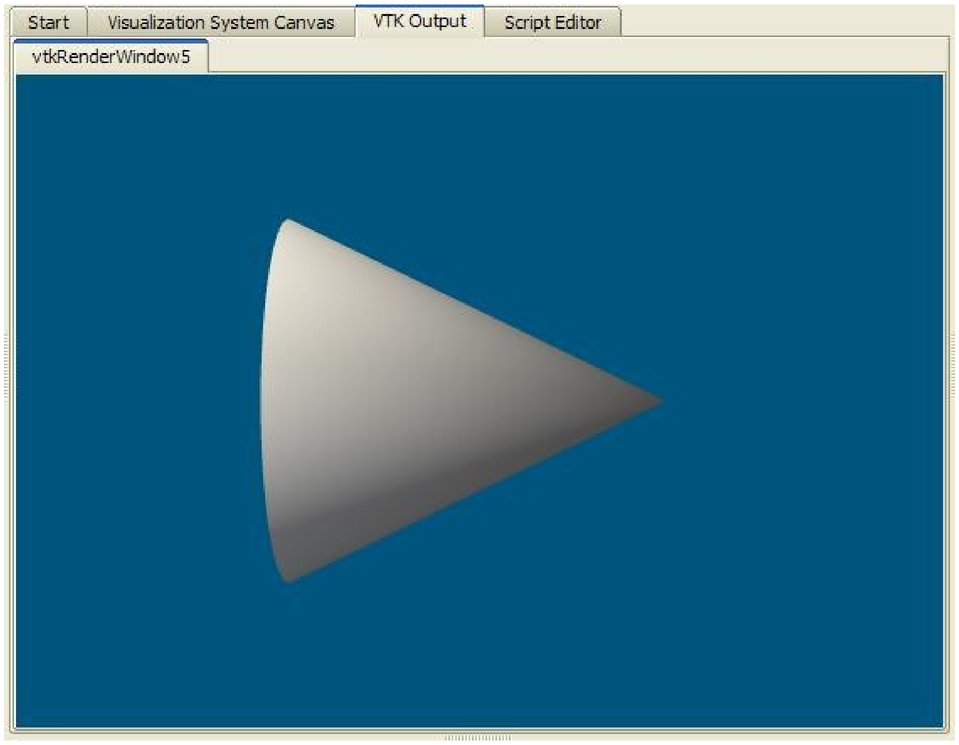
   1. First disconnect the link between "vtkConeSource1" and "vtkPolyDataMapper2". Right click on the connector and select "Remove Connection".



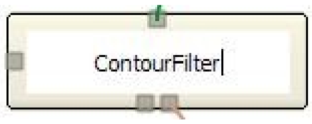
   2. Drag and drop a "vtkPolyDataNormals" object and link it up as shown in the image below.



4. Now you can notice that the surface of the cone is much more smooth.



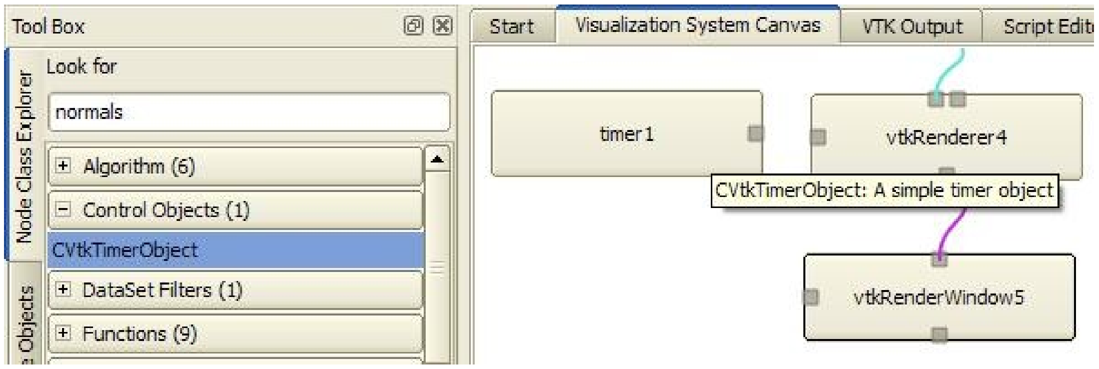
If you want to change the name of any node in the pipeline, just click on the node and press F2. You will get a input box superimposed on the node into which you can type the new name of the node.



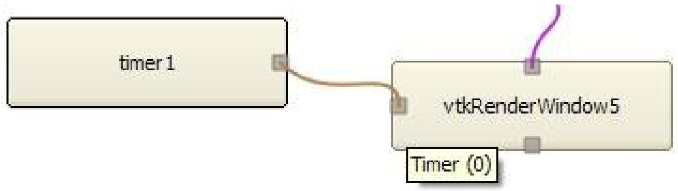
Animating the cone using ECMA Scripts

One of the key strengths of VTK Designer 2 over its predecessors is that it allows you to script your pipelines. Suppose that you wanted to rotate the cone actor by 3 degrees about its Z axis at regular intervals thereby creating a rotation animation. To do this you would need a timer that would generate the timer signals and a function that would specify what to do when the timer times out. The following steps show you how you can achieve this in VTK Designer 2.

1. Drag a timer object on to the canvas

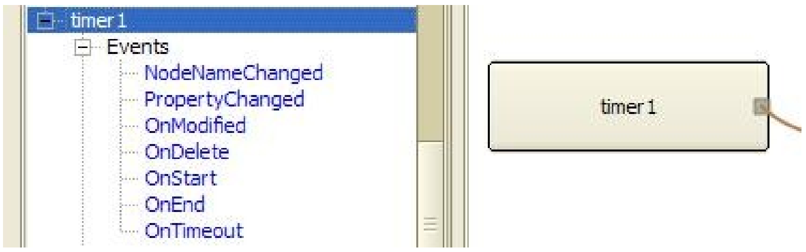


2. You will notice that timer1 has no input and output connectors, but it has one control connector. This is because the sole responsibility of a timer node is to generate timeout signals. Connect timer1 and vtkRenderWindow5 as shown in the figure below.

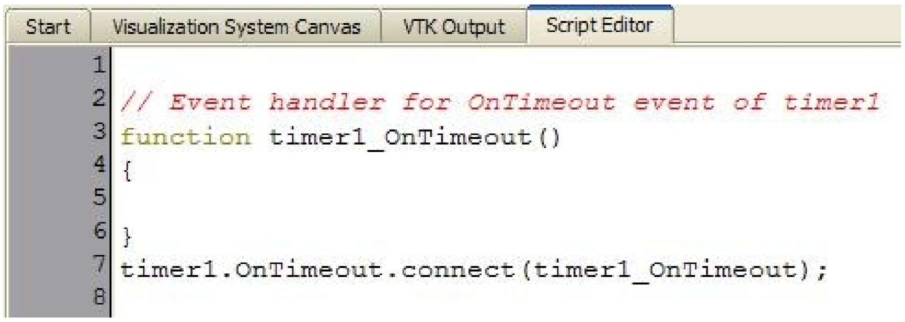


The control connection signifies that when ever the timer1 node signals a timeout, the render window is requested to redraw itself.

3. Next we have to ensure that everytime the timeout signal is raised, the actor is rotated by 3 degrees. To do this, click on the "Scriptable Objects" panel on the left dockarea of the main window. Now click on the timer0 node in the canvas. You will notice that the "Scriptable Objects" panel takes you to the timer0 object tree.



4. Under the events tree you will see a list of all the events that timer1 allows you to script. Double click on the "OnTimeout" event. VTK Designer 2 will provide you a function stub in the script editor into which you can type your code. *Script code should be written in JavaScript.*

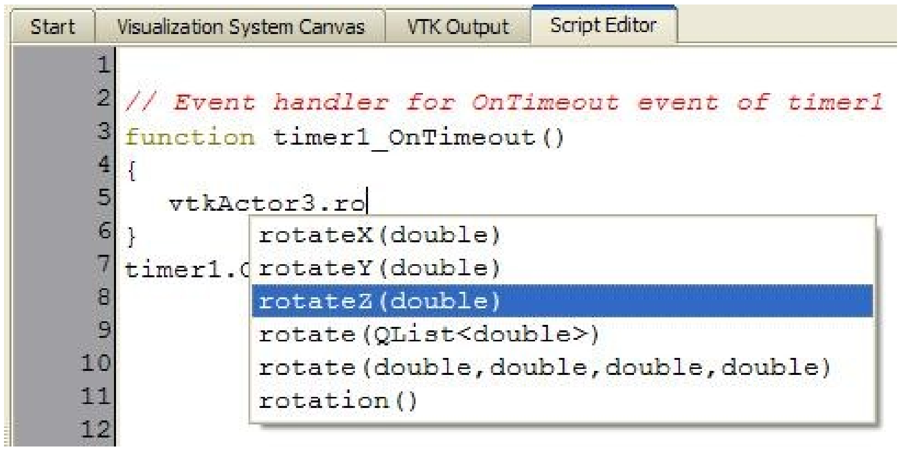


```
// Event handler for OnTimeout event of timer1
function timer1_OnTimeout()
{

}
timer1.OnTimeout.connect(timer1_OnTimeout);
```

Since we want the actor to rotate by 3 degrees when the timeout event is raised, you can type the following code into the script editor.

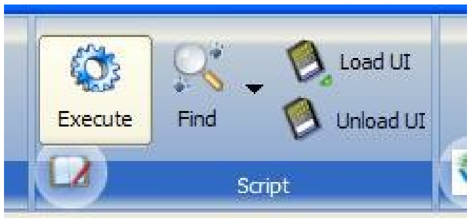
```
vtkActor3.rotateZ(3);
```

The script editor provides context sensitive code completion menus to help you type your code faster.

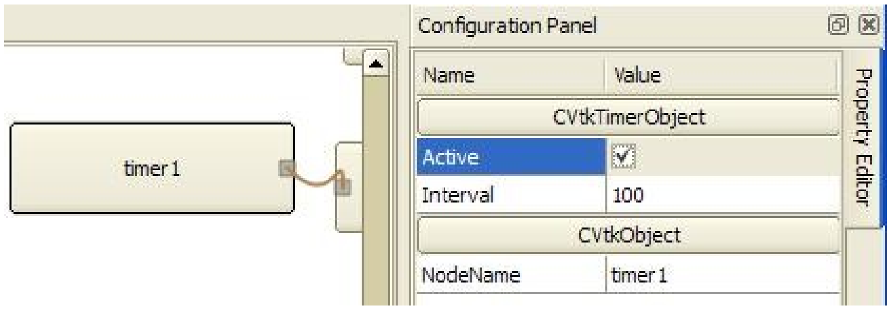


```
// Event handler for OnTimeout event of timer1
function timer1_OnTimeout()
{
    vtkActor3.ro
}
timer1.c
```

5. Once you have finished typing the code, you can click on the "Execute" icon in the "Script" group of the menu strip to execute the code.



6. You can notice now that the cone begins to rotate about its Z axis once you activate the timer. To activate the timer you will have to click on timer1 and turn on the timer in the "Property Editor"



Name	Value
CVtkTimerObject	
Active	✓
Interval	100
CVtkObject	
NodeName	timer1

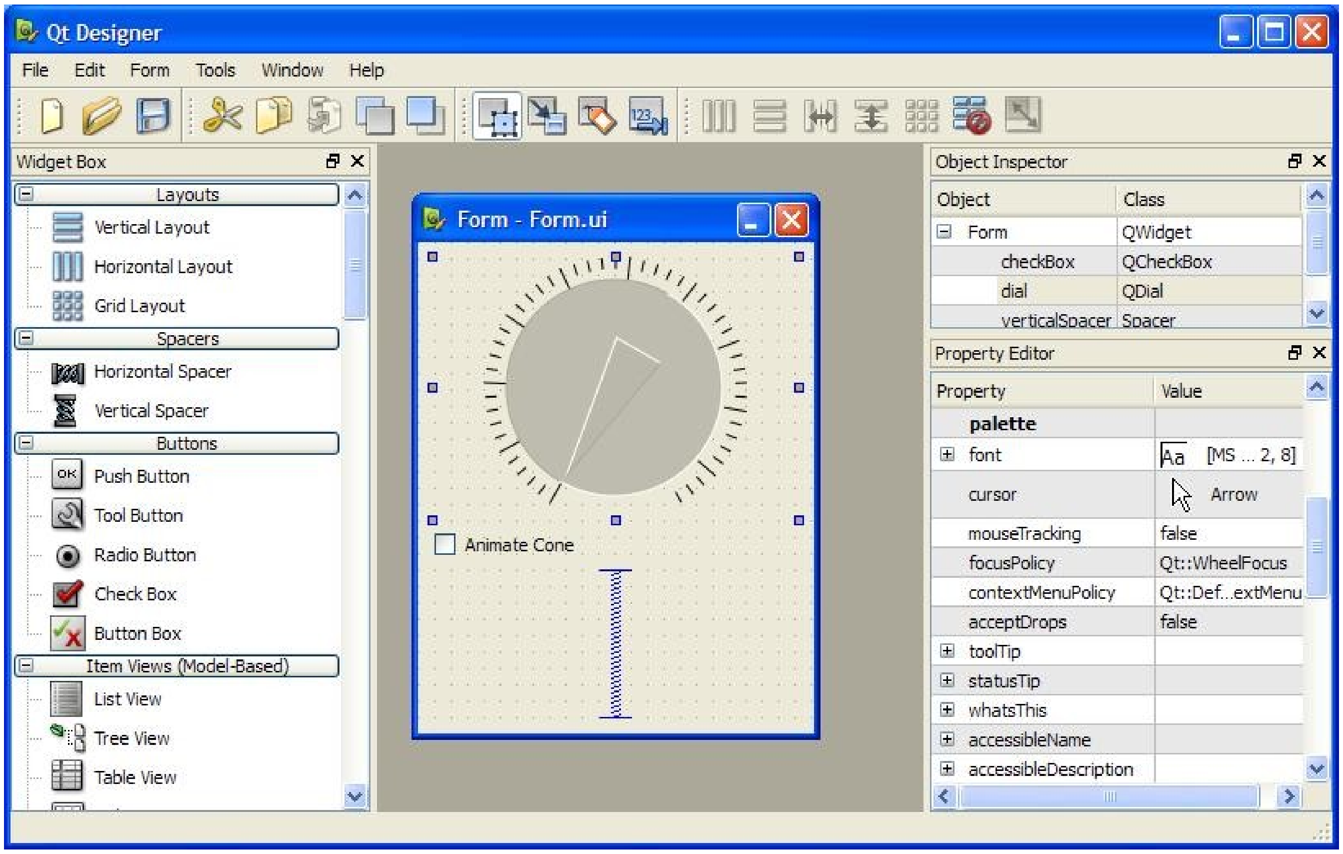
You can increase or decrease the Interval to change the speed of the animation.

