ActiViz .NET User's Guide

Version 9.5

http://www.kitware.com

For More Information:

- Kitware provides training for VTK and ActiViz. Learn more at https://www.kitware.eu/training
- Kitware provides support and consulting services for ActiViz at https://www.kitware.eu/what-we-offer/#support
- Books explaining the theory and use of VTK are available from https://www.kitware.eu/what-we-offer/#books
- VTK examples and documentation

Contributors:

- Jeff Baumes — Wikipedia Browser
- David Cole — Technical Lead
- Jon Crall — Technical Contributor, Wikipedia Browser, examples
- Bill Hoffman — Technical Contributor
- Niki Russell — Documentation, web support
- Will Schroeder — Documentation, examples
- Tristan Coulange — Technical Contributor
- Lucas Gandel — Technical Lead
- Alexy Pellegrini — Technical Contributor

and the world-wide VTK community at http://www.vtk.org.

Join the VTK Community at http://www.vtk.org

Table of Contents

1. Introduction

Welcome to the ActiViz .NET User's Guide. This document is organized into several parts: introduction, installation guide, tutorial by example, and additional information. The introduction provides a high-level overview of ActiViz .NET. The installation guide explains how to install the software. Once installed, you can try out the examples distributed with this software, and follow the tutorial by example section in preparation to writing your own code. Finally, if you would like to learn more, see the final section which provides links to related information.

1.1. What is ActiViz .NET?

ActiViz .NET provides an integration layer for The Visualization Toolkit (see the next section for more information about VTK) enabling VTK to be used in the Microsoft .NET framework. This means that you can tap into the power of VTK using .NET programming languages such as C# and Visual Basic .NET. ActiViz .NET is designed for the application developer creating software in the Microsoft .NET framework. While the system does come with a useful application (i.e., the Wikipedia Browser), realizing the full power of ActiViz .NET requires you to write software programs.

1.2. What is VTK?

VTK, or the Visualization Toolkit, is an object-oriented software system for 3D graphics, data visualization, data processing, human-computer interaction, information visualization, volume rendering and much more. It has been under development for decades, and is used by researchers, developers and businesses from around the world. VTK is used to view large datasets in supercomputing environments, and is used in US National Labs, research organizations, and supercomputing centers. Tens of millions of dollars of labor have been invested in the system from commercial entities, government funding, and the open-source community. It is used for a diverse set of applications, including volume rendering, medical imaging (http://www.slicer.org), visualization (http://www.paraview.org), and many, many more.

1.3. How ActiViz .NET Differs from VTK

VTK is an open-source system written in C++ that you can download and use for free. However, using VTK requires significant C++ developer skills, and VTK does not easily integrate into the Microsoft development environment. ActiViz .NET provides the appropriate integration layer so that VTK seamlessly fits into the .NET framework. This means that you can use languages such as C# and Visual Basic to add powerful 3D

visualization capabilities to your own applications. This integration layer provides the benefits of the .NET layer including on-line documentation and intelligent coding.

ActiViz interfaces with many existing applications and frameworks written in C#, including WindowsForm, Windows Presentation Foundation (WPF), WinUI 3, Avalonia, and the Unity software. This enables a seamless and fast integration of advanced algorithms and rendering techniques in various environments, Windows, Linux and MacOS.

1.4. Licensing

While VTK is under the BSD 3-Clause License, ActiViz is proprietary and has its own license that allows you to redistribute Activiz as part of your commercial application without having to purchase additional licenses for the end-user.

Read the ActiViz license agreement or contact us for more information.

2. Getting ActiViz .NET

Request a trial version of ActiViz .NET here: https://www.kitware.eu/product/activiz
You can find details about the latest version of ActiViz and its pricing at this link : https://www.kitware.eu/get-activiz.

Notes:
The latest “Open-Source” version available is ActiViz .NET 5.8.0.
The latest “Supported” version available is ActiViz .NET 9.5.

3. Installing ActiViz .NET

This section describes how to download and install ActiViz .NET, and how to configure Microsoft Visual Studio to use ActiViz .NET on Windows.

Packages for Linux and MacOS are provided as archives. They can be made available locally to NuGet command-line tools after installing dotnet. Please refer to the Command line section.

Please refer to the Microsoft documentation for dotnet installation instructions for your system https://dotnet.microsoft.com/en-us/download.

3.1. System Requirements

ActiViz .NET runs on Windows 10 to Windows 11, GNU Linux and OSX 13+.

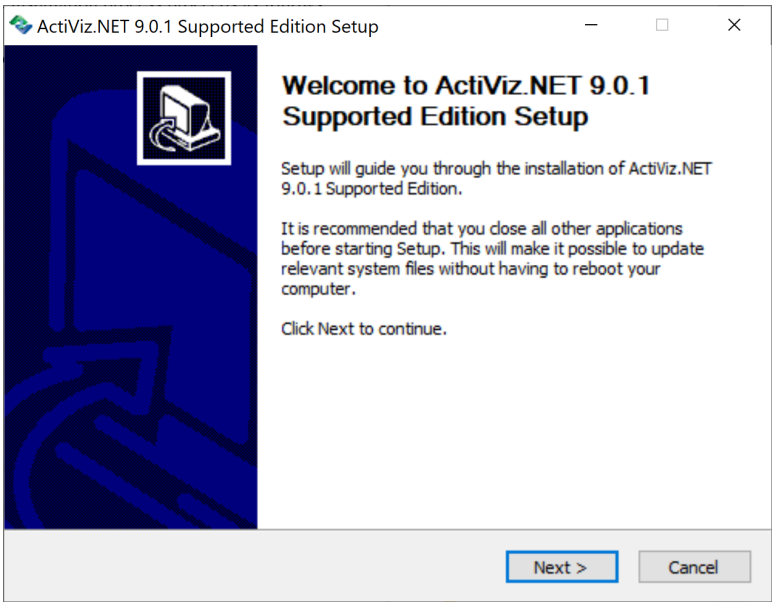
While VTK itself can run on small to large computers, it is a sophisticated, powerful

system that requires adequate computing resources. We advise using hardware with a dedicated graphics card.

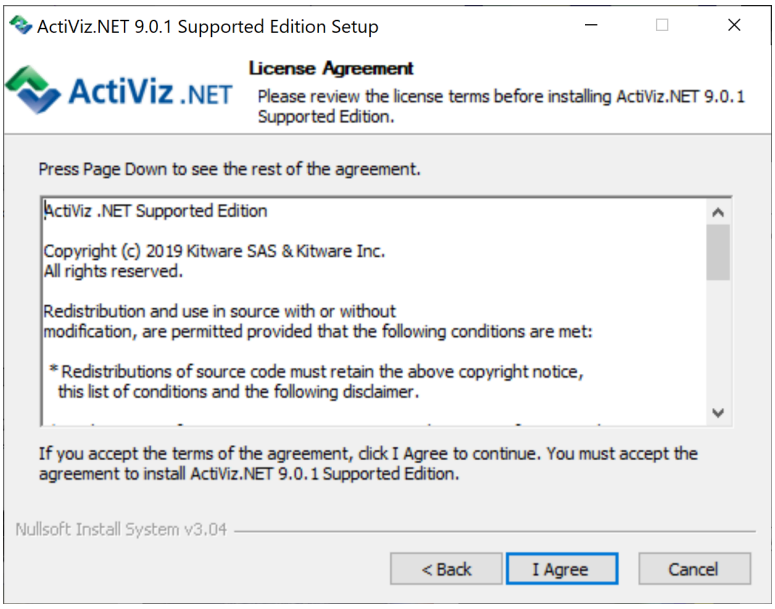
3.2. Installation With Installer (Windows)

Once you have downloaded the appropriate installer, open it and follow the instructions. The installation process proceeds as follows.

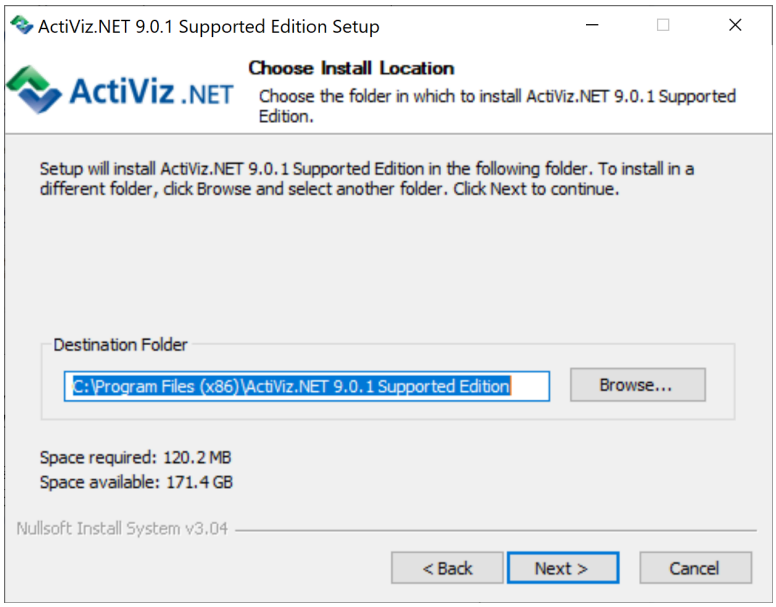
- First, you will see the welcome splash screen. Choose “Next>” and proceed to the next step.



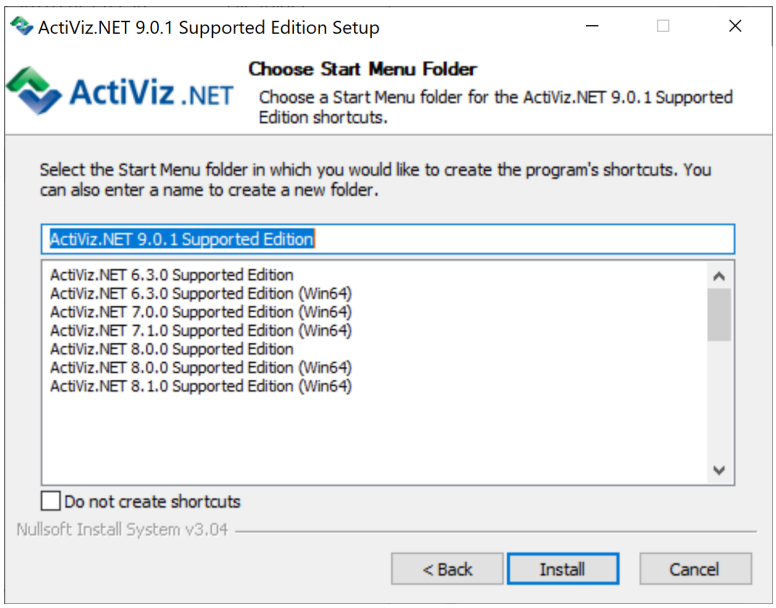
- Next, please review the license agreement and agree to its terms.



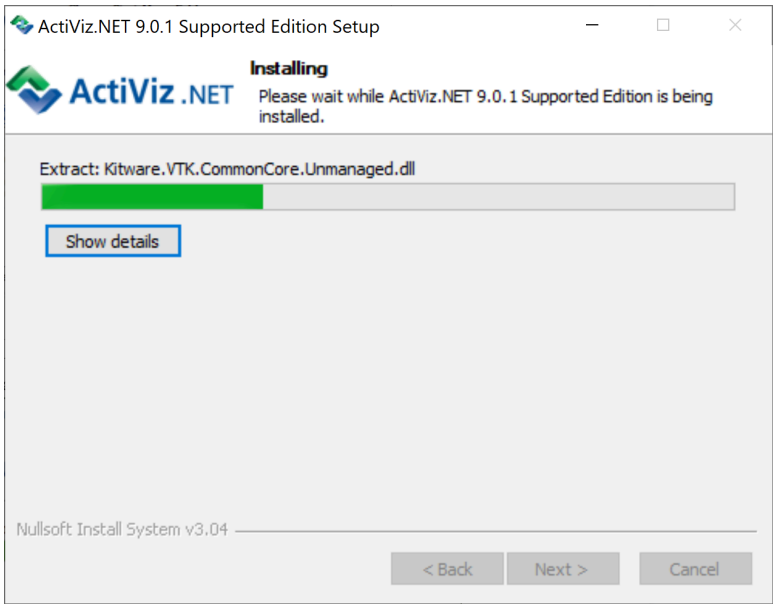
- Indicate where on your computer you would like to install the software. For the purposes of this *User's Guide*, the installation location is in C:\Program Files (x86)\ActiViz.NET 9.3 Supported Edition. (Note that administrative privileges are required to install ActiViz in the "Program Files" directory. However, you should install the software anywhere on the system.) Choose "Next" and proceed with the installation.