In this example we looked at a very simple ways in which the scripting capability of VTK Designer 2 can be made use of. In the coming chapters we will explore more complicated examples.

Importing QtDesigner forms

Sometimes it may be necessary to import custom UI forms into VTK Designer to enable fast and easy interaction. For example suppose that we wanted to provide a simple UI form to activate/deactivate and increase/decrease the interval of the timer created in the previous example. The following steps show you how you can design a custom UI form in QtDesigner and import it into VTK Designer 2.

1. Design the form in Qt designer. *For information on using QtDesigner refer to the QtDesigner user manual shown in QtAssistant.*

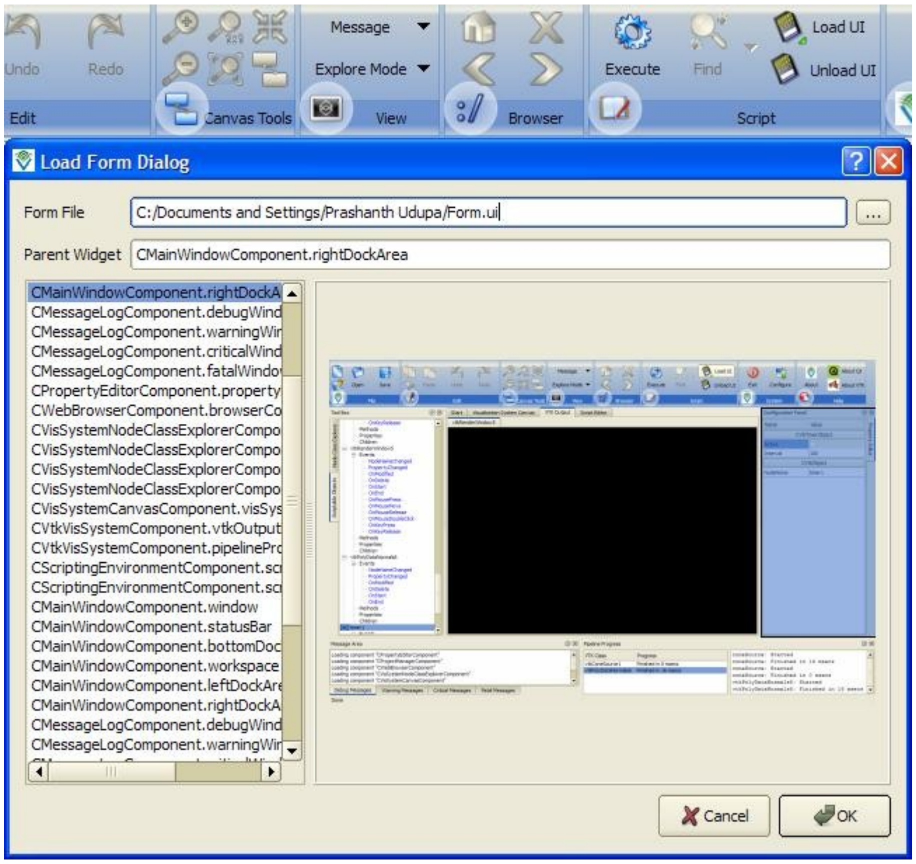


Object	Class
Form	QWidget
checkBox	QCheckBox
dial	QDial
verticalSpacer	Spacer

Property	Value
palette	
font	Aa [MS ... 2, 8]
cursor	Arrow
mouseTracking	false
focusPolicy	Qt::WheelFocus
contextMenuPolicy	Qt::Def...extMenu
acceptDrops	false
toolTip	
statusTip	
whatsThis	
accessibleName	
accessibleDescription	

Save the form as "Form.ui" in your computer.

2. In "VTK Designer", click on the "Load UI" button in the "Script" group of the menu strip. This will show the "Load Form Dialog". Select the "Form.ui" file in the "Form File" field and click Ok. You can choose a region into which the form will be imported, but for now just click Ok.

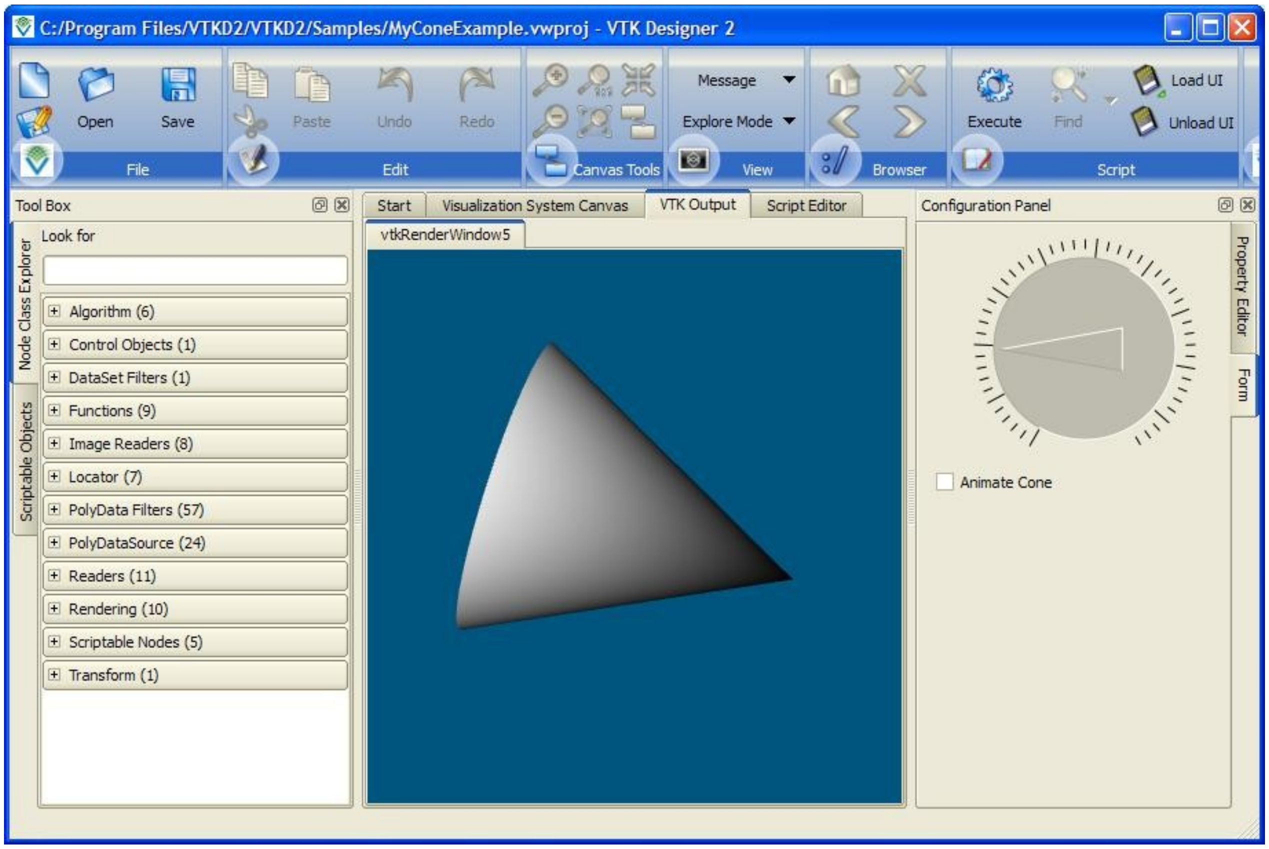


Form File: C:/Documents and Settings/Prashanth Udupa/Form.ui

Parent Widget: CMainWindowComponent.rightDockArea

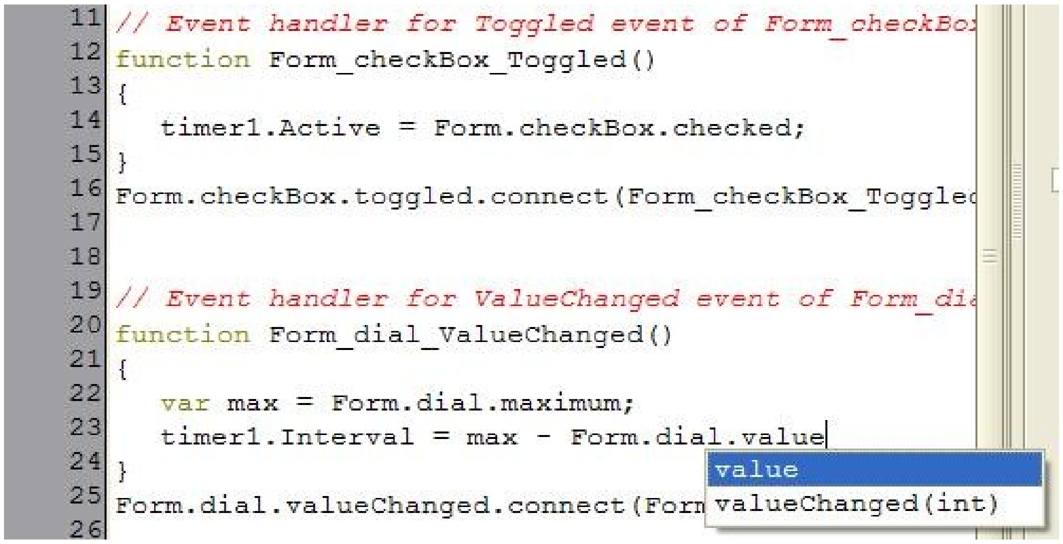
CMainWindowComponent.rightDockA
CMessageLogComponent.debugWind
CMessageLogComponent.warningWir
CMessageLogComponent.criticalWind
CMessageLogComponent.fatalWindo
CPropertyEditorComponent.property
CWebBrowserComponent.browserCo
CVisSystemNodeClassExplorerCompo
CVisSystemNodeClassExplorerCompo
CVisSystemNodeClassExplorerCompo
CVisSystemNodeClassExplorerCompo
CVisSystemCanvasComponent.visSys
CVtkVisSystemComponent.vtkOutput
CVtkVisSystemComponent.pipelinePro
CScriptingEnvironmentComponent.scr
CScriptingEnvironmentComponent.scr
CMainWindowComponent.window
CMainWindowComponent.statusBar
CMainWindowComponent.bottomDoc
CMainWindowComponent.workspace
CMainWindowComponent.leftDockAre
CMainWindowComponent.rightDockA
CMessageLogComponent.debugWind
CMessageLogComponent.warningWir

3. The imported form now shows up in the main window as shown in the screenshot below.



You can notice now that the objects placed on the form and its events are also shown in the "Scriptable Objects" panel. When you click on any widget in the imported UI form, the "Scriptable Objects" panel automatically takes you to the corresponding object tree.

4. Script the check box's Toggled event and the dial's ValueChanged event as follows.



```
// Event handler for Toggled event of Form_checkBo
function Form_checkBox_Toggled()
{
   timer1.Active = Form.checkBox.checked;
}
Form.checkBox.toggled.connect(Form_checkBox_Toggle


// Event handler for ValueChanged event of Form_di
function Form_dial_ValueChanged()
{
   var max = Form.dial.maximum;
   timer1.Interval = max - Form.dial.value
}
Form.dial.valueChanged.connect(For
```

5. Click on the "Execute" button of the "Script" group in the menu strip again. Now you can turn on/off and change the speed of the timer using the custom form.

Loading and Saving Pipelines

VTK Designer 2 allows you to save all aspects of your pipeline into a single file so that you can load it back in the future and continue from where you left off. Within the file VTK Designer 2 stores information about the visualization network (or pipeline), properties of each node in the pipeline, connection configurations, script and even imported forms.

Use the “Open”, “Save”, “Save As” buttons in the “File” group of the menu strip to manage load and save pipelines.

Summary

In this chapter we saw how you can build a simple cone source pipeline using VTK Designer 2. You also saw how you can make use of the scripting and UI import features in VTK Designer to customize the execution and appearance of the pipeline.

In the coming chapters we will look at some interesting visualization problems and how they can be solved using VTK Designer 2. The problems chosen in this chapter are mostly inspired from the examples described in the book Visualization Toolkit 4th Edition, ISBN: 1-930934-19-X.

Mathematical Function Visualization.

Mathematical functions are of the form

$$f(x, y, z) = k$$

k would be equal to 0 if a point $P(x, y, z)$ satisfies the above equation. It would be greater than or less than zero depending on whether the point is inside the function or outside the function. For example consider the equation of a sphere whose radius is r.

$$f(x, y, z) = x^2 + y^2 + z^2 - r^2 = k$$

In the above function k would become zero, only if a point $P(x, y, z)$ lies on the surface of the sphere. It would return a positive real number if the point is outside the sphere, negative real number if the point is inside the sphere.

Visualizing mathematical functions.

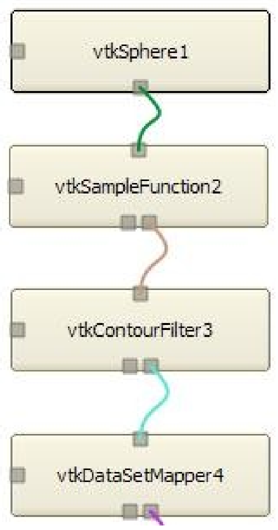
Suppose that you wanted to shape of a function in 3D. The way to do this would be to sample the function in a 3D view volume, extract all the points in the view volume where the value of the function was zero and make a surface out of it.

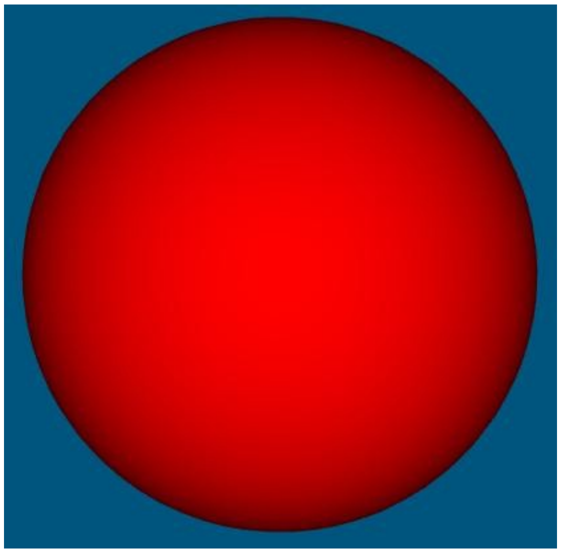
In VTK we make use of the following algorithms to visualize functions.

1. vtkSampleFunction: This algorithm samples the function in a 3D bounding box that can be specified. The output of this function is a 3D image of sampled function values.
2. vtkContourFilter: This algorithm extracts specific points from a 3D image sample and uses an algorithm like marching cubes to construct a surface out of the extracted points.

Shown below is a pipeline constructed in VTK Designer and its corresponding output.

The complete pipeline is not shown here because by now it must be obvious that the mapper would get connected to vtkActor, vtkRenderer and vtkRenderWindow





Now lets replace the vtkSphere1 node with other functions and see how the result changes.

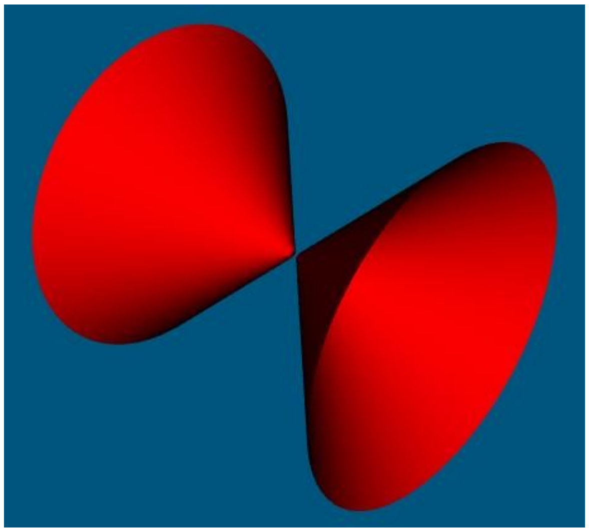


Illustration 2: vtkCone Function

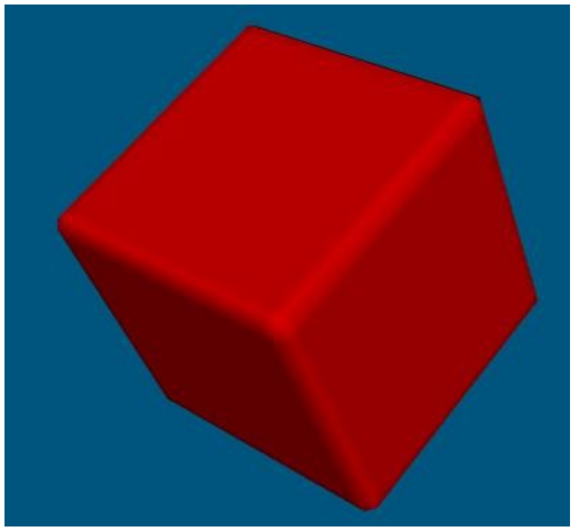


Illustration 3: vtkBox Function

Union, Difference and Intersection of mathematical functions

Suppose we have two functions

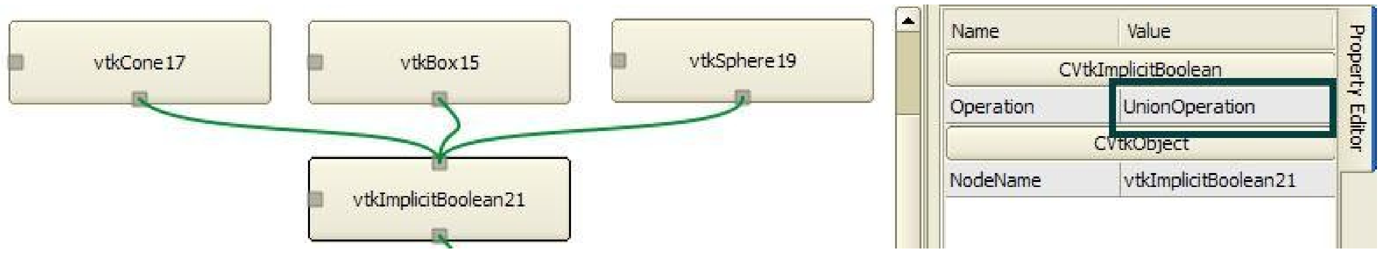
$f(x,y,z)=k$ and $g(x,y,z)=l$

1. Union of the above mathematical functions is defined as $f\,U\,g=k<l\,?\,k:l$
2. Intersection is defined as $f\cap g=k>l\,?\,k:l$

Now if we want to visualize the union or intersection of two or more mathematical functions then we need to perform the union or intersection of those functions and then present the resulting function to vtkSampleFunction. To do this we make use of

1. vtkImplicitBoolean: This function performs a union, intersection and difference operation on one or more input functions and passes the resulting function further down the pipeline.

Show below is the pipeline used to visualize the union of vtkCone, vtkBox and vtkSphere.



Name	Value
CVtkImplicitBoolean	
Operation	UnionOperation
CVtkObject	
NodeName	vtkImplicitBoolean21

Illustration 4: vtkImplicitBoolean Function. Notice that Operation is set to UnionOperation in the property editor

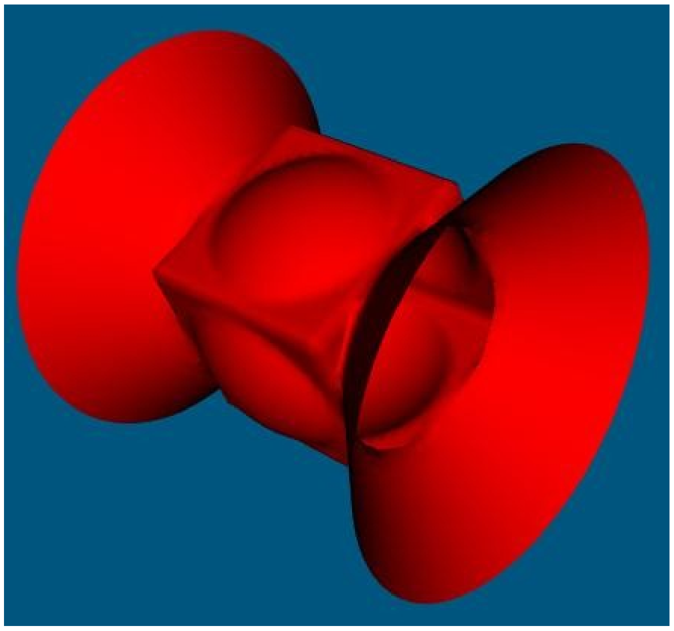


Illustration 5: Cone, Box and Shpere: UnionOperation

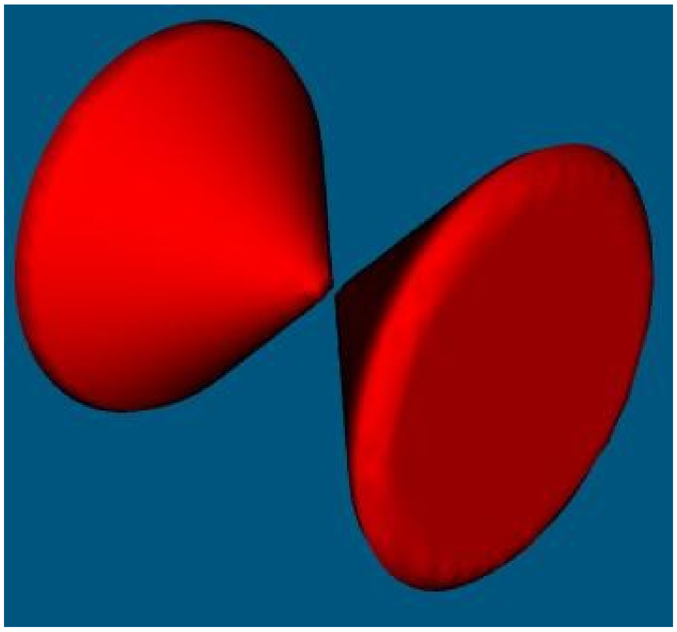
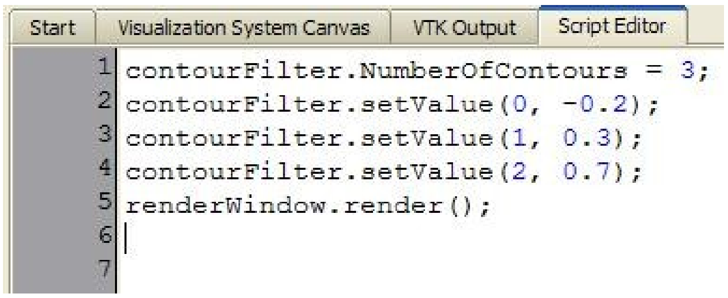


Illustration 6: Cone, Box and Shpere: Intersection

Extracting multiple surfaces in a function

So far we have looked at pipelines where only one surface of the function was extracted. By surface we mean a mesh connecting points at which the function evaluates to a common value. Lets now try and extract multiple surfaces.

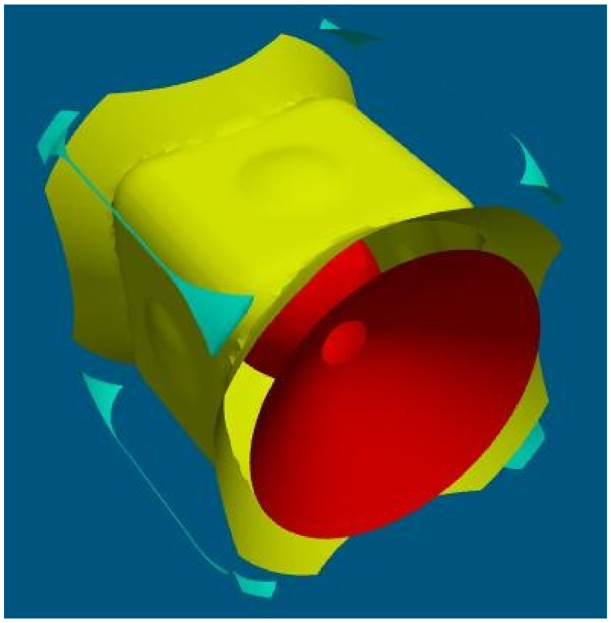
1. Rename the vtkContourFilter3 node to something nicer, like contourFilter. To rename a node just click on it, press F2 and type the new name of the node. Similarly rename the vtkRenderWindow node as renderWindow.
2. Go to the "Script Editor" tab and type the following code.



```
contourFilter.NumberOfContours = 3;
contourFilter.setValue(0, -0.2);
contourFilter.setValue(1, 0.3);
contourFilter.setValue(2, 0.7);
renderWindow.render();
```

In the code above, we are asking the vtkContourFilter algorithm to extract 3 surfaces. The values for each surface is also specified in the code.