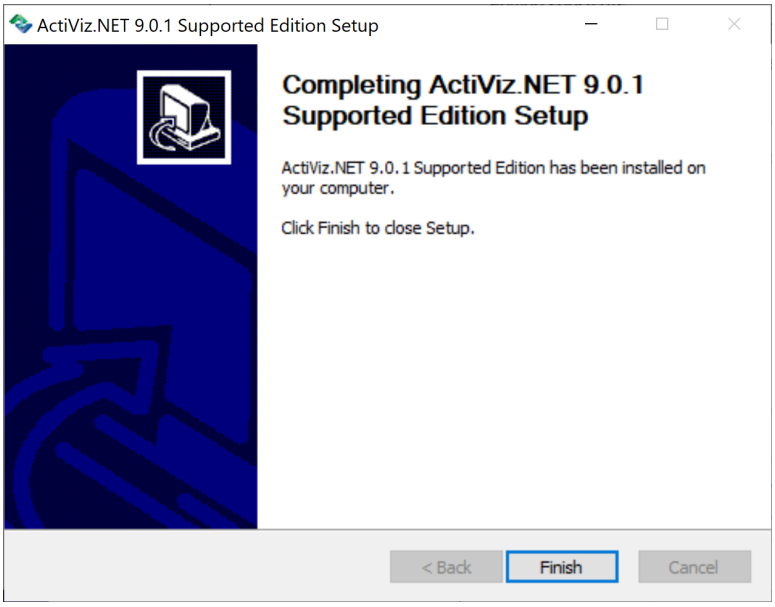
- You will also need to indicate where to place the installation on your Start Menu Folder.



- Once you select “Install” the software installation process begins as illustrated below. It should take less than a minute.



- That’s it, you have successfully installed ActiViz .NET on your computer.

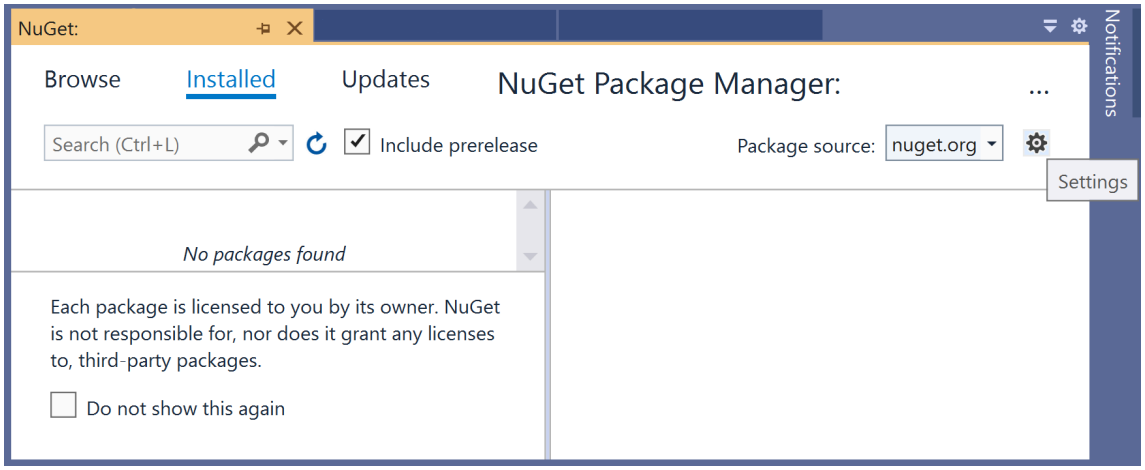


3.3. Installation With NuGet

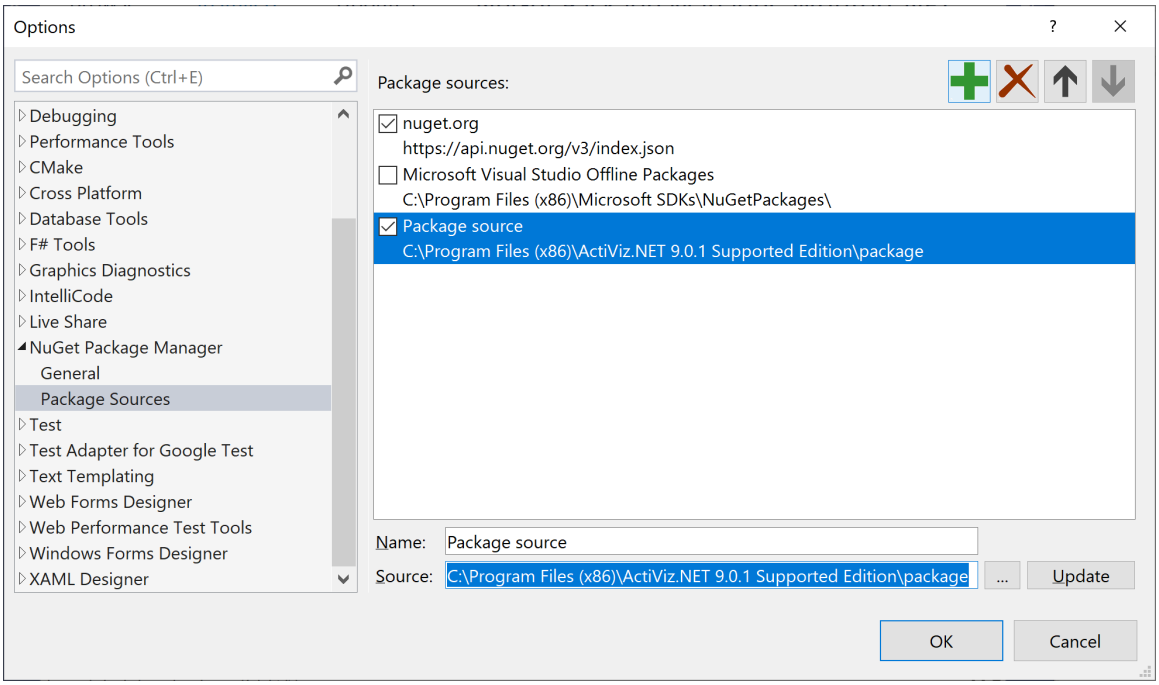
Activiz Nuget packages of the latest Supported Version targeting .NET Core are provided with the installer (Windows) and archives (Linux and Mac).
To install and use Activiz in .NET Core projects, you must provide nuget with the location of the packages that come with the installer. This can be achieved by adding the “package” directory, located in your Activiz installation folder, to the local nuget package feed.
Visual Studio

To do so, first install Activiz with the installer, then proceed with the following steps in Visual Studio:

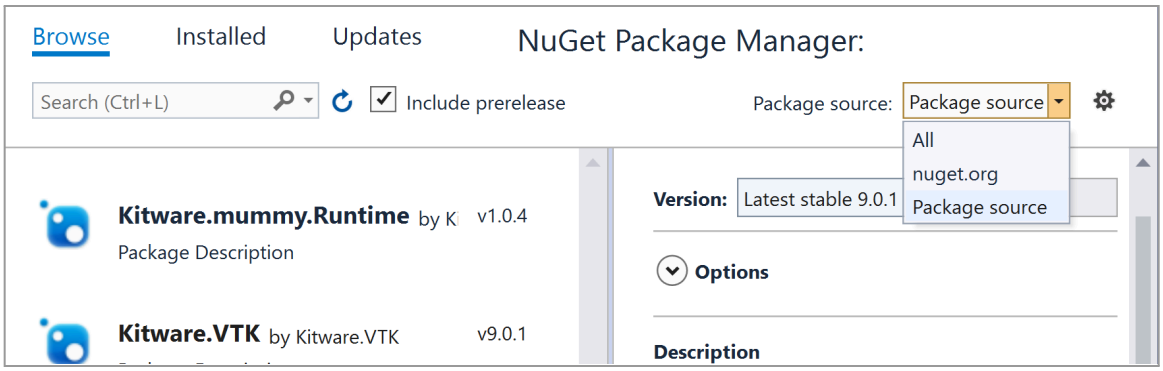
- Right-click on your project
- Select “Manage NuGet Packages”
- Click the Package Source Settings wheel in the upper right corner of the Nuget Package Manager



- Add a new package source using the "+" icon, and update the Source path to include the location of your Activiz nuget packages



- The new package source appears in the list and allows for browsing local packages. You can now install Kitware.VTK and start using Activiz.



Command line

On any system, one can add a package source using the following command:

```
dotnet nuget add source .../ActiViz/package
```

After this has been called, dotnet will be able to automatically find ActiViz NuGet packages when building a project that depends on Kitware.VTK package.

To build and run a project, use the following command from the directory containing the "csproj" file:

```
dotnet run
```

3.4. ActiViz Control in Visual Studio

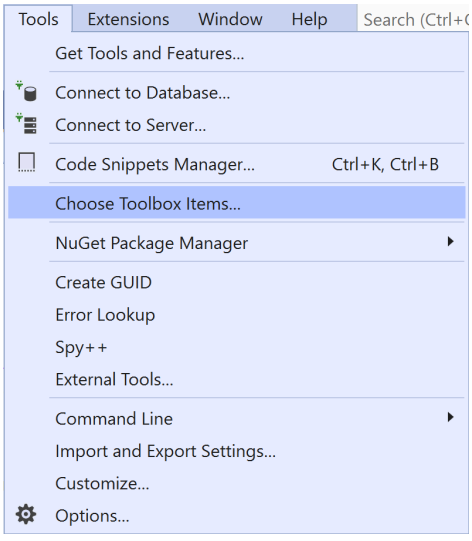
ActiViz .NET provides a user control to integrate VTK advanced rendering into your application GUI.

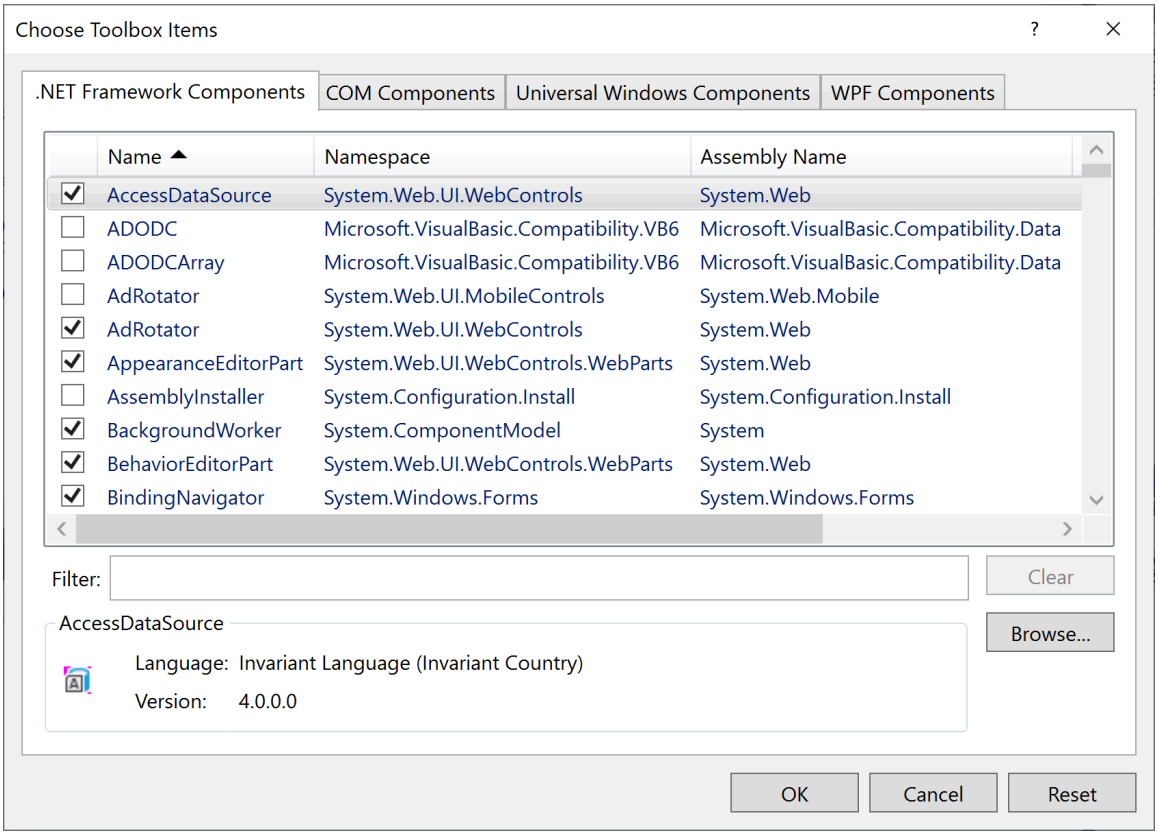
In recent versions of Activiz, the RenderWindowControl provided in the Kitware.VTK assembly should be automatically detected and added to the VisualStudio Designer ToolBox when adding a reference to Activiz in your project. (See next section for more information on how to add references)

The RenderWindowControl can be used when designing applications.