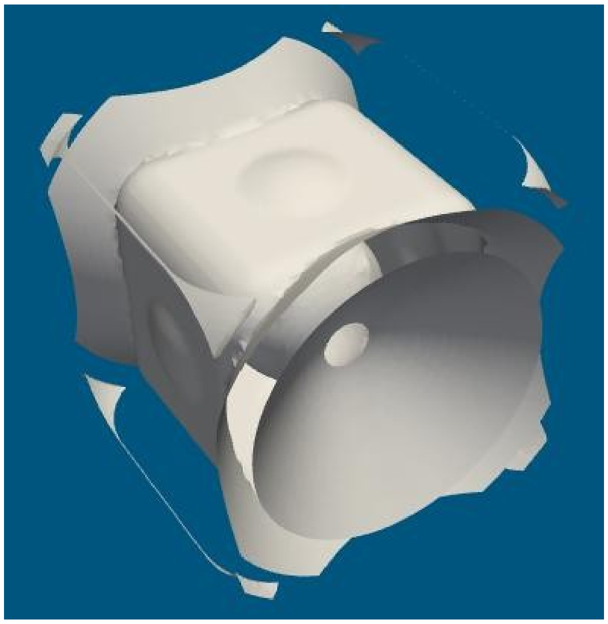
3. When the above code is executed you will see, in the VTK Output tab, an output as shown below.



vtkDataSetFilter automatically assigns a different color for each surface.

1. The red colored surface represents the region where the function value is -0.2
2. The yellow colored surface represents the region where the function value is 0.3
3. The cyan colored surface represents the region where the function value is 0.7

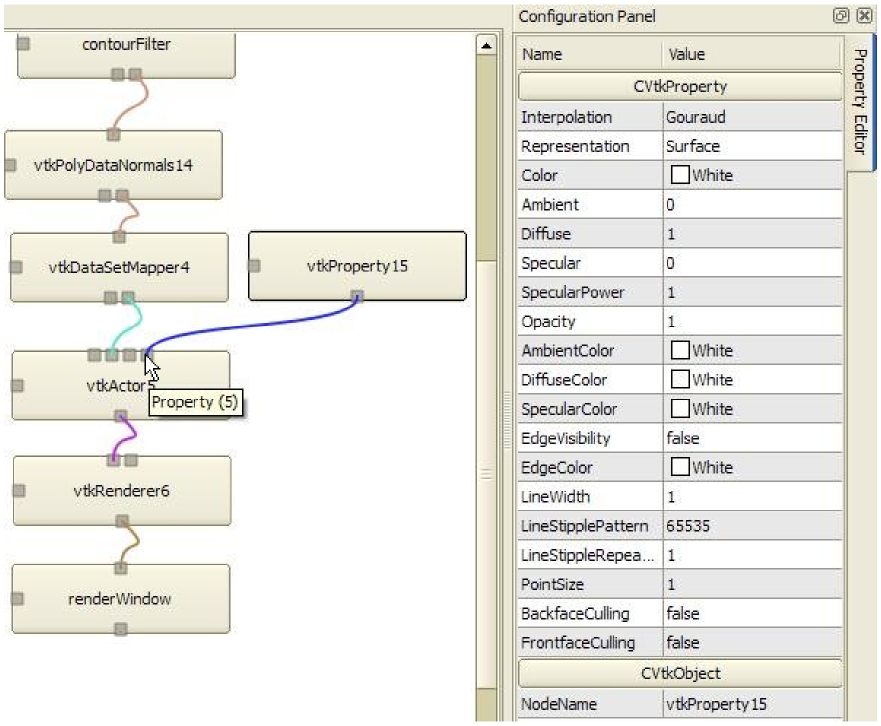
You can turn this off by setting the "ShowScalars" property of vtkDataSetMapper to false. Although that would make the output less meaningful.



Improving the visual appearance.

The default mesh render characteristics work well for most cases. However in sometime you might want to change the way the output is rendered. To do this you have to make use of the "vtkProperty" class.

1. Drag an instance of "vtkProperty" into the canvas and connect it to the instance of "vtkActor" in the scene.

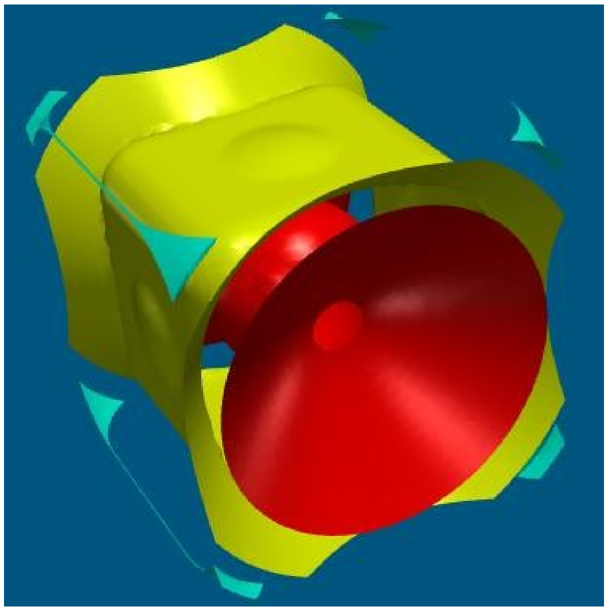


Configuration Panel

Name	Value
CVtkProperty	
Interpolation	Gouraud
Representation	Surface
Color	☐ White
Ambient	0
Diffuse	1
Specular	0
SpecularPower	1
Opacity	1
AmbientColor	☐ White
DiffuseColor	☐ White
SpecularColor	☐ White
EdgeVisibility	false
EdgeColor	☐ White
LineWidth	1
LineStipplePattern	65535
LineStippleRepea...	1
PointSize	1
BackfaceCulling	false
FrontfaceCulling	false
CVtkObject	
NodeName	vtkProperty15

2. Now you can edit the properties of vtkProperty to change the appearance characteristics of your output. When we change the "Diffuse" property to 0.8, Specular to 1 and SpecularPower to 128, the resulting

output would now show up as



Visualizing custom mathematical functions.

With VTK Designer 2 it is now possible to visualize your own mathematical functions. Lets visualize the following function.

$$f(x,y,z)=ax^2+by^2+cz^2-d$$

We will need to assume the values of a, b, c and d to actually visualize a specific function. For this example lets say we assume the following values

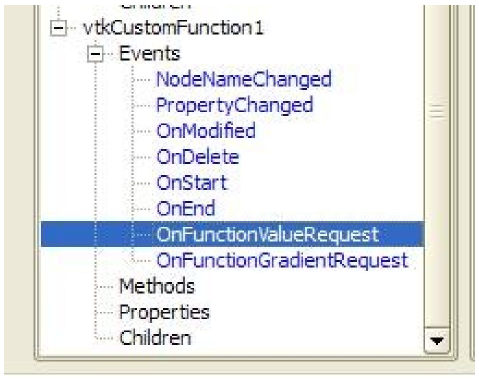
1. a = 1
2. b = 2
3. c = 3
4. d = 4

So the equation now is

$$f(x,y,z)=x^2+2y^2+3z^2-4$$

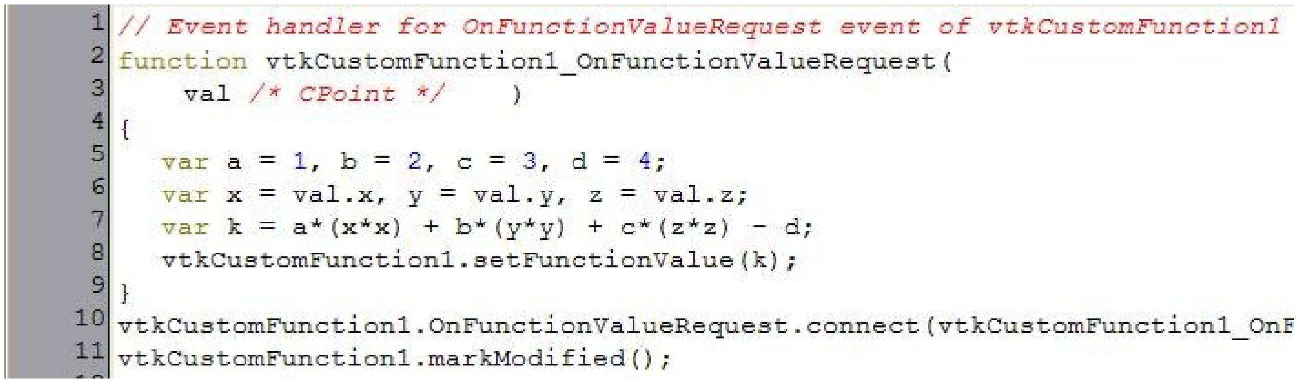
The following steps show you how you can visualize the above function.

1. Start "VTK Designer 2" if you have not already started it. Create a new file by clicking the "New" button on the "File" group of the menu strip.
2. Drag and drop vtkCustomFunction on to the canvas.
3. Go to the "Scriptable Objects" panel and double click on "OnFunctionValueRequest" event under "vtkCustomFunction1".



```
vtkCustomFunction1
    Events
        NodeNameChanged
        PropertyChanged
        OnModified
        OnDelete
        OnStart
        OnEnd
        OnFunctionValueRequest
        OnFunctionGradientRequest
    Methods
    Properties
    Children
```

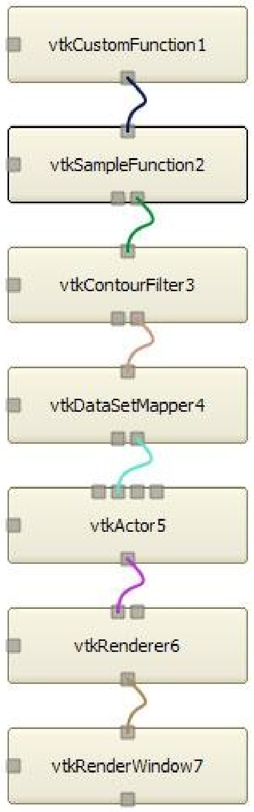
4. In the script editor type the function as shown in the screenshot below.

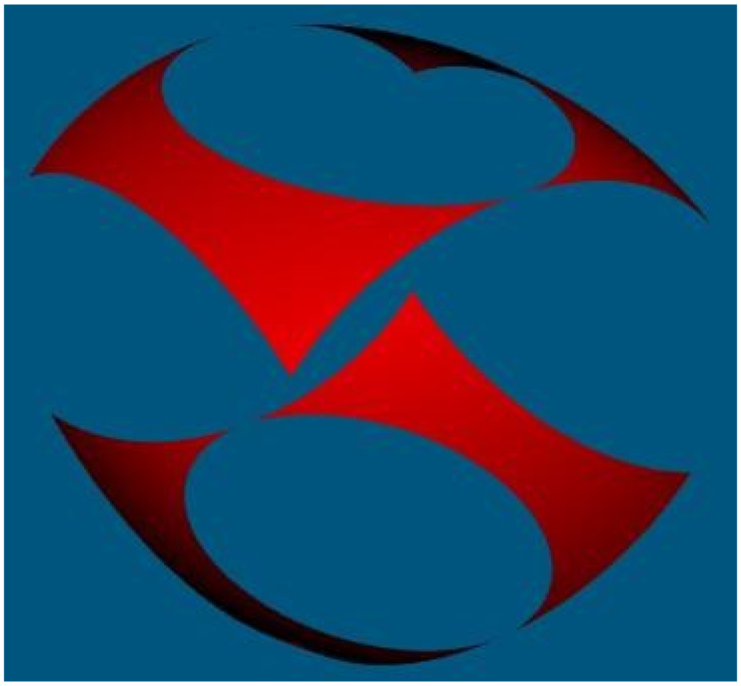


```
// Event handler for OnFunctionValueRequest event of vtkCustomFunction1
function vtkCustomFunction1_OnFunctionValueRequest(
    val /* CPoint */    )
{
   var a = 1, b = 2, c = 3, d = 4;
   var x = val.x, y = val.y, z = val.z;
   var k = a*(x*x) + b*(y*y) + c*(z*z) - d;
   vtkCustomFunction1.setFunctionValue(k);
}
vtkCustomFunction1.OnFunctionValueRequest.connect(vtkCustomFunction1_OnF
vtkCustomFunction1.markModified();
```

The val parameter passed to the event handler function is the point at which the function value is being calculated. The event handler computes the value of the function at that point and send it to the function using the `setFunctionValue()` method. After typing the above code, click on the “Execute” button.

5. Now construct the remaining part of the pipeline as follows and click on the “VTK Output” tab. The pipeline will take sometime (usually 10 to 20 seconds) before it gets fully executed. Once the pipeline execution is done, you can view the output as shown below.





Scripted functions are always slower than native VTK functions. You can increase the speed of the pipeline by

1. Reducing the resolution at which the function is sampled by vtkSampleFunction

OR

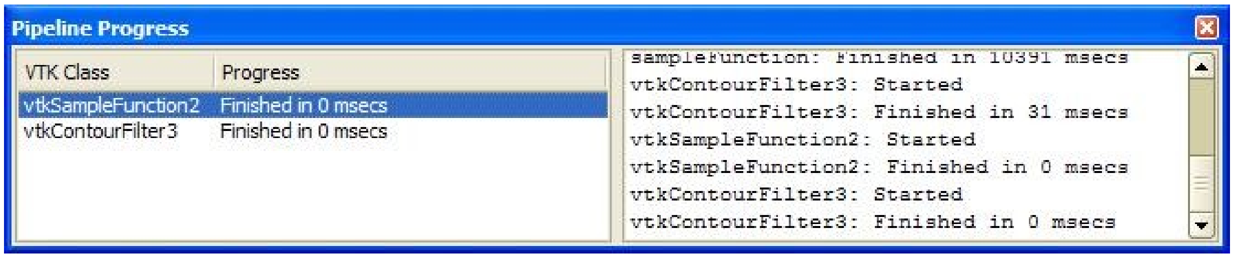
2. Writing a plugin for VTK Designer that evaluates your function at high speed.

We will not discuss the latter because it is beyond the scope of this manual. However we can discuss the former. Go to the script editor and type the following script anywhere outside the event handler code.

```
vtkSampleFunction2.SampleDimensions = new Array(5, 5, 5);
```

Now re-execute the script by clicking on the “Execute” button in the “Script” group of the menu strip, then click on the “VTK Output” tab.

You will notice that although the generated output is not as smooth as before, but the pipeline got executed faster. The “Pipeline Progress” dock window shown near the bottom edge of the main window reports the processing time taken by different algorithms in the pipeline.



VTK Class	Progress
vtkSampleFunction2	Finished in 0 msecs
vtkContourFilter3	Finished in 0 msecs

```
sampleFunction: Finished in 10391 msecs
vtkContourFilter3: Started
vtkContourFilter3: Finished in 31 msecs
vtkSampleFunction2: Started
vtkSampleFunction2: Finished in 0 msecs
vtkContourFilter3: Started
vtkContourFilter3: Finished in 0 msecs
```

You can change the number of contours extracted by vtkContourFilter and/or co-efficients of the custom function and re-execute the script to see some interesting function surface(s).

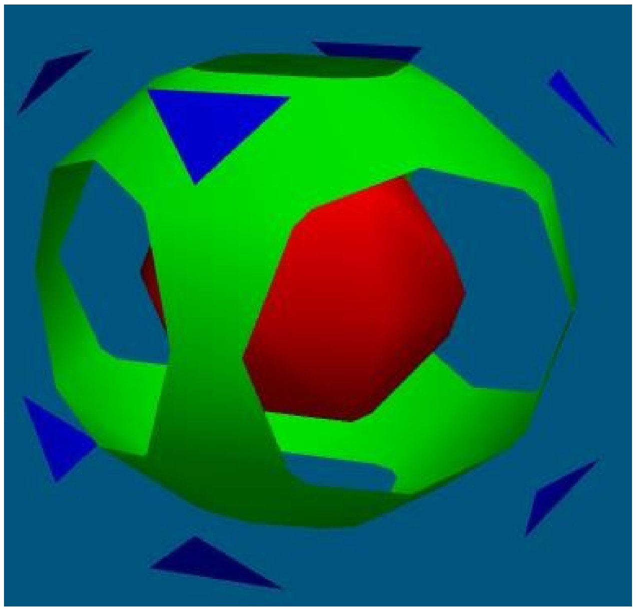


Illustration 7: a=0.5, b=0.45, c=0.65, d=0.3

Visualizing Meshes, Line Sets and Point Sets

VTK makes use of a standard format for representing polygonal data sets. In this format one can represent the geometry and topology of the data set. The geometry and topology is specified either explicitly or implicitly. One can also store special data fields (like scalars, normals, vectors etc) along with the geometry and topology.

Geometry is the description of the location of the vertices or points that make an object. Topology is the description of how the vertices are connected to make the object appear the way it should appear.

Algorithms in VTK either create or process data sets. Algorithms that create data sets are called "Sources". Algorithms that process data sets by either converting it from one form to another or by altering it are called "Filters". Algorithms that read information contained in data sets to render it in OpenGL are called mappers.

We have been using "Sources", "Filters" and "Mappers" all along in different forms. In this chapter lets look at a few ready made algorithms that come with "VTK Designer". But before that lets list out few types of data sets

Point Dataset	This data set contains a set of points.
Line Dataset	This data set contains a set of line segments.
Polygonal Dataset	This data set contains a set of triangles or other higher order polygons.
Image Dataset	This data contains an image.

Source Algorithms

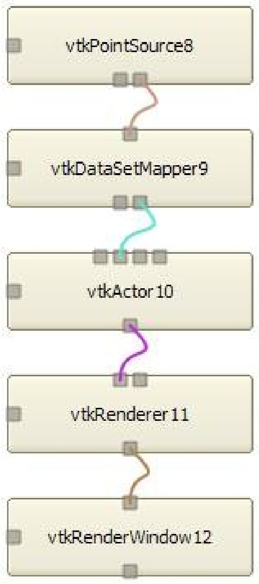
Source algorithms create data sets. They create data sets by

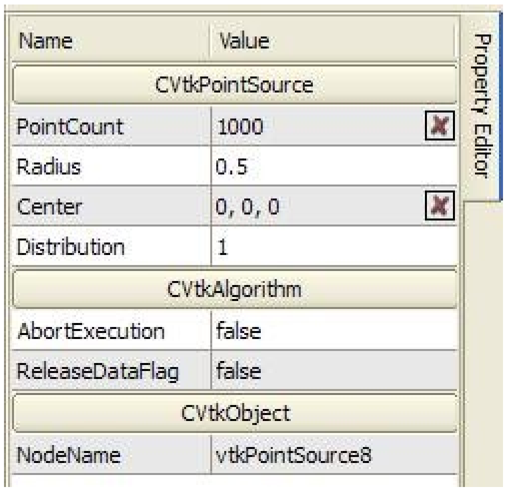
1. Mathematically generating them. (vtkConeSource, vtkCubeSource, vtkSphereSource etc..)
2. Reading the data set information from a file. (vtkPolyDataReader, vtkJPEGReader etc...)
3. Reading the data set from an external device.

PointSet Source Algorithms

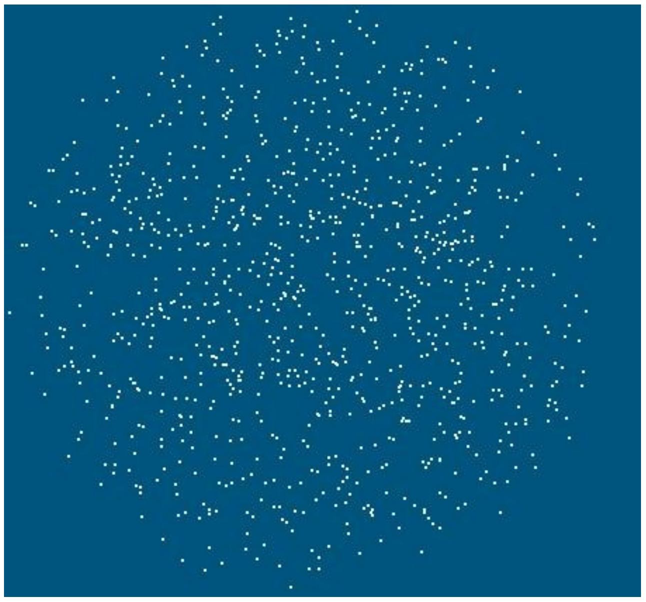
The vtkPointSource algorithm in "VTK Designer 2" can be used to generate a point data set containing randomly scattered points within a given radius from a center.

Shown below is a simple pipeline created using VTK Designer 2 that visualizes the point data set created by vtkPointSet algorithm. The vtkPointSet algorithm in the pipeline shown below is configured to randomly generate 1000 points within a radius of 0.5 units from (0, 0, 0).





Name	Value
CVtkPointSource	
PointCount	1000
Radius	0.5
Center	0, 0, 0
Distribution	1
CVtkAlgorithm	
AbortExecution	false
ReleaseDataFlag	false
CVtkObject	
NodeName	vtkPointSource8



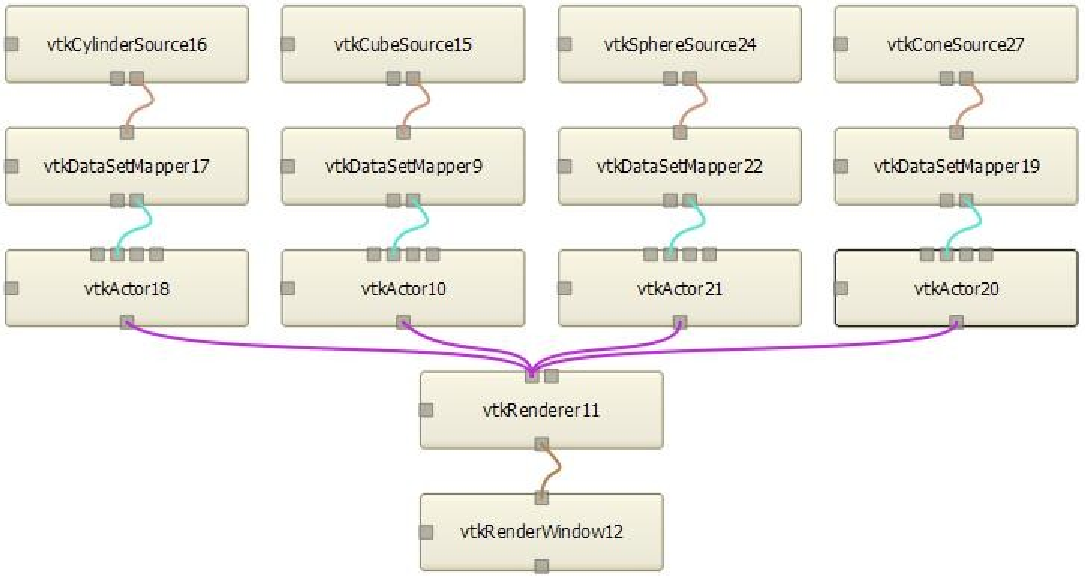
In a later section you will learn how to create custom point set sources.

LineSet Source Algorithms

VTK Designer 2 does not have any inbuilt line set algorithm, but we can create a custom line set source by making use of its scripting capability. More on this in a later section.

Polygonal Dataset Source Algorithms

VTK Designer 2 has quite a few polygonal dataset source algorithms. We have already seen one in the chapter "Cone Source – Your First Pipeline in VTK Designer". Shown below is another example using few polygonal dataset source algorithms.



The above pipeline is configured such that the actors are placed one next to the other. By default all the actors are positioned at (0, 0, 0). When the above pipeline is executed the following output can be viewed.

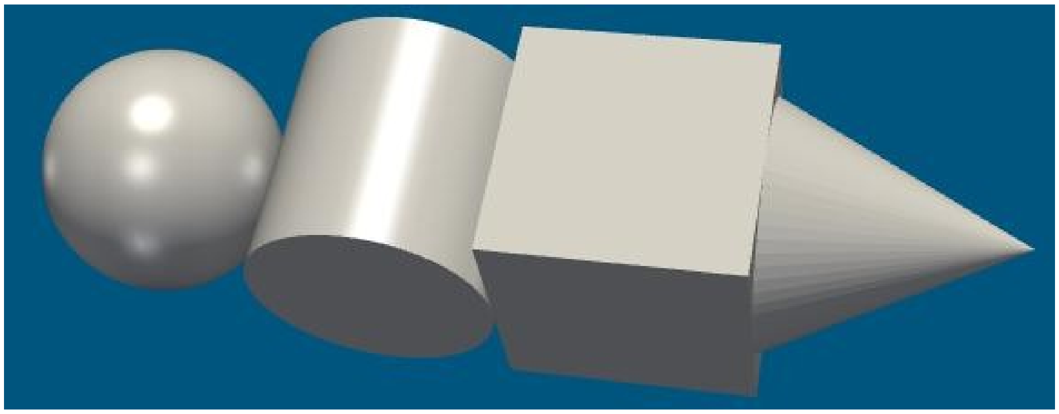
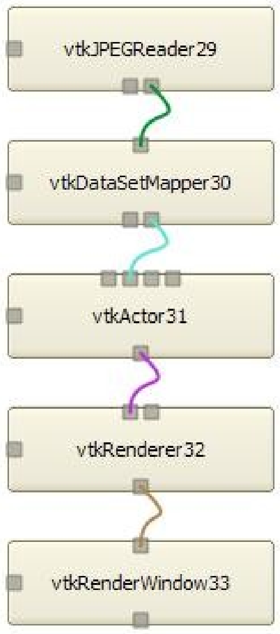
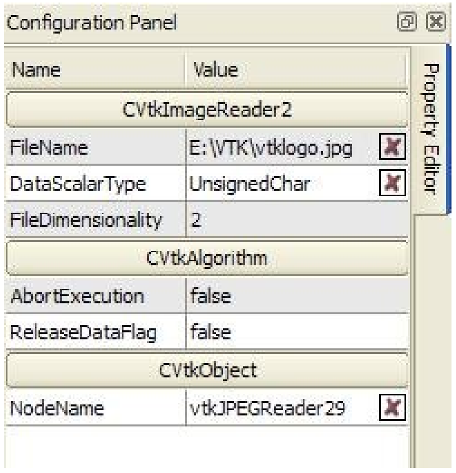


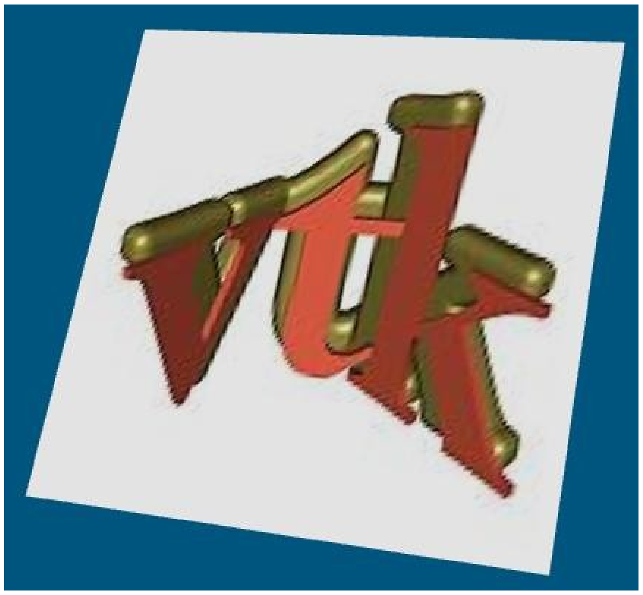
Image Dataset Sources

VTK Designer 2 provides support for image reader classes like vtkJPEGReader, vtkPNGReader, vtkBMPReader etc. Using these classes you can read image from the disk and render it on a render window.