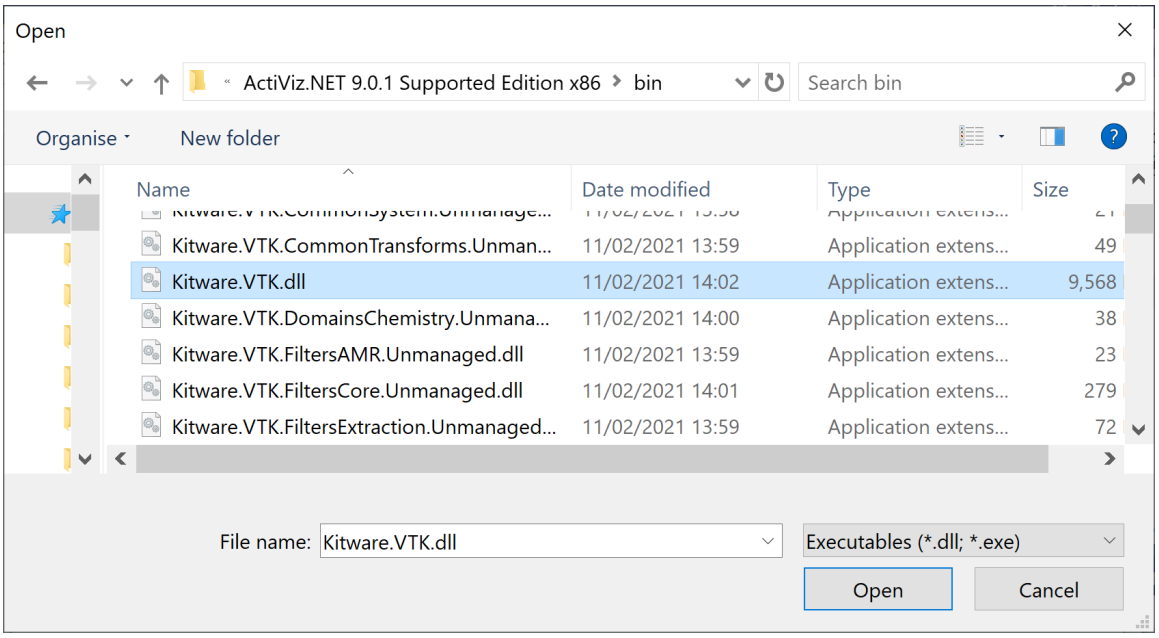
If the RenderWindowControl is not available by default in the Designer ToolBox, you may need to configure Visual Studio as follows.

First, choose "Tools" from the menu bar and select "Choose Toolbox Items" as shown in the figure. A popup will appear.





You will need to browse for .NET Framework Components by selecting "Browse" to search in the ActiViz bin directory (e.g., C:\Program Files\ActiViz.NET 9.5 Supported Edition x64\bin). The following selection will appear:



Select the **Kitware.VTK.dll** assembly. This will make the ActiViz control available in

your toolbox when designing WindowsForm applications.

4. Using ActiViz .NET

4.1. Activiz .NET Core Nuget Package (Recommended)

Starting at version 9.1, nuget packages of ActiViz are provided for .NET Core applications. The following target frameworks are available within the package:

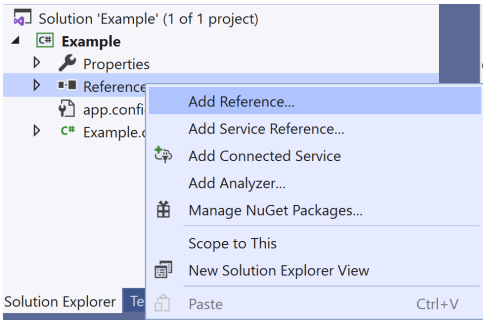
- net9
- net9-windows (For WindowsForm and WPF support)

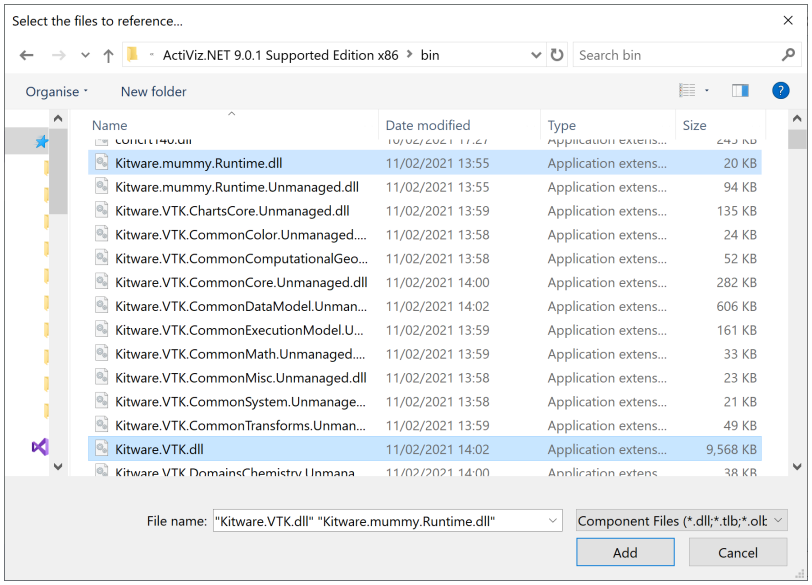
To start using Activiz in .NET Core applications, follow both the installation instructions of sections Installation With Installer and Installation With Nuget.

4.2. ActiViz .NET Framework Reference

The first step to use ActiViz in your project is to add a reference to ActiViz libraries. For this, right click on your project in the Solution Explorer, and choose "Add Reference". This will bring up a dialog, at which point you will "Browse" to the ActiViz .NET install location C:\Program Files\ActiViz.NET 9.5 Supported Edition x64\bin (or appropriate). Then choose the two assemblies as shown in the figure below:

- "Kitware.mummy.Runtime.dll"
- "Kitware.VTK.dll".





4.3. A Ridiculously Brief Overview of VTK

We begin with a brief introduction of VTK. Though VTK is a large and complex software system, knowledge of its conceptual framework will greatly assist you in understanding the following examples.

First, VTK is an object-oriented system. The practical realization of this design is that objects are instantiated and then combined in a variety of patterns to form applications. Each class represents a focused piece of functionality. The instances (or objects) of these classes are manipulated by invoking methods upon them.

Second, VTK is a data-centric toolkit manifesting a data-flow pipeline. The so-called visualization pipeline is created by connecting algorithms (also called process objects) together. Behind the scenes, the algorithms exchange data objects between themselves across the pipeline. For example, a pipeline can be created that reads polygonal data, decimates the data, smooths it, and then passes it on to VTK's rendering subsystem. Practically the purpose of the visualization pipeline is to transform data into rendering primitives which are eventually displayed through VTK's graphics subsystem; although in some cases VTK may just be used as a data processing engine—loading data, processing it, and writing it back to disk.

The graphics subsystem is used to display data of various forms including polygonal data and volumes (i.e., regularly sampled data). The rendering system consists of the following key objects that are combined into a *scene* to produce the final 3D display.

- vtkActor and vtkProp — the objects to be rendered that appear in the scene. In general, we refer to these objects as "actors" although vtkActor is in fact a subclass of vtkProp (like the "props" found on stage).
- vtkCamera — the object used to project the actors from 3D space into a 2D image.
- vtkLight — used to illuminate the scene.
- vtkProperty — used to apply material (i.e., lighting) properties to actors.
- vtkRenderer — this is the object where the rendered image is shown.
- vtkRenderWindow — one or more renderers can be combined into a render window.

This organization of objects is consistent with the "lights, cameras, actors" conceptual model that is familiar to many of us from the movie/video making business. Note that many other objects are present behind the scenes such as transformation matrices (vtkTransform), interactors (process mouse and keyboard events), and texture maps (vtkTexture). Note that when building Form Applications, ActiViz .NET system combines the vtkRenderer and vtkRenderWindow into a single class called the "RenderWindowControl". This is the form that is created in ActiViz applications and embedded into the .NET program.

One important note: the vtkRenderWindowInteractor class is the keystone class for managing mouse and keyboard events in the render window. Through interactor styles (subclasses of vtkInteractorObserver) it is possible to customize the interaction behavior. By default, the interactor supports the following bindings:

- left mouse — rotate camera
- middle mouse — translate camera

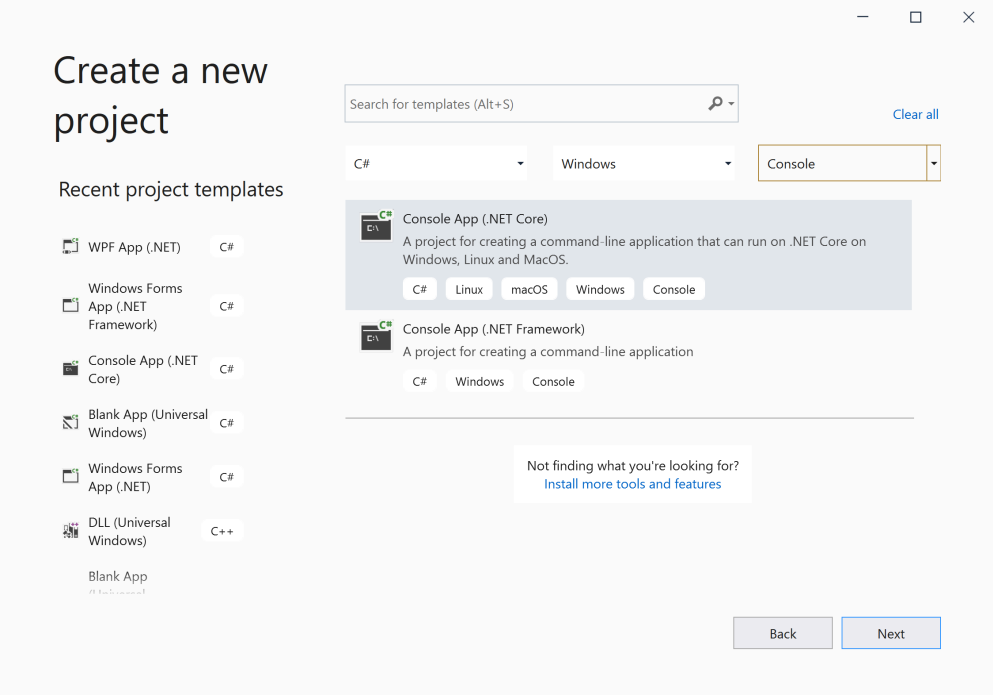
- right mouse — zoom in/out
- keypress j — enter joystick mode (mouse down causes continuous camera motion), exits trackball mode
- keypress t — enter trackball mode (mouse down plus motion causes camera motion), exits joystick mode
- keypress f — press this key when over an actor (a pick is performed behind the scenes); the camera focal point is set to the picked position and the camera flies towards the point.
- keypress w — the actors are shown in wireframe
- keypress s — the actors are shown as surfaces
- keypress r — reset the camera so that all visible actors appear in the scene.

The examples that follow in this section all implement these mouse and keyboard bindings.

Besides the basic classes described here, there are hundreds more classes that implement key functionality for the VTK system. This includes filters for processing data, interaction widgets for direct manipulation of data, image processing, volume rendering, information visualization, mathematics, and computational geometry (to name just a few). The following examples demonstrate a variety of ways in which VTK can be used, and how to use VTK in the .NET framework.

4.4. Hello VTK - Console Application

In the first example we will create a console application. Create a new Visual Studio project and choose the Console application template.



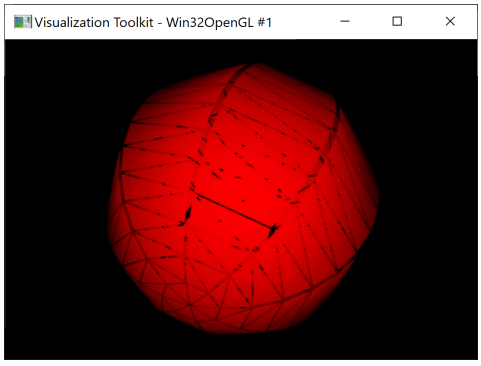
Open the created project and add the required references to ActiViz depending on your .NET version, by following the instruction in Using Activiz .NET.

Now it's simply a matter of adding in the appropriate references and writing some VTK code. In this example we will use C#. (Please read the short introduction to VTK to help clarify some of the concepts.) To add in references, make sure that the line "using Kitware.VTK;" (highlighted below) is added to your application.

```
using System;
using System.Collections.Generic;
using System.Text;
using Kitware.VTK;

namespace HelloVTK

{
    class Program
   {
        static void Main(string[] args)
        {
        // add code here
        }
    }
}
```

Next, insert the appropriate VTK code into the Main() function as exemplified by the following.

```
public static void Main(String[] argv)
{
    // Create a simple sphere. A pipeline is created.
    sphere = vtkSphereSource.New(); sphere.SetThetaResolution(8);
    sphere.SetPhiResolution(16);

    shrink = vtkShrinkPolyData.New();
    shrink.SetInputConnection(sphere.GetOutputPort());
    shrink.SetShrinkFactor(0.9);

    mapper = vtkPolyDataMapper.New();
    mapper.SetInputConnection(shrink.GetOutputPort());

    // The actor links the data pipeline to the rendering subsystem
    actor = vtkActor.New();
    actor.SetMapper(mapper); actor.GetProperty().SetColor(1,0,0);

    // Create components of the rendering subsystem
    //
    ren1 = vtkRenderer.New();
    renWin = vtkRenderWindow.New();
    renWin.AddRenderer(ren1);
    iren = vtkRenderWindowInteractor.New();
    iren.SetRenderWindow(renWin);

    // Add the actors to the renderer, set the window size
    //
    ren1.AddViewProp(actor);
    renWin.SetSize(250,250);
    renWin.Render();
    camera = ren1.GetActiveCamera();
    camera.Zoom(1.5);

    // render the image and start the event loop
    //
    renWin.Render();
    iren.Initialize();
    iren.Start();

    deleteAllVTKObjects();
}

static vtkSphereSource sphere;
static vtkShrinkPolyData shrink;
static vtkPolyDataMapper mapper;
static vtkActor actor;
static vtkRenderer ren1;
static vtkRenderWindow renWin;
static vtkRenderWindowInteractor iren;
static vtkCamera camera;
```



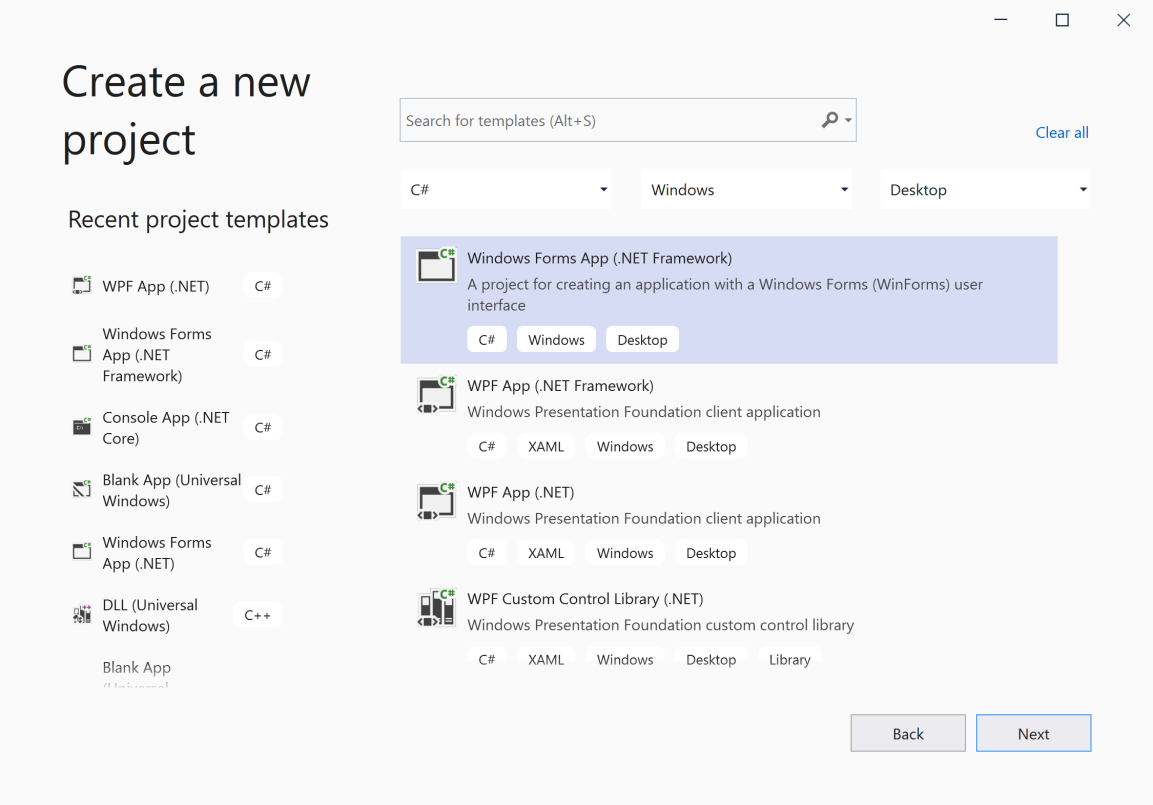
```
///<summary>Deletes all static objects created</summary> public static
void deleteAllVTKObjects()
{
    //clean up vtk objects
    if (sphere != null) { sphere.Dispose(); }
    if (shrink != null) { shrink.Dispose(); }
    if (mapper != null) { mapper.Dispose(); }
    if (actor != null) { actor.Dispose(); }
    if (ren1 != null) { ren1.Dispose(); }
    if (renWin != null) { renWin.Dispose(); }
    if (iren != null) { iren.Dispose(); }
    if (camera != null) { camera.Dispose(); }
}
```

Compiling and running the C# program yields the red sphere above. (Note that some interaction with the camera was performed to move the camera into the position shown.)

In this example, a simple pipeline is implemented that generates some polygonal data (the sphere source); shrinks the polygons towards their center (the shrink filter), and then maps the data (e.g., polygons) to the graphics library. Since no lights and cameras are manually created, they are automatically created. Also, the vtkRenderWindowInteractor is used to control mouse and keyboard events in the window. Note: calling iren.Start() in the example runs a Windows message loop. To exit the message loop, and hence the application, simply close the window.

4.5. *Hello VTK - Windows Form Application*

To create a Windows Form Application (shown here in C#), select “Create a new project” from the Visual Studio start menu. Select the Windows Forms App template for either the .NET Framework or .NET Core, and fill the project information.

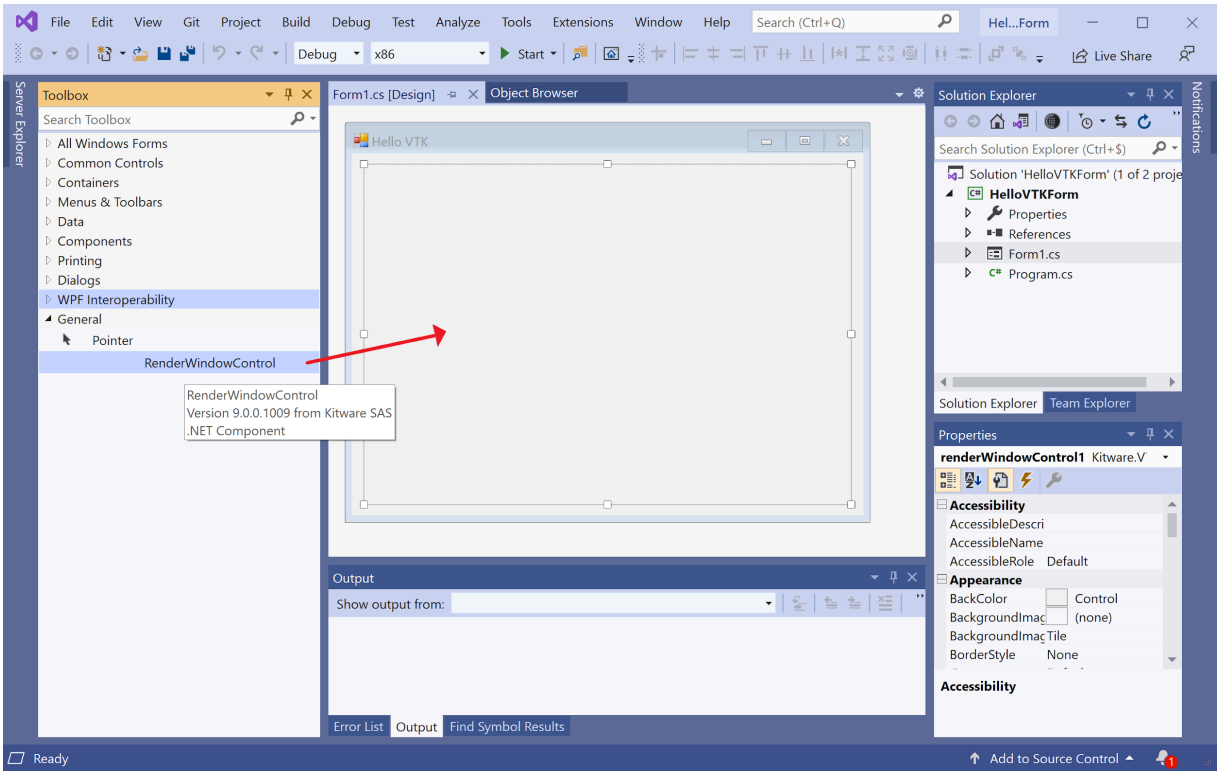


Open the created project and add the required references to ActiViz depending on your .NET version, by following the instruction in Using Activiz .NET.

Next, go to the View Menu and make sure the Toolbox is visible.

Under the General category in the toolbox, the RenderWindowControl will appear. To add the control to your application, select the RenderWindowControl and place it into the form as shown in the figure below.

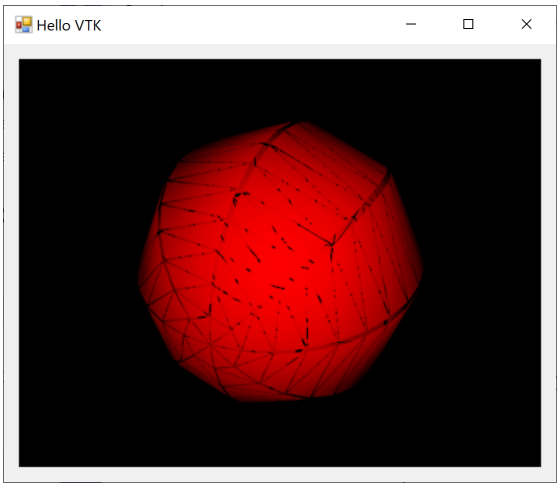
Note: The integration of the ActiViz control in your application can be done programmatically. This can be necessary if the control is not in the ToolBox (see the part ActiViz Control in Visual Studio for more details).



Next, double click on the RenderWindowControl to bring up the C# code:

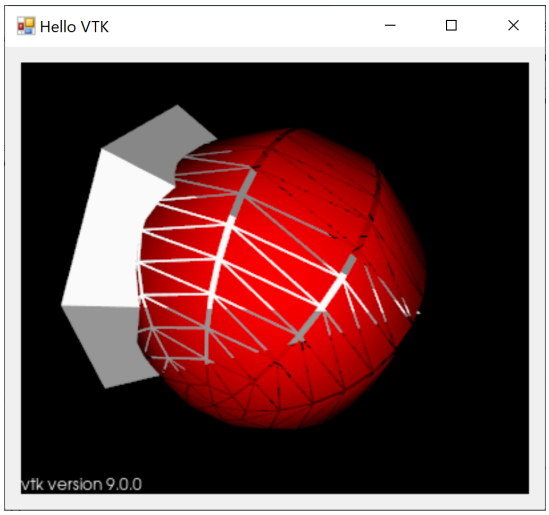
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace HelloVTK
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void
        renderWindowControl1_Load(object sender, EventArgs e)
        {
         //add code here
        }
    }
}
```

As in the Console Application, code can now be inserted into the skeleton application. Make sure that "using Kitware.VTK;" is added, and add references to the solution. Finally, note that the control provides some convenience methods for accessing the renderer and render window, and builds in a render window interactor. Here is what the code looks like, and the image that results from running the application:




```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

using Kitware.VTK;

namespace HelloVTK
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void
        renderWindowControl1_Load(object sender, EventArgs e)
        {
            // Create a simple sphere. A pipeline is created.
            vtkSphereSource sphere = vtkSphereSource.New();
            sphere.SetThetaResolution(8);
            sphere.SetPhiResolution(16);

            vtkShrinkPolyData shrink = vtkShrinkPolyData.New();
            shrink.SetInputConnection(sphere.GetOutputPort());
            shrink.SetShrinkFactor(0.9);

            vtkPolyDataMapper mapper = vtkPolyDataMapper.New();
            mapper.SetInputConnection(shrink.GetOutputPort());

            // Link the data pipeline to the rendering subsystem
            vtkActor actor = vtkActor.New();
            actor.SetMapper(mapper);
            actor.GetProperty().SetColor(1, 0, 0);

            // Create components of the rendering subsystem
            //
            vtkRenderer ren1 = renderWindowControl1.RenderWindow.
                   GetRenderers().GetFirstRenderer();
            vtkRenderWindow renWin = renderWindowControl1.RenderWindow;