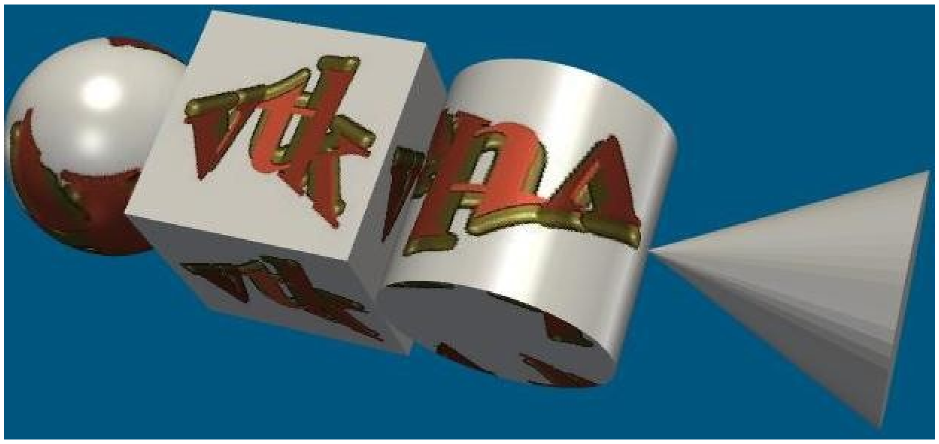


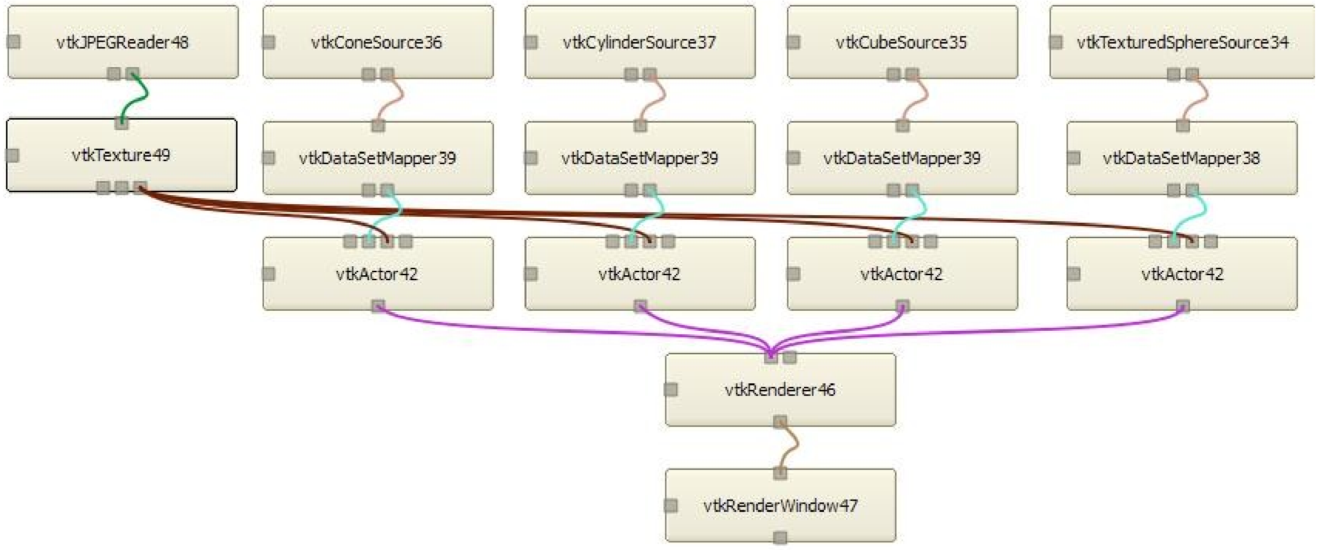


Name	Value
CVtkImageReader2	
FileName	E:\VTK\vtklogo.jpg
DataScalarType	UnsignedChar
FileDimensionality	2
CVtkAlgorithm	
AbortExecution	false
ReleaseDataFlag	false
CVtkObject	
NodeName	vtkJPEGReader29



One of the interesting uses of image readers is that it can be used to load a texture and paste it on a polygonal mesh. For example take a look at the following output and the corresponding pipeline below it.





The texture was not mapped on the cone, because vtkConeSource did not generate texture coordinates for the cone.

Filter Algorithms

Filter algorithms accept a data set as input and gives out a modified or new dataset(s) as output. For example in the chapter on "Mathematical Function Visualization" we made use of vtkContourFilter to extract surfaces from the image dataset provided by vtkSampleFunction. More about filters in the next chapter.

Mapper Algorithms

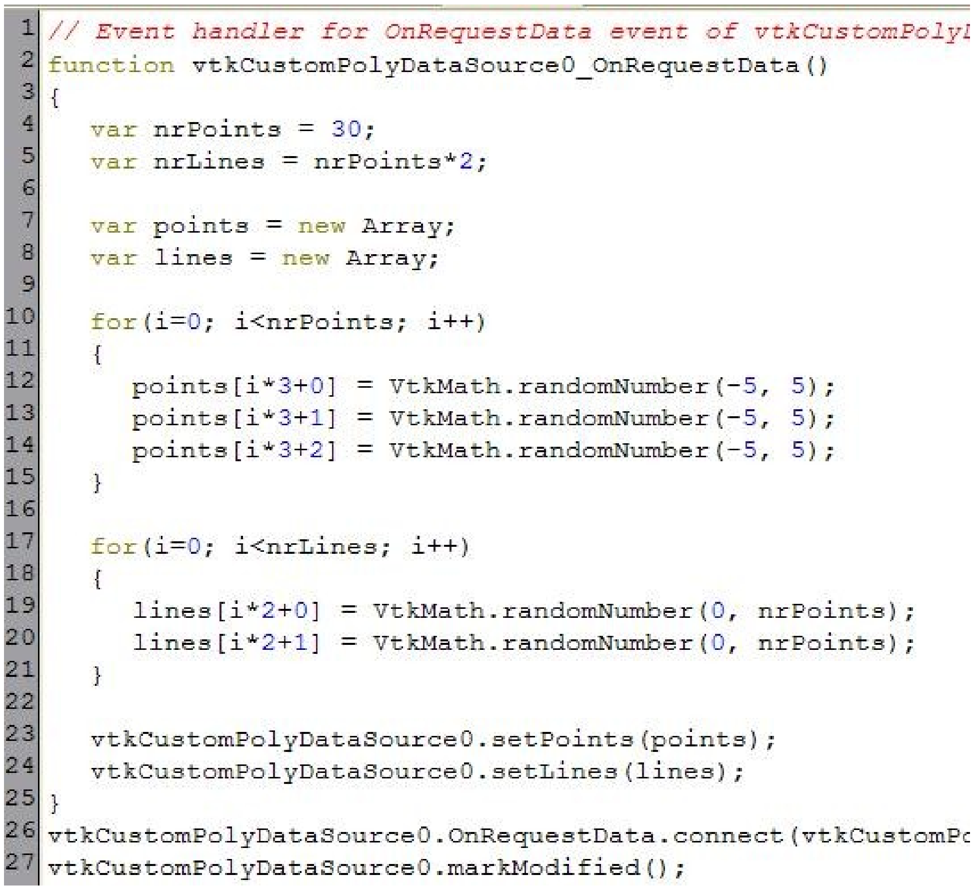
Mapper algorithms read the data contained in an input dataset and render them using OpenGL. VTK Designer supports vtkDataSetMapper and vtkPolyDataMapper classes from VTK. These mappers are sufficient for a wide range of mapping tasks. A complete discussion about mappers is beyond the scope of this document.

Custom Sources

With VTK Designer 2 you can now create custom pointset, lineset, polygonal and image datasets. Creating custom datasets is similar in concept to custom functions.

Lets understand the concepts behind creating custom datasets by visualizing a custom "spider-web" line dataset.

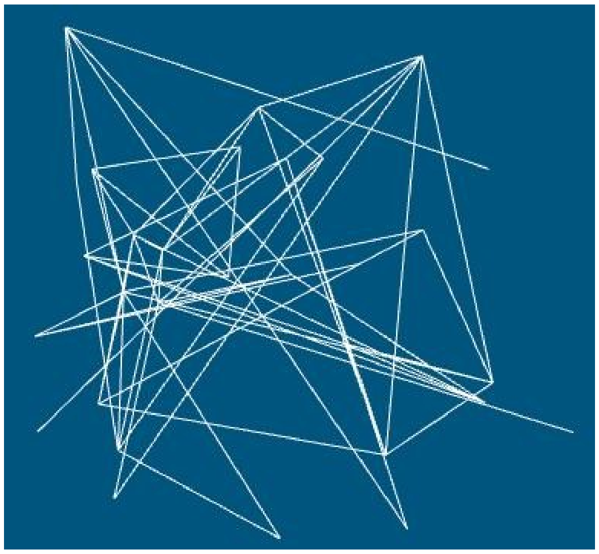
1. Start a new pipeline in VTK Designer.
2. Drag an instance of vtkCustomPolyDataSource into the pipeline.
3. Open the "Scriptable Objects" panel. Click on the vtkCustomPolyDataSource instance that you just dropped into the pipeline. You will notice that the events supported by vtkCustomPolyDataSource are listed in the "Scriptable Objects" panel. Double click on the "OnRequestData" event.
4. Type the following code into the script editor to generate a random lineset web.



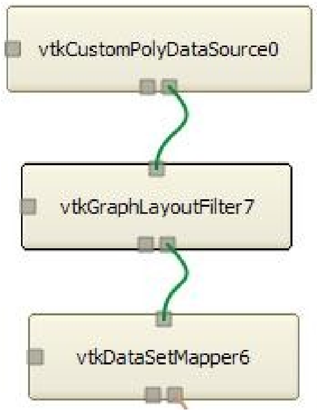
```
// Event handler for OnRequestData event of vtkCustomPolyl
function vtkCustomPolyDataSource0_OnRequestData()
{
   var nrPoints = 30;
   var nrLines = nrPoints*2;

   var points = new Array;
   var lines = new Array;

   for(i=0; i<nrPoints; i++)
   {
      points[i*3+0] = VtkMath.randomNumber(-5, 5);
      points[i*3+1] = VtkMath.randomNumber(-5, 5);
      points[i*3+2] = VtkMath.randomNumber(-5, 5);
   }

   for(i=0; i<nrLines; i++)
   {
      lines[i*2+0] = VtkMath.randomNumber(0, nrPoints);
      lines[i*2+1] = VtkMath.randomNumber(0, nrPoints);
   }

   vtkCustomPolyDataSource0.setPoints(points);
   vtkCustomPolyDataSource0.setLines(lines);
}
vtkCustomPolyDataSource0.OnRequestData.connect(vtkCustomPc
vtkCustomPolyDataSource0.markModified();
```

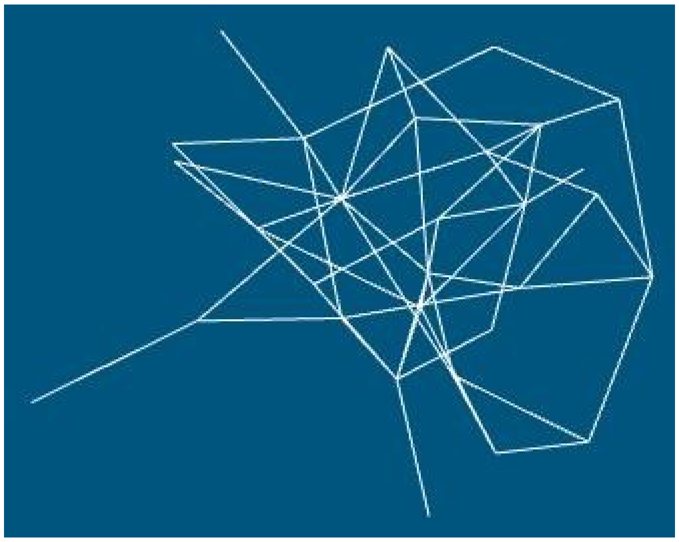
Execute the script.