            // Add the actors to the renderer, set the window size
            //
```

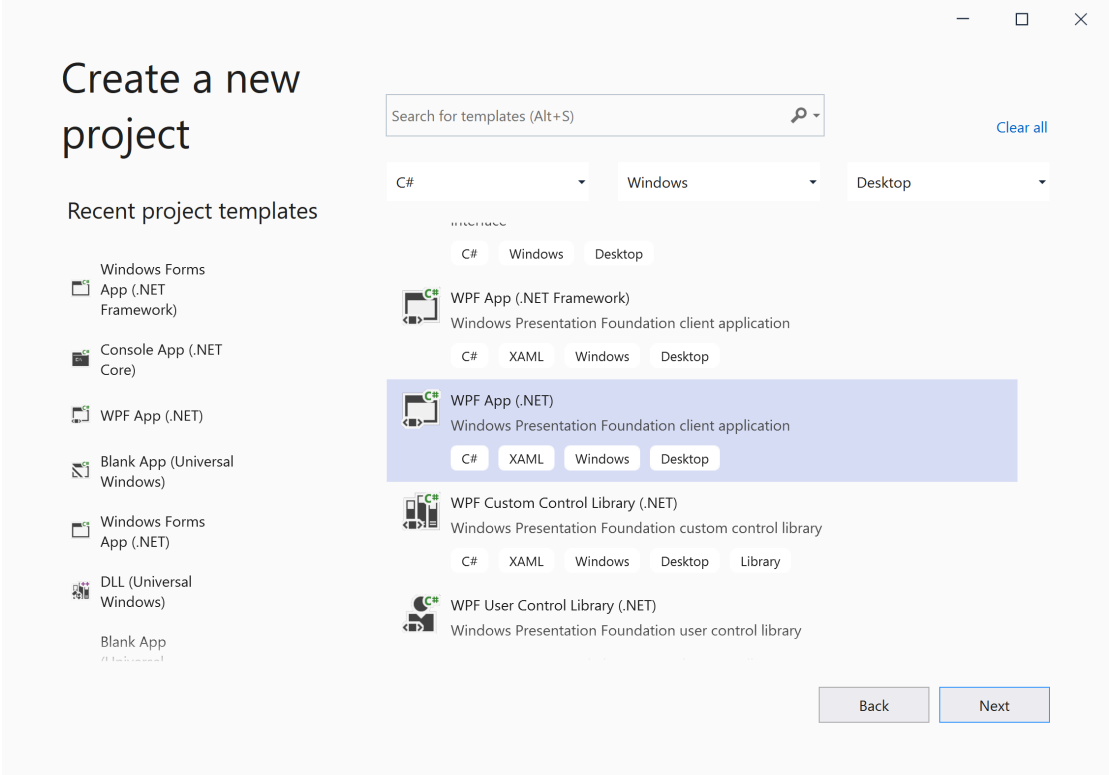
```
            ren1.AddViewProp(actor);
            renWin.SetSize(250,250);
            renWin.Render();
            vtkCamera camera = ren1.GetActiveCamera();
            camera.Zoom((double)1.5);
        }
    }
}
```



Note that the RenderWindowControls has an important property that aid in debugging. If the property AddTestActors is set to "True" then running the application results in an image similar to the one shown to the right. An extra cones is inserted into the VTK scene. This property addresses the frustrating blank image problem that can occur when the VTK pipeline is configured incorrectly, or the application is executing improperly.

4.6. *Hello VTK - WPF Application*

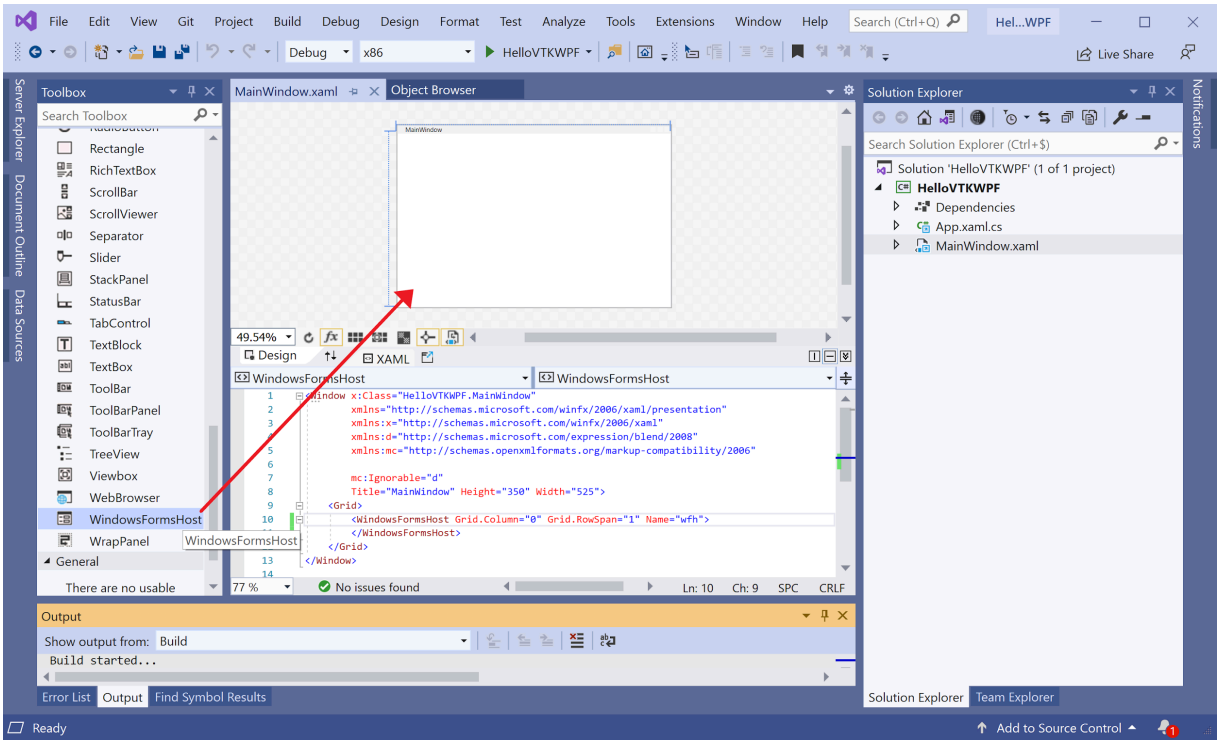
To create a Windows Presentation Foundation application, select “Create a new project” from the Visual Studio start menu. You can use ActiViz in both WPF applications for the .NET Framework and for .NET Core. Select the WPF App template you prefer and choose the project name and location.



ActiViz render control relies on WPF-WindowsForm interop to run in a WPF application. We use the WindowsFormHost component to provide a compatible area for the RenderWindowControl.

A few limitations to this approach mainly prevent the control from being rotated or causes air-space issues when having overlapping elements. This has generally no impact in the application design.

Open the toolbox and add a WindowsFormHost XAML component in your main page:



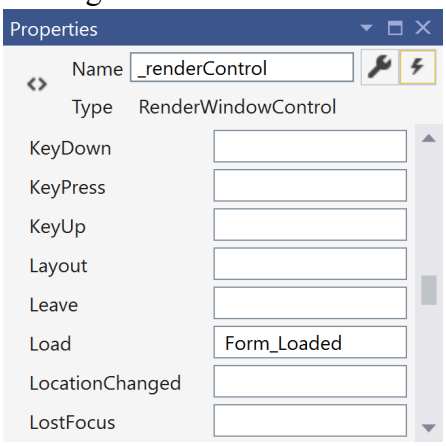
Add the required references to ActiViz depending on your .NET version by following the instruction in Using Activiz .NET.

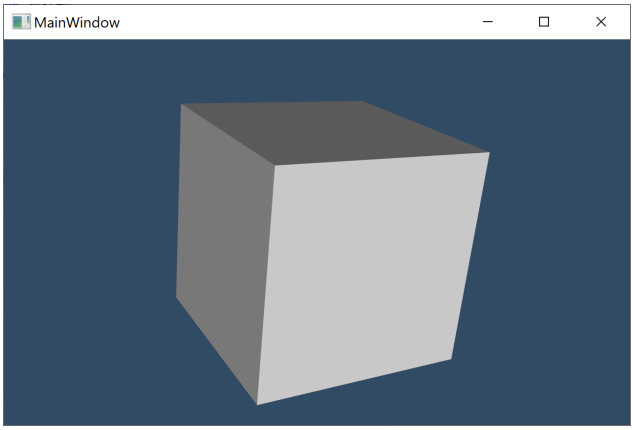
Edit the MainWindow.xaml description to add the ActiViz .NET RenderWindowControl in the WindowsFormHost component. Make sure the namespace containing the control matches the one in the xmlns definitions.

```
<Window x:Class="HelloVTKWPF.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:vtk="clr-namespace:Kitware.VTK;assembly=Kitware.VTK"
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <WindowsFormsHost Grid.Column="0" Grid.RowSpan="1" Name="wfh">
            <vtk:RenderWindowControl x:Name="_renderControl" Load="Form_Loaded" />
        </WindowsFormsHost>
    </Grid>
</Window>
```

An event handler is added to the Load event of the RenderWindowControl. This will be used to initialize the VTK scene.

Properties and Events of the selected element can be easily edited in the Properties window by clicking the xaml line of the element.



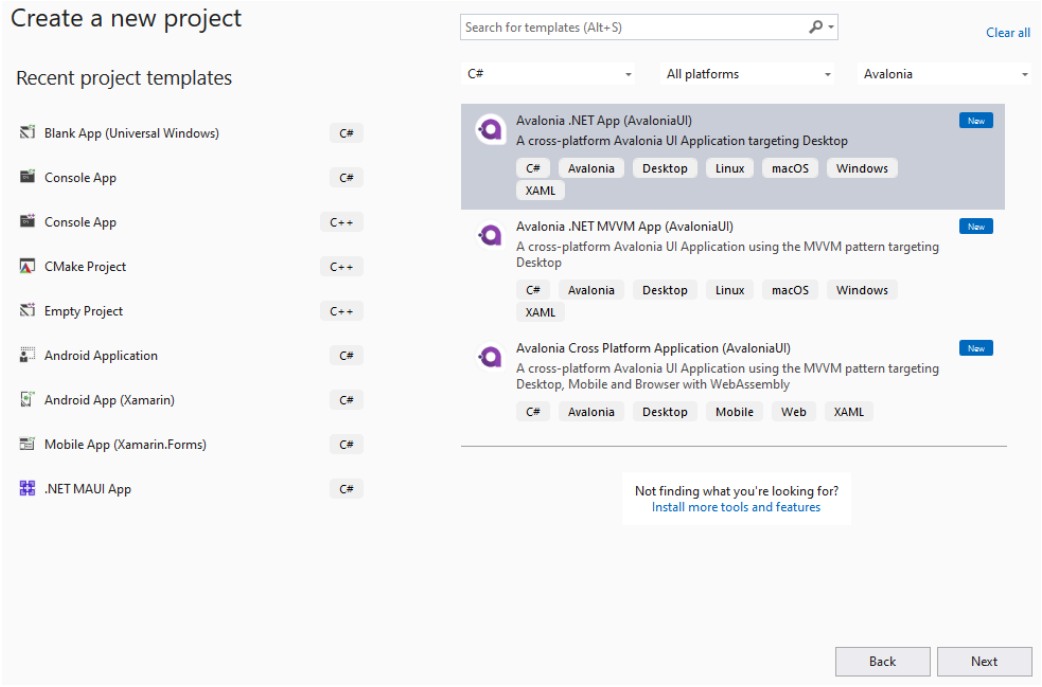


Edit the MainWindow.xaml.cs code to initialize the scene in the event callback we just added:

```
using System;
using System.Windows;
using Kitware.VTK;

namespace HelloVTKWPF
{
  public partial class MainWindow : Window
  {
    public MainWindow()
    {
      InitializeComponent();
    }

    private void Form_Loaded(object sender, EventArgs e)
    {
      // Create a simple cube. A pipeline is created.
      vtkCubeSource cube = vtkCubeSource.New();

      vtkPolyDataMapper mapper = vtkPolyDataMapper.New();
      mapper.SetInputConnection(cube.GetOutputPort());

      // The actor links the data pipeline to the rendering subsystem
      vtkActor actor = vtkActor.New();
      actor.SetMapper(mapper);

      // Create components of the rendering subsystem
      vtkRenderer renderer =
_renderControl.RenderWindow.GetRenderers().GetFirstRenderer();
      renderer.SetBackground(.2, .3, .4);

      // Add the actors to the renderer
      renderer.AddActor(actor);
    }
  }
}
```

4.7. *Hello VTK - Avalonia Application*

To create an Avalonia application, install the Avalonia templates using the following command: `dotnet new install Avalonia.Templates`.
Then, select "Create a new project" from the Visual Studio start menu. Select the Avalonia App template you prefer and choose the project name and location.



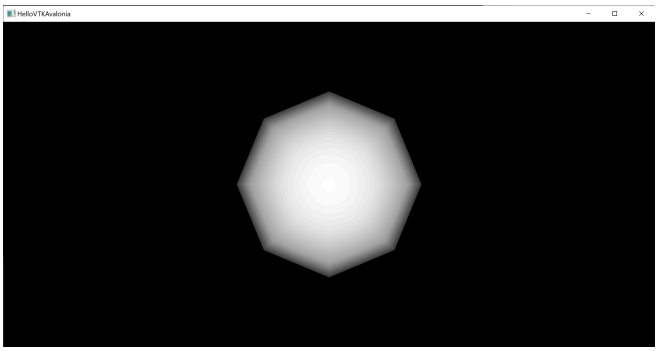
Once the solution is loaded, add the required Kitware.AvaloniaControls nuget package to the list of dependencies by following the instruction in Using Activiz .NET Core Nuget Package. The Kitware.VTK and Kitware.mummy.Runtime packages are automatically added to the project as a dependency of Kitware.AvaloniaControls.

To add the ActiViz RenderWindowControl to the application window, open the MainWindow.axaml file, and edit the xml to define the vtk namespace and add the control to the window as highlighted below:

MainWindow.axaml

```
1  <Window xmlns="https://github.com/avaloniaui"
2          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
3          xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
4          xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
5          xmlns:vtk="clr-namespace:Kitware.AvaloniaControls;assembly=Kitware.AvaloniaControls"
6          xmlns:ctrl-pages="clr-namespace:ControlCatalog.Pages"
7          mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="450"
8          x:Class="HelloVTKAvalonia.MainWindow"
9          Title="HelloVTKAvalonia">
10
11   <vtk:RenderWindowControl Name="VTKControl"/>
12
13 </Window>
14
```

An event handler is then added to the AttachedToVisualTree event of the RenderWindowControl. This will be used to initialize the VTK scene.
Edit the MainWindow.xaml.cs code as follows to add the AttachedToVisualTree event handler and initialize the scene in the corresponding callback:




```
using Avalonia.Controls;
using Avalonia.Markup.Xaml;
using Kitware.AvaloniaControls;
using Kitware.VTK;
using System;

namespace HelloVTKAvalonia
{
  public partial class MainWindow : Window
  {
    public MainWindow()
    {
      InitializeComponent();
    }

    private void InitializeComponent()
    {
      AvaloniaXamlLoader.Load(this);

      RenderWindowControl ctrl = this.FindControl<RenderWindowControl>("VTKControl");
      if (ctrl != null)
      {
        ctrl.AttachedToVisualTree += InitVTKScene;
      }
    }

    public void InitVTKScene(object? sender, EventArgs args)
    {
      RenderWindowControl? mainView = sender as RenderWindowControl;

      vtkRenderer renderer = vtkRenderer.New();
      renderer.SetBackgroundAlpha(1.0);

      mainView.RenderWindow.AddRenderer(renderer);

      vtkInteractorStyleTrackballCamera interactorStyle = vtkInteractorStyleTrackballCamera.New();
      renderWindow.GetInteractor().SetInteractorStyle(interactorStyle);

      vtkSphereSource src = vtkSphereSource.New();
      vtkPolyDataMapper mapper = vtkPolyDataMapper.New();
      mapper.SetInputConnection(src.GetOutputPort());
      vtkActor actor = vtkActor.New();
      actor.SetMapper(mapper);
      renderer.AddActor(actor);
    }
  }
}
```

4.8. VTK events and observers

Activiz supports VTK objects events observers, but in an higher-level form based on C# delegates and events.
vtkCommand and vtkCallbackCommand must not be used in ActiViz, they won't work.
Instead, ActiViz provides the following API:

```
object.<EventName>Evt += callback; // add observer
object.<EventName>Evt -= callback; // remove observer
```

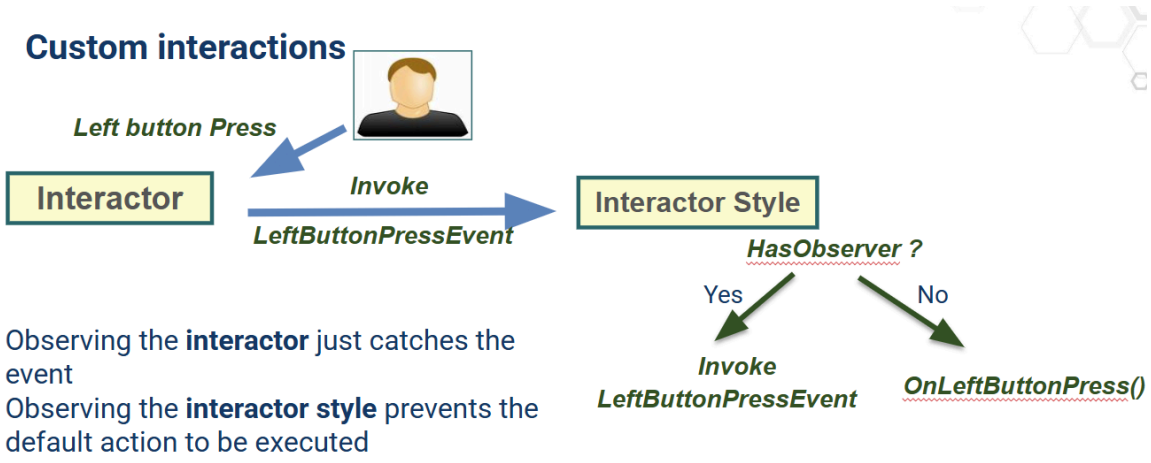
EventName is one of vtkCommand event names, the "Event" at the end is replaced with "Evt" to prevent name conflict, so if you want to hook on the "ModifiedEvent", use the "ModifiedEvt" delegate.

The signature of all delegates is `void(vtkObject sender, vtkObjectEventArgs e)`.
`sender` is the object on which you have added the delegate, you may use `vtk<Type>.SafeDownCast(sender)` to get the object back from the delegate directly. This is especially useful if you connect the same function to multiple delegates.
`e` can be used to retrieve the optional user data sent by some events. Please refer to vtkCommand documentation for more information.

It is possible to use lambda functions as observer:

```
object.ModifiedEvt += (sender, e) => { ... };
```

To receive keyboard and mouse events information, you may use observers on a `vtkRenderWindowInteractor` or on a `vtkInteractorStyle`.
Adding an observer to a `vtkRenderWindowInteractor` enables passive event watching, meaning that you will receive the event but won't block it.
Adding an observer to a `vtkInteractorStyle` will override the interactor style default handler for the given interaction. If you want to do something before the default behavior of the interactor style, you can add an observer on it, then call `sender.On<Event>` to let default behavior flow.

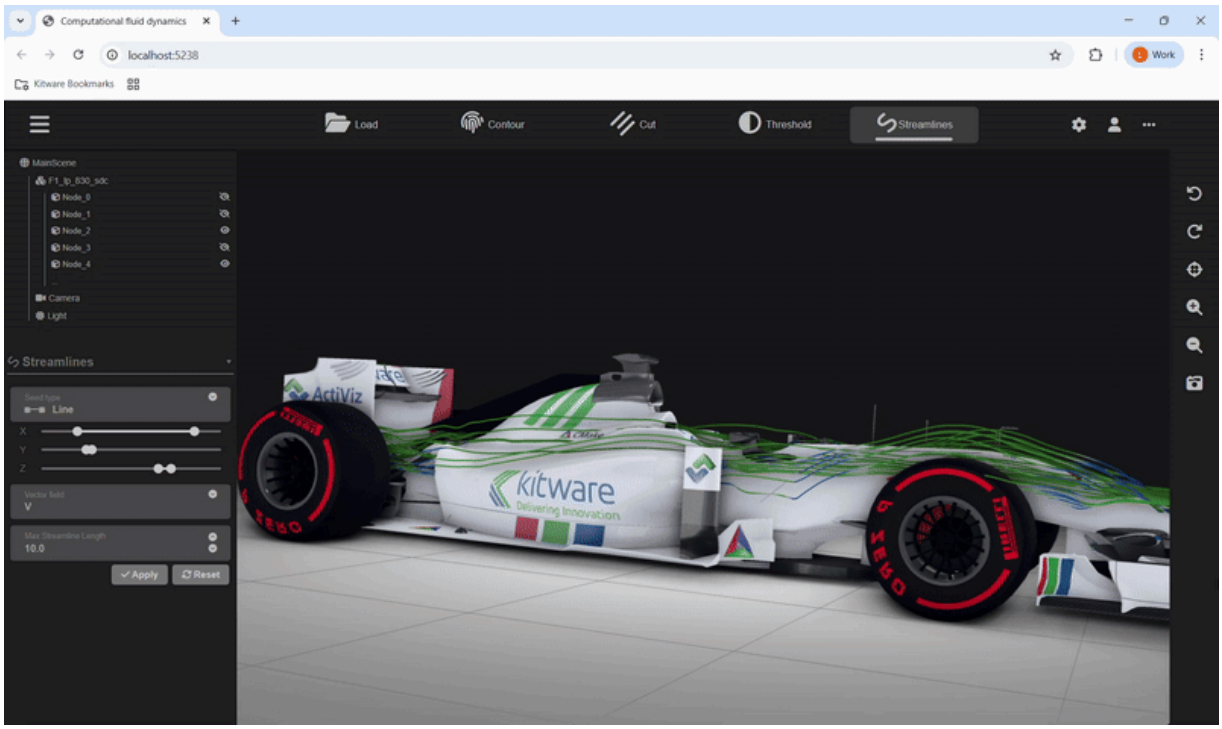


Example:

```CSharp
var interactor = vtkRenderWindowInteractor.New();
interactor.MouseMoveEvt += (vtkObject sender, vtkObjectEventArgs e) => {
   Console.WriteLine("Mouse move"); // Do something meaningful
};

var style = vtkInteractorStyleTrackballCamera.New();
interactor.SetInteractorStyle(style);
style.CharEvt += (vtkObject sender, vtkObjectEventArgs e) => {
    var style = vtkInteractorStyle.SafeDownCast(sender)
                ?? throw new Exception("Wrong type");

    if(style.GetInteractor().GetKeySym() == "p") {
        Console.WriteLine("P has been pressed!"); // Do something meaningful
    } else {
        style.OnChar(); // keep default behavior for other cases
    }
};
```

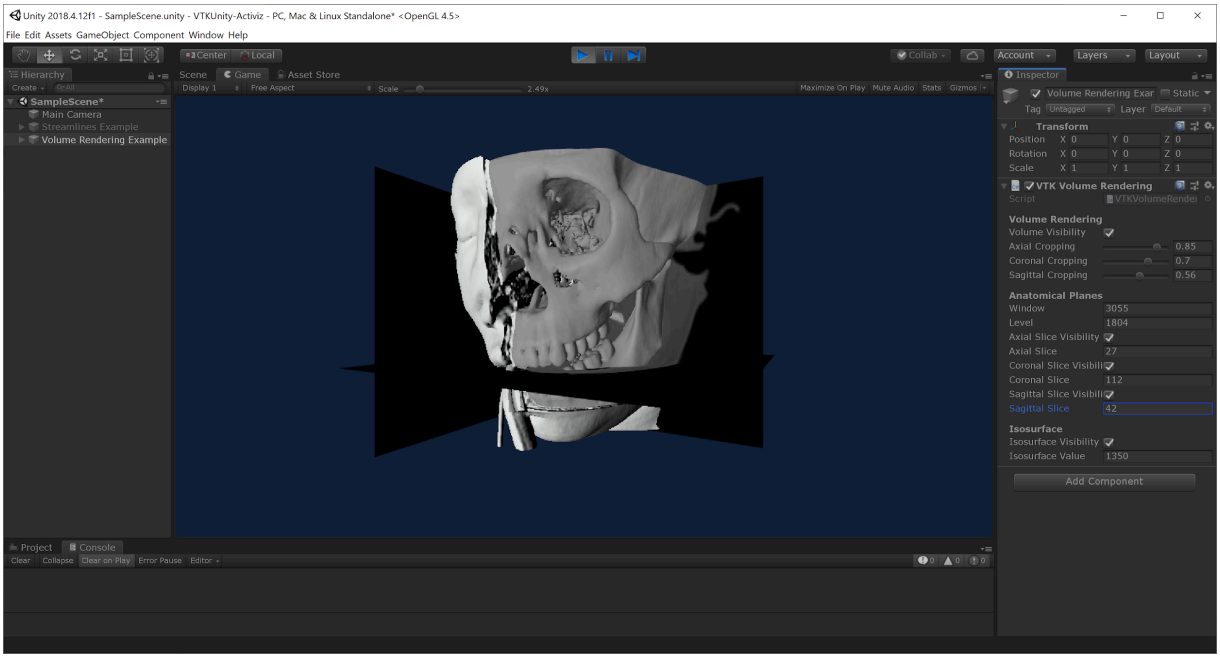
5. ActiViz on the Web

Starting with Activiz 9.5, it is now possible to use ActiViz .NET in your web browser, thanks to VTK WASM support, mono-wasm and Emscripten project.
To start a WASM project, please take a look at the HelloVTK-WASM example `README.md` file.



6. ActiViz in Unity

ActiViz enables the integration of the high-performance algorithms, data representation, scalability, and visualization techniques offered by VTK into the high quality and convenient environment that Unity provides.
Check out Kitware's blog for more information or contact us to request a trial version.



7. Examples

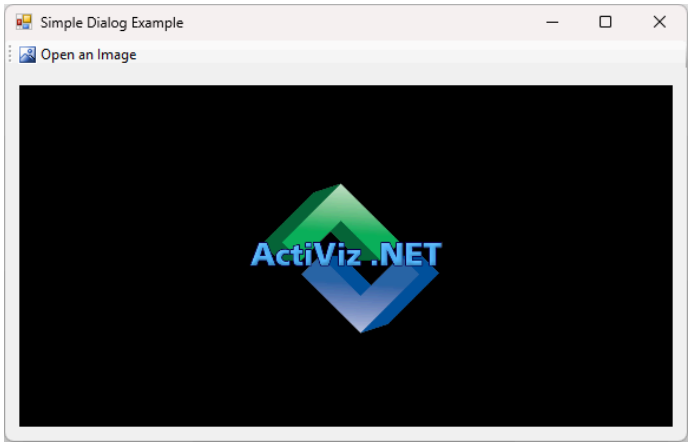
In the following examples, we dispense with the details of the code and project creation. Rather we provide an eclectic mix of examples demonstrating some of the power of VTK, including highlighting some code snippets that are relevant to key functionality. The complete code, in C# and Visual Basic, is available in the installation directory under the Examples subdirectory.

Important Notes: These examples were written to be simple, clear demonstrations of the potential of ActiViz .NET. In general less than an hour was spent writing these examples, and even the Wikipedia browser was completed in well under a day. Thus Kitware does not claim that these are bulletproof applications meant for industrial application. Further, there are some specific limitations of which you should be aware:

- If you install the software in the "Program Files" directory (or other privileged location), then you will have to build the software with admin privileges, or preferably, copy the examples to a non-privileged location and fix the appropriate reference paths.
- The two pre-compiled applications, the Wikipedia Browser and the File Browser, may experience problems when requests lead to processing large amounts of data. Please refer to the specific examples for further clarification.

Despite these caveats, VTK is commonly used in applications requiring robust, high-performance code. However, this requires extra programming safeguards omitted from these examples for the sake of clarity.

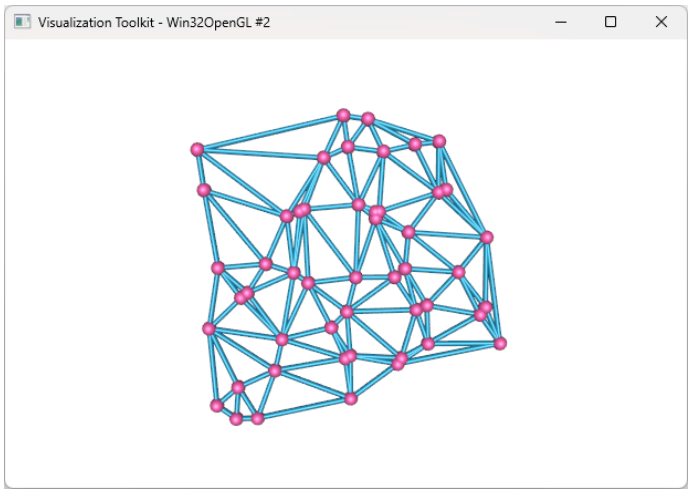
7.1. Load Image Files Dialog



This example demonstrates a simple dialog to open and display an image. It supports several file formats including .png, .jpg, .jpeg, .tif, .slc, .dicom, .minc, .bmp, and .pmn. It will also support .vtk files, which supports data types including polygonal data, structured grids, and unstructured grids.

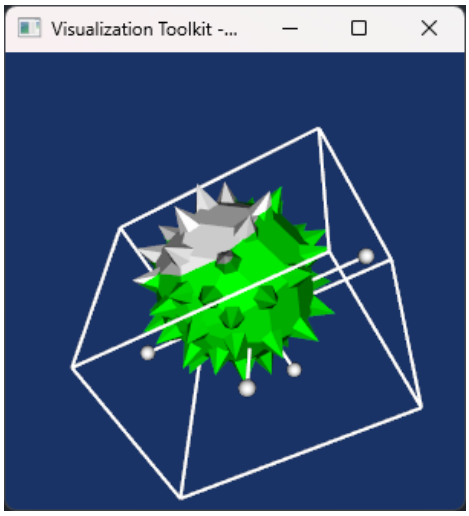
This image viewer is unusual in that it places the image on a quadrilateral polygon using texture mapping. The polygon exists in 3D space so it is possible to move the camera around the polygon using the render window interactor. The "Simple Dialog Example" illustrates what the application looks like after loading a simple image file.

7.2. Delaunay Triangulation



The Delaunay triangulation is a construct in computational geometry used to generate triangulations (i.e., polygonal meshes where the polygons are all triangles). It has many useful functions including interpolating data. In this example, a random set of points in a 2D plane is generated. Next, VTK filters are used to place tubes around the edges and ball glyphs at the mesh vertices. You may wish to play with the number of points generated, as well as the appearance of the graph by modifying the underlying C# source code.

7.3. Box Widget



VTK has an extensive set of widgets. Widgets are objects that appear in the scene but may be directly interacted with. Widgets are analogous to 2D GUI devices such as buttons and sliders, except in 3D they can take on much more complex forms. The figure to the right shows one very general widget called the box widget.

The box widget has six faces that can be selected and then rotated. Each face can be translated separately to modify the extent of the box. The box can also be uniformly scaled and translated. The widget can be queried to return useful information such as the current transformation matrix, and the six planes that form the box. In VTK, this information can be used by the data processing pipeline to perform additional operations. For example in the figure shown, the six planes are used to clip the mace. The surface of the mace outside of the box is grey, the surface inside is green. Also, when you run the example, keypress—i is used to enabled/disable the widget (it will appear and disappear from the scene).

One of the important features of this example is the use of events and associated callbacks to couple the widget with other VTK objects. In this example, we define a callback function as follows:

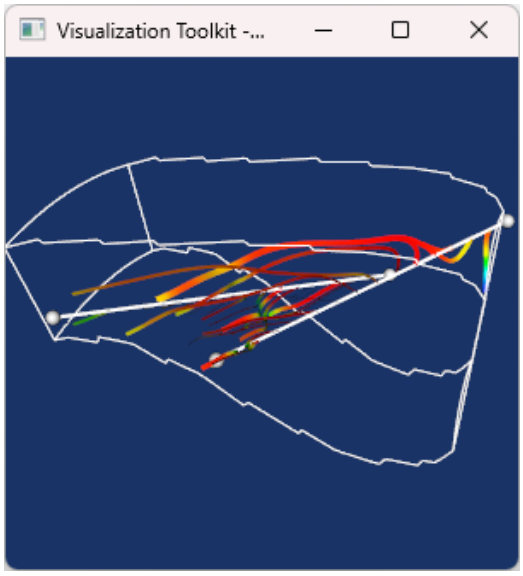
```
public static void
    SelectPolygons(vtkObject sender, vtkObjectEventArgs e)
{
    boxWidget.GetPlanes((vtkPlanes)planes);
    selectActor.VisibilityOn();
}
```

Here the “planes” object is a collection of the six planes retrieved from the widget. In turn, the planes define a clip function to the vtkClipPolyData filter. Next, the callback function is connected to the box widget by observing the end interaction event. (Typically widgets provide three events: start interaction, interaction, and end interaction.)