5. Connect the rest of the pipeline. (vtkDataSetMapper, vtkActor, vtkRenderer and vtkRenderWindow).
6. Click on the "VTK Output Tab" to view the web that was just created.



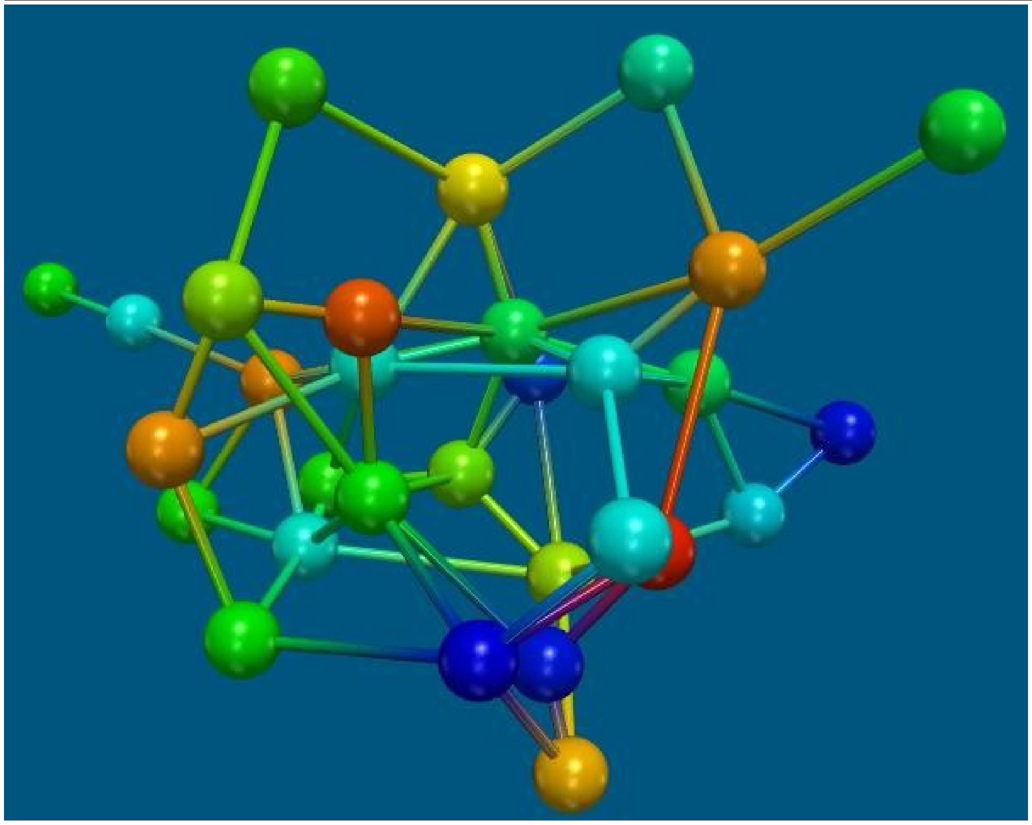
Let us now try to beautify the web by properly laying out the lines using the vtkGraphLayoutFilter.





Exercise:

After learning about glyph filters in the next chapter, come back to this example and figure out how you can convert the above web into something like this



Summary

VTK makes use of a standard format for representing polygonal data sets. In this format one can represent the geometry and topology of the data set. The geometry and topology is specified either explicitly or implicitly. One can also store special data fields (like scalars, normals, vectors etc) along with the geometry and topology.

There are primarily four types of datasets: point, line, polygonal and image datasets.

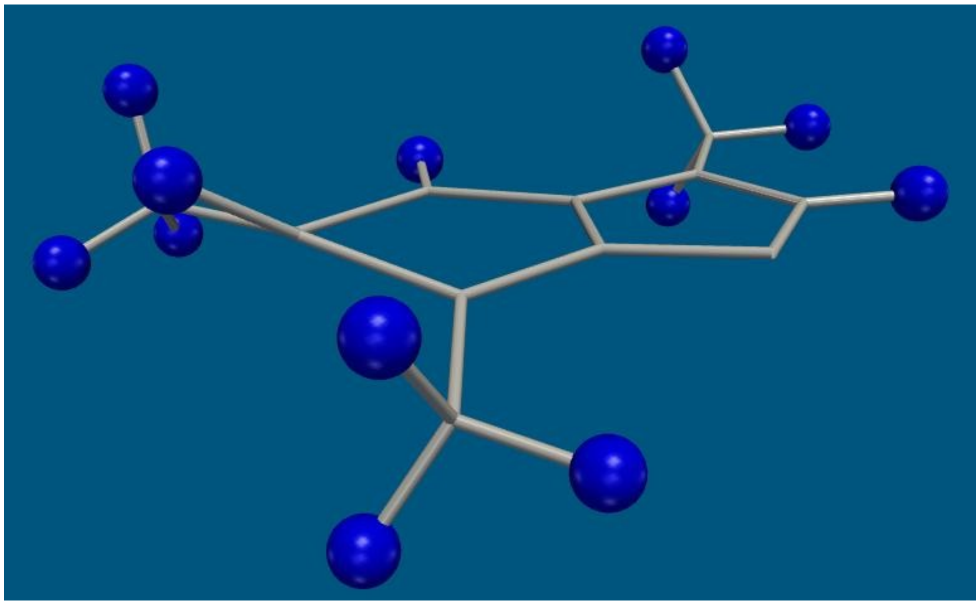
VTK Designer 2 supports creation of custom datasets via its "vtkCustomPolyDataSource" algorithm.

Data Set Filters in VTK Designer

Filter algorithms accept a data set as input and gives out a modified or new dataset(s) as output. For example in the chapter on "Mathematical Function Visualization" we made use of vtkContourFilter to extract surfaces from the image dataset provided by vtkSampleFunction. In this chapter we will take a look at few simple data set filters in VTK Designer.

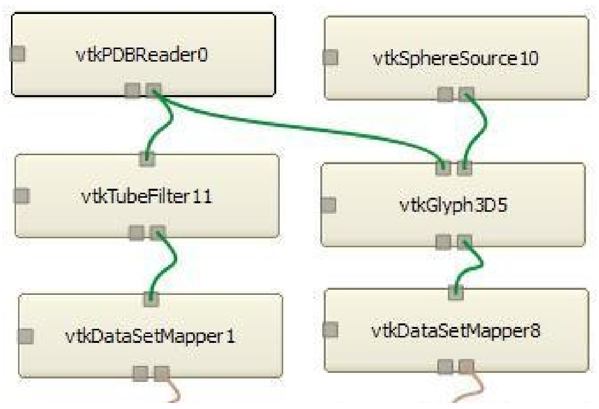
Glyph Filter

VTK Designer 2 supports a filter algorithm called vtkGlyph3D that copies a geometric representation (called as glyph) to every point in an input dataset. For example lets say you wanted to visualize a chemical molecule in 3D, you can make use of the vtkPDBReader to read the molecule data. Molecule data would mostly be a line dataset. To make it look like a molecule you might want to show a sphere at every point in the line dataset.



The standard VTK Datafile set from www.vtk.org includes a the PDB (Protein DataBase) file called caffeine.pdb.

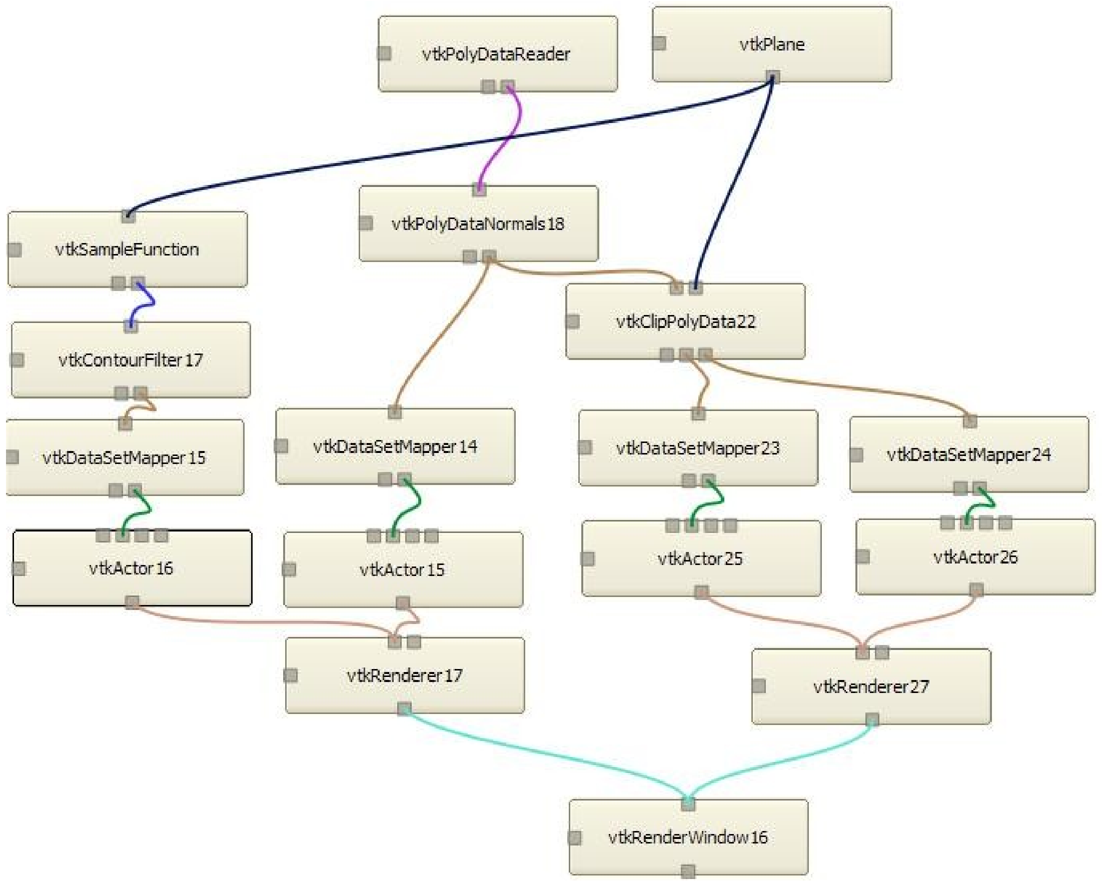
This data file when read and visualized using the pipeline shown in the next page, gives an output as shown here.



1. vtkPDBReader reads from caffeine.pdb and provides a line dataset as output.
2. vtkTubeFilter generates a tube mesh around line segments in the input dataset given by vtkPDBReader.
3. vtkGlyph3D places at every point in the input dataset, given by vtkPDBReader, the polygonal mesh given by vtkSphereSource.

Clipping Filter

VTK Designer 2 supports a filter algorithm called vtkClipPolyData[3] that can cut an input data set into two parts about a function.

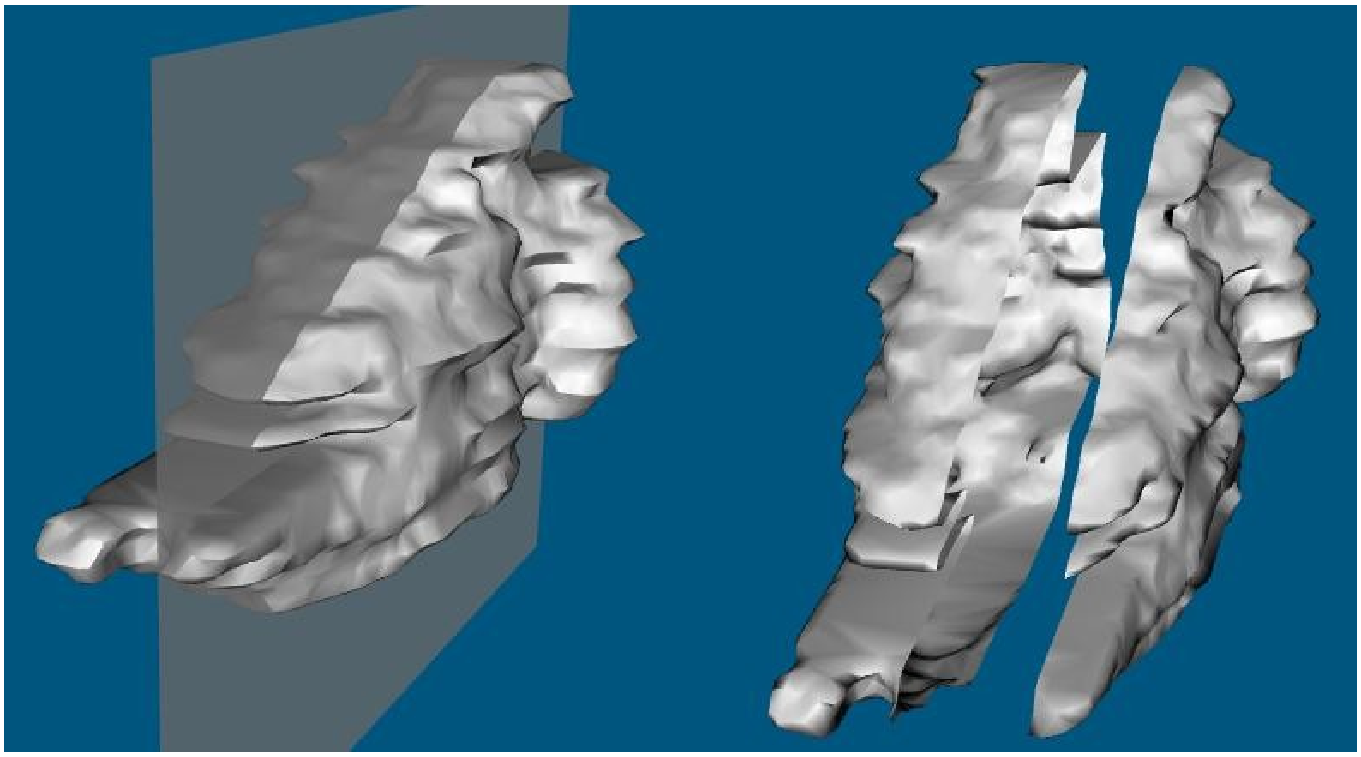


In the pipeline above,

1. we are making use of "vtkPolyDataReader" to read a polygonal dataset from an on-disk file.
2. The read polygonal dataset is then passed through a "vtkPolyDataNormals" filter to smoothen the mesh.
3. The output of which is visualized using a mapper and rendered on to a viewport (vtkRenderer17) in the render window.
4. In a parallel path vtkPlane mathematical function is used by vtkClipPolyData to split the smoothened mesh given out by vtkPolyDataNormals into two parts. The plane function is visualized by making use of vtkSampleFunction and vtkContourFilter. This helps in visualizing where the clipping is done.
5. These two parts, cut by vtkClipPolyData, are rendered into yet another viewport on the same render window.

Shown below is the input data set (left) and the two clipped parts (right).

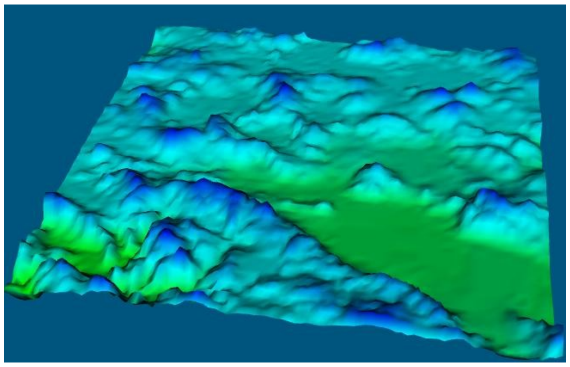
3 You can also vtkClipDataSet



Instead of the vtkPlane function we can make use of a custom function (via scripts) to cut the input dataset about any mathematical function.

Custom Filters

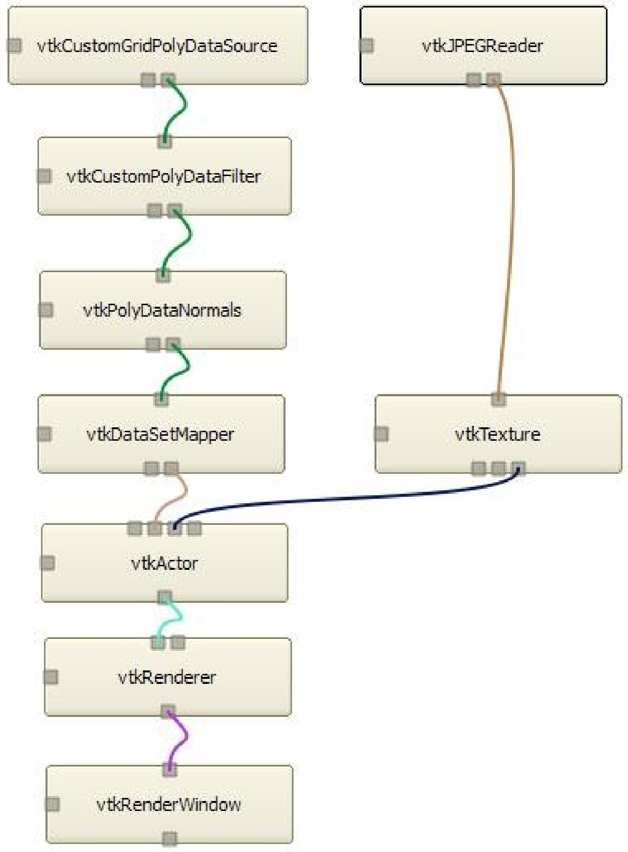
Just like you can create custom functions and custom dataset sources, you can also create custom filters in VTK Designer 2. Lets create a "bump map" like filter to understand custom filters in VTK Designer 2. The goal of our "bump map" like filter is to load a gray scale image and convert it into a 3D terrain polygonal mesh. In-short the filter should convert

to 

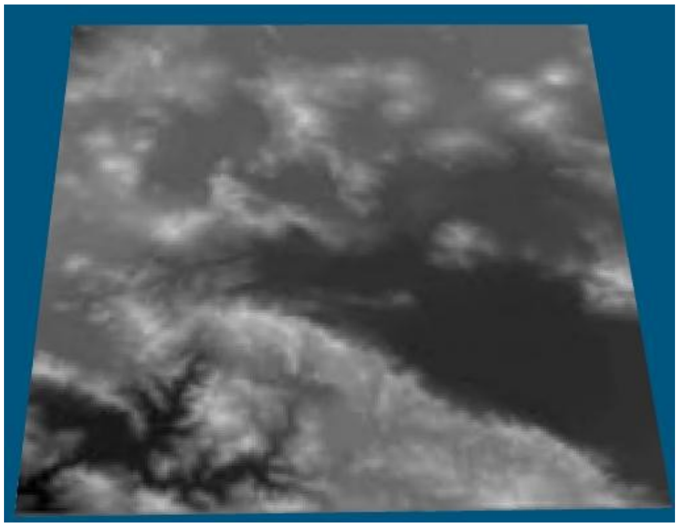
Bump mapping is a computer graphics technique where at each pixel, a perturbation to the surface normal of the object being rendered is looked up in a heightmap and applied before the illumination calculation is done (see, for instance, Phong shading). The result is a richer, more detailed surface representation that more closely resembles the details inherent in the natural world. Normal mapping is the most commonly used bump mapping technique, but there are other alternatives, such as parallax mapping.

Lets first wireup a simple pipeline to create a grid and paste the grayscale terrain image as a texture on the grid.

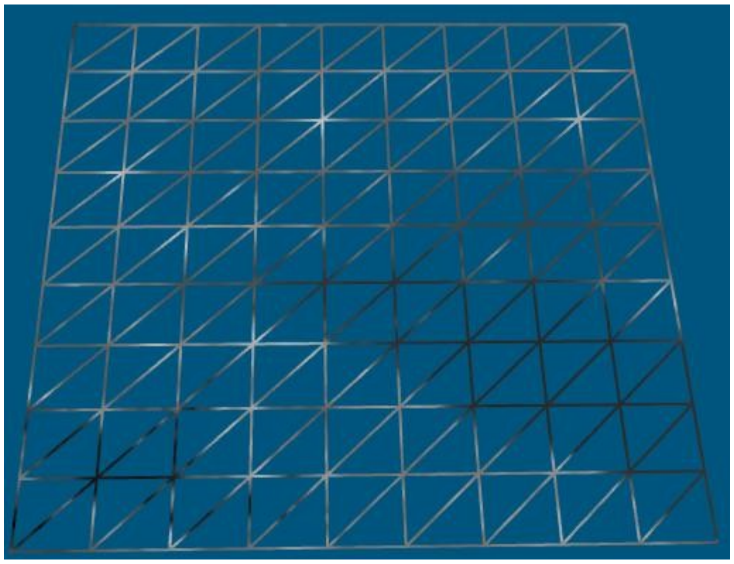
1. Start a new pipeline in VTK Designer 2.
2. Construct the following pipeline



3. Configure "vtkJPEGReader" to load the image C:\Program Files\VTKD2\VTKD2\Samples\terrain.jpg. Configure "vtkPolyDataNormals" to flip normals.
4. Click on the "VTK Output" tab to view the output as shown in the image below



vtkCustomGridPolyDataSource generates a regular grid mesh. To view the grid in wireframe, click on the render window and press the 'W' key. To return to surface rendering mode, press the 'S' key.



By default the vtkCustomGridPolyDataSource algorithm generates a regular 10 x 10 grid. Each point on the grid is spaced apart by 0.1 units. When the grid is viewed in wireframe mode, you will notice the color of pixels on the image that gets super imposed on the grid lines and grid points. In our "bump map" like filter we should capture the color at every grid point and alter the height (or z value) of the point based on the color. For example we could increase the height for grid points that have a white pixel from the texture on them, and decrease the height for the ones that have a black pixel from the texture on them.

The code for the custom filter is written in the event handler for the OnRequestData event of vtkCustomPolyDataFilter.

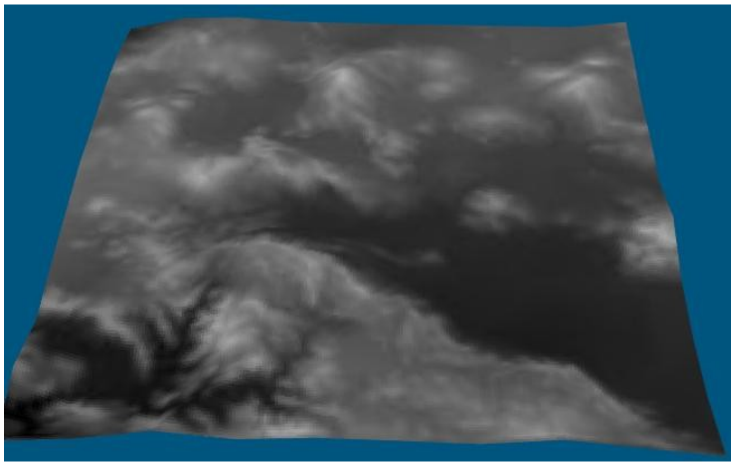
At the time of writing only custom polygonal dataset filters were supported in VTK Designer.

The event handler code for OnRequestData is as shown below.

```
   // Event handler for OnRequestData event of vtkCustomPolyDataFilter
```

```
function vtkCustomPolyDataFilter_OnRequestData()
{
    var imageData = vtkJPEGReader.imageDataObject();
    var dim = imageData.dimensions();
    var gridX = vtkCustomGridPolyDataSource.GridCountX;
    var gridY = vtkCustomGridPolyDataSource.GridCountY;
    var gridXF = dim[0] / gridX;
    var gridYF = dim[1] / gridY;
    var index = 0;
    var nrPoints = gridX*gridY;
    var maxHeight = (gridX * vtkCustomGridPolyDataSource.GridDistanceX)/30;

    for(i=0; i<gridX; i++)
    {
        var t = i*gridXF;
        for(j=0; j<gridY; j++)
        {
            var s = j*gridYF;
            var scalar = imageData.scalarComponent(s, t, 0);
            var zVal = maxHeight*(scalar[0] + scalar[1] + scalar[2])/256;
            var point = vtkCustomPolyDataFilter.point(index);
            point.z = zVal;
            vtkCustomPolyDataFilter.setPoint(index, point);
            index = index + 1;
        }
    }
}
vtkCustomPolyDataFilter.OnRequestData.connect(vtkCustomPolyDataFilter_OnRequestData);
vtkCustomPolyDataFilter.markModified();
vtkRenderer.resetCamera();
vtkRenderWindow.render();
```

The above filter code basically queries the color of the pixel in the texture that is mapped on to each and every point in the grid. This color is used to determine the height of the grid point. When the above script is executed you will see a "bump mapped" grid as shown below.

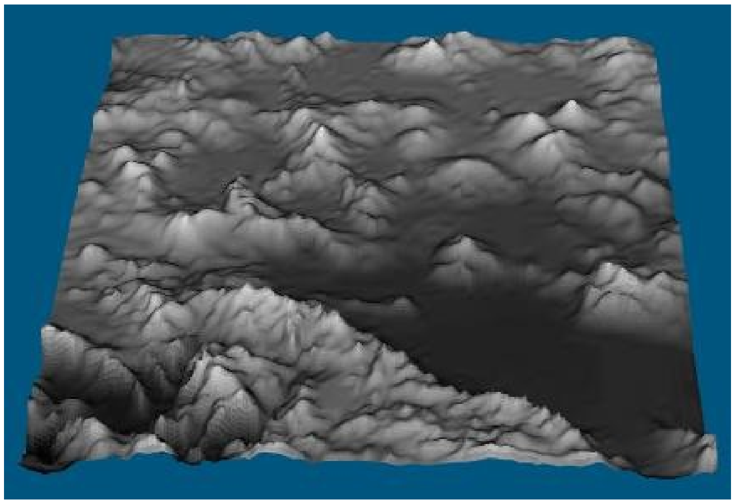


To improve the quality of the "bump mapped" grid, all you have to do is

1. Configure the GridX and GridY properties of vtkCustomGridPolyDataSource to 150 (some high enough

value)

2. Configure the FeatureAngle property of vtkPolyDataNormals to 120



Summary

Filter algorithms accept a data set as input and gives out a modified or new dataset(s) as output.

vtkGlyph3D filter copies a geometric representation (called as glyph) to every point in an input dataset.

vtkClipPolyData filter can cut an input data set into two parts about a function.

vtkCustomPolyDataFilter class can be scripted to created custom filter algorithms.