```
boxWidget.EndInteractionEvt +=
     new vtkObject.vtkObjectEventHandler(SelectPolygons);
```

Thus, when the widget is manipulated, the event is triggered (at the end of motion) and the callback function is invoked. Thus widgets are very easy to add and to use in ActiViz.

7.4. Streamline Generation



This example shows some of the visualization capabilities of VTK. A structured grid (think of a volume warped in 3D space so that it is topologically regular but geometrically distorted) is read with associated scalar and vector data. This 3D dataset is the result of a computational simulation of combustion in a segment of an annular combustor from an aircraft engine.

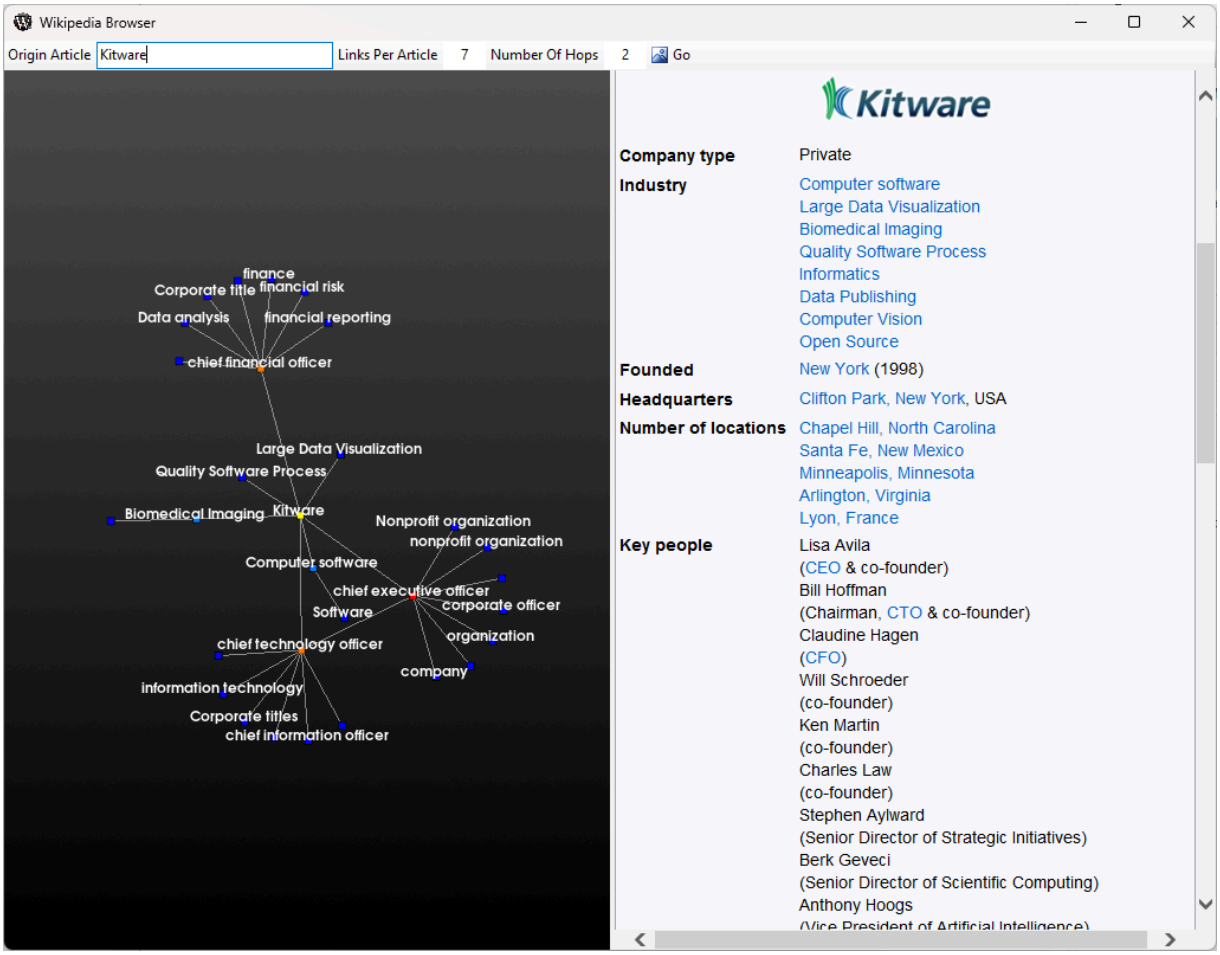
To visualize the data, streamlines are generated. These synthetic streamlines are similar to smoke traces used in wind tunnels, and represent the path that a massless particle would take in a vector field (here the vector field is the flow momentum). The streamlines are modified by using a VTK filter to place a ribbon on the streamline. Since streamlines must have a starting point, a line widget is used to seed the streamlines.

Once you compile and run the example, aside from using the interactor to position the camera, it is possible to move the line widget. Do this by selecting the widget end points (marked as little balls), or grabbing the line and translating it around. It is also possible to modify the code to increase the line resolution in order to generate more streamlines emanating from it.

7.5. *Wikipedia Browser*

The general idea of this application is to enable browsing of Wikipedia (the free encyclopedia wikipedia.org) pages. The Wikipedia Browser example uses VTK's information visualization classes to show relationships between Wikipedia entries as a graph (or network). You can interact with this graph to see how articles relate to one another.

The application consists of two panels: on the left a graph browser indicating the relationship of Wikipedia articles to one another; on the right, the Wikipedia web page corresponding to the recently selected graph node (see Figure below).



The application enables the user to type in a search string as a start point for the browser (make sure to select the "Go" button to initiate the search). In the figure below, the initial search string is "VTK". Given this initial string, the search expands out into *N* links where *N* is specified by the user (here the default value is 10). The first *N* links in the initial string are followed to other Wikipedia pages, and this process continues *H* times, where *H* is the user specified number of hops to follow. Note that *N* and *H* must be carefully specified. If these numbers are big enough, the graph may grow rapidly (it is

possible to crash the application as currently written in *N* and *H* are too large). Further, since the nodes represent a web page, each page must be accessed over the internet, and then parsed, which can be very slow depending on network performance.

Once the initial graph is specified, it is possible to expand it iteratively. Simply use the left mouse button to select a rectangular selection region in the left panel. Any nodes in the selection region are further expanded in the graph. Note that when expanded, VTK's graph layout algorithm will run and can reposition the existing graph nodes. You may also select a single node by left-mouse clicking on the node. If you select a single node, its corresponding Wikipedia page is shown in the right panel. If you select a group of nodes with a rectangular selection, then an arbitrary selection from the group is made and the corresponding Wikipedia page is shown in the right panel.

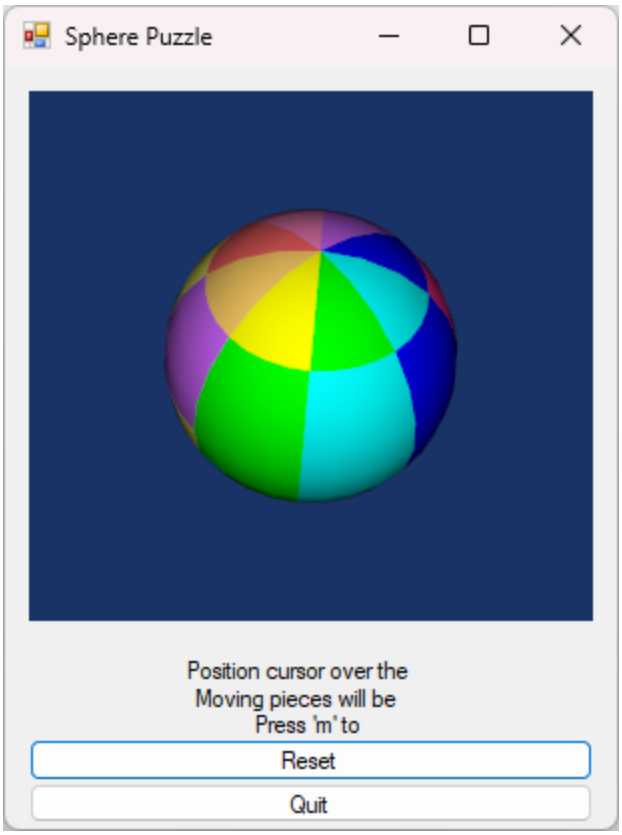
The graph layout has some features that facilitate navigation. While the left mouse button is used for selection, the middle mouse button can be used to translate the graph, and the right mouse button to zoom in and out (move the mouse "up" to zoom, and "down" to zoom out). Zooming in and out causes text to appear and disappear dynamically. Finally, if your mouse has a scrolling wheel, scrolling the wheel also zooms in and out on the graph.

The source code for this application is included in the ActiViz .NET examples directory; feel free to extend it. Other VTK classes exist to improve the behavior of this application, including ways to adjust graph layout, control mouse bindings, change the selection process, and populate the scene.

7.6. *Sphere Puzzle*

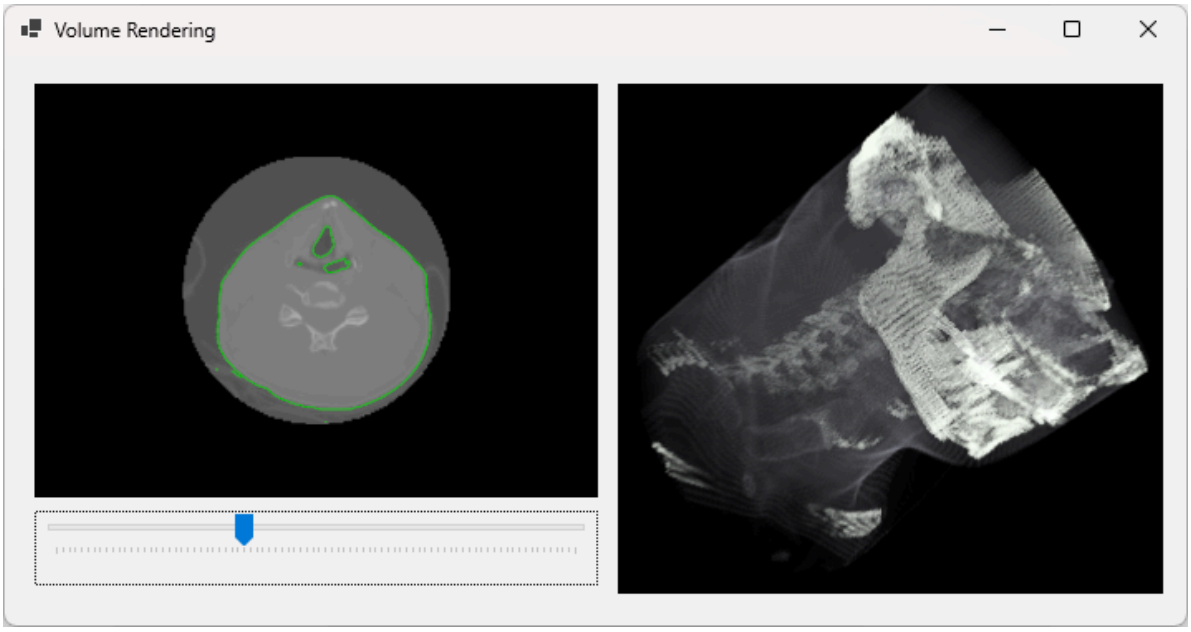
This is a cute example much like a Rubik's cube puzzle except on a sphere rather than a cube. After the application initializes itself by randomizing the panels on the sphere, the user attempts to restore the sphere to the proper coloring (to see the final coloring, hit the "Reset" button).

Moving the sphere panels involves rotating the sphere either in the longitudinal or latitudinal directions. By moving the mouse pointer close to a latitude or longitude line, a portion of the sphere will light up, indicating the portion that will rotate. By hitting the "m" key, the highlighted portion of the sphere will rotate. Repeat this process until you return the sphere to the proper coloring.



7.7. Volume Rendering

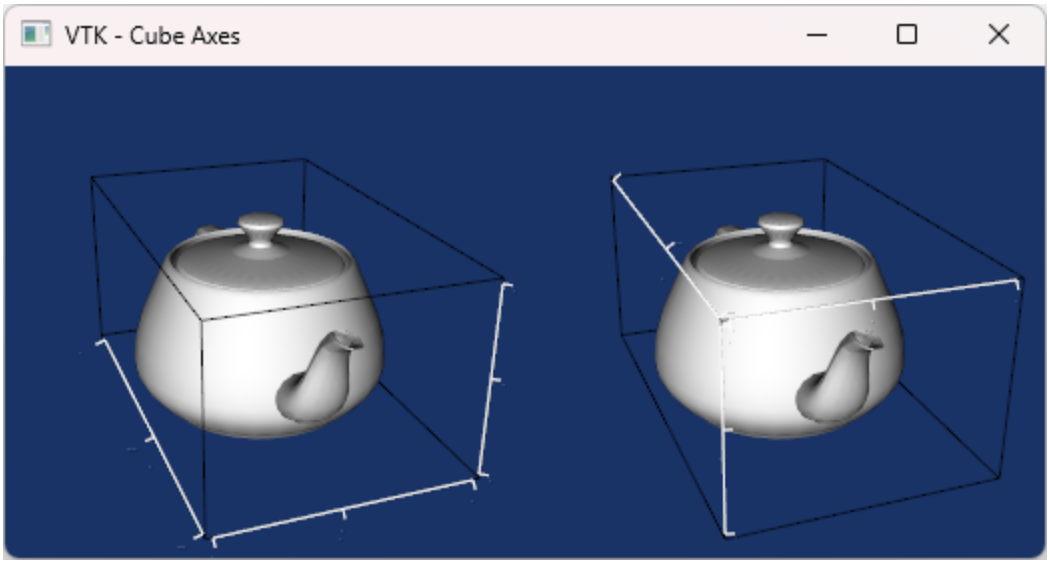
Volume rendering is a visualization technique for rendering regularly sampled 3D data (think of a stack of 2D images or slices, with a uniform vertical spacing between slices). While there are many ways of implementing volume rendering, probably the simplest way to think about it as a ray casting algorithm. Imagine that the volume varies in transparency, where part of the volume may be fully transparent, and part opaque. Further the volume may vary in color as well. A transfer function controls the transparency and color; the transfer function typically maps the volume data value (e.g., intensity) into a color and transparency value. Then to volume render, rays are cast from the camera through each pixel in the renderer. Some may pass into and then traverse the volume, while other rays may miss the volume entirely. For those rays that intersect the volume, sample points are selected, in order, along the ray and mapped through the transfer function based on the data values at each sample point. The color and transparency is accumulated until the ray exits the volume, or the pixel becomes fully opaque. In the past, volume rendering was a compute intensive, typically slow process. However, with modern CPUs and GPUs, volume rendering performance realizes interactive frame rates. VTK implements several different volume rendering strategies as implemented by vtkVolumeMapper and subclasses.



In this example, two render windows are used. One window is connected to a slider and shows the volume slices. The second window shows a volume-rendered image. The second window supports the standard interactor bindings. On modern computers, the application will be fully interactive. (Note: careful examination of the volume rendering window shows that level-of-detail (LOD) is being used while interacting with the data. Once the mouse is released, the rendering reverts to full resolution. LOD strategies are common in computer graphics and VTK supports this in a variety of ways. See vtkLODActor, for example.)

7.8. Cube Axes Actor

VTK provides many classes for annotating data. In the following example, two renderers are placed in a single render window, and a vtkCubeAxesActor is used to annotate the data as shown in the figure below. The two instances of vtkCubeAxesActor are configured differently so that one always emanates from the corner of the object's bounding box closest to the camera, and the other follows the closest edges of the bounding box. Also, a camera is shared between the two renderers so that the two views are synchronized with each other.

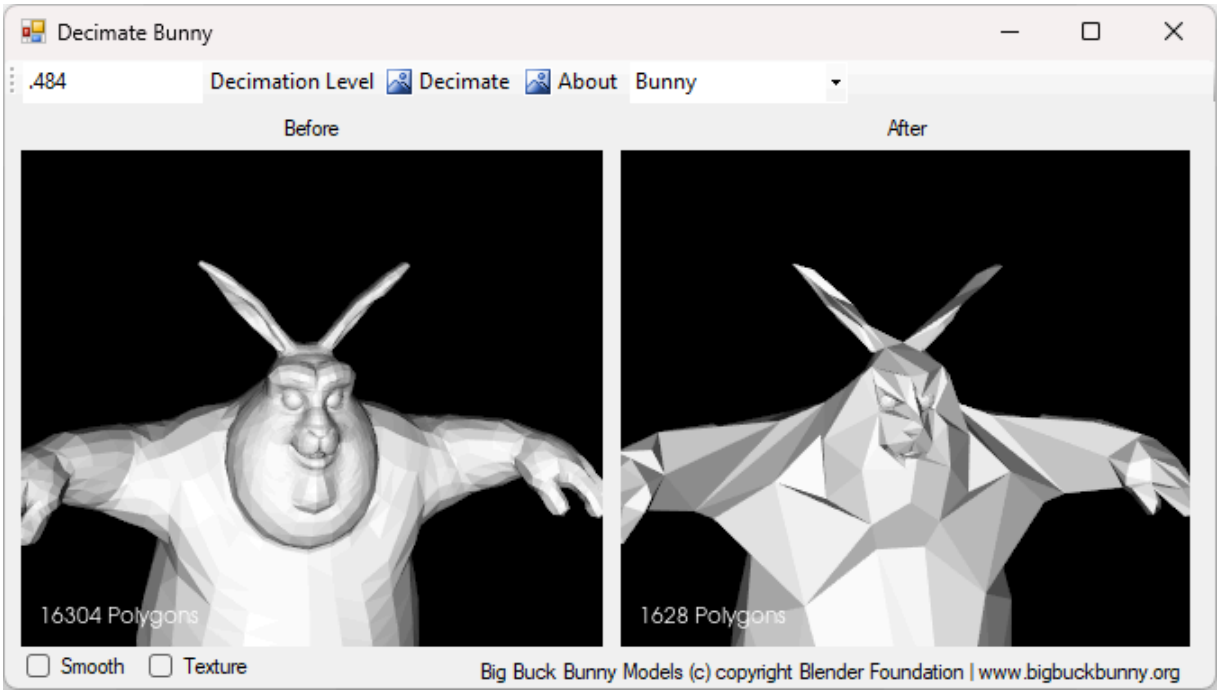


Other types of annotation in VTK include text (both 2D, residing on top of the 3D geometry, or 3D, embedded in the 3D scene); captions, popup balloons, axes, labels, *x*-*y* plots, and many other classes.

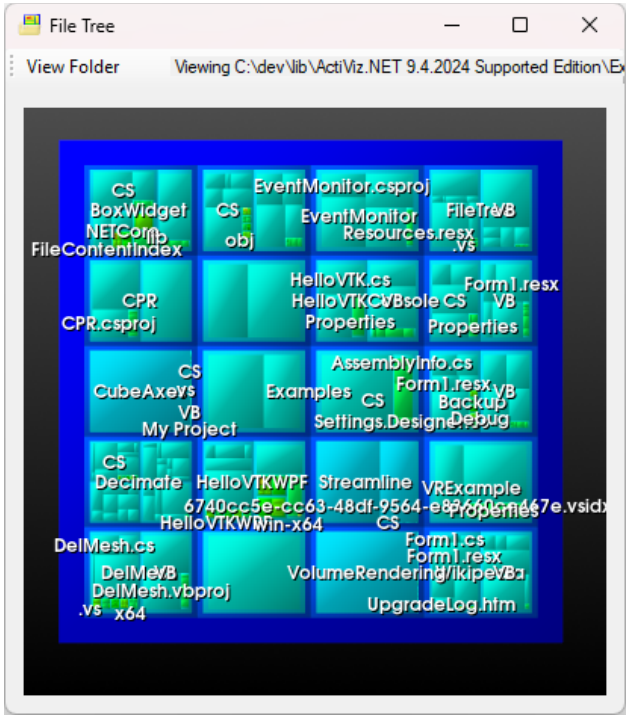
7.9. Decimation

Decimation, or polygon reduction, is a technique used in computer graphics to reduce the complex of geometric models. For example, in the figure below, the original model of *Big Buck Bunny* contains over 16,000 polygons; the decimated model contains approximately 8,000 polygons. In computer graphics and visualization, it is common to produce models with large polygon counts. Such models can be difficult to manipulate due to the delays in rendering and processing the data. Decimation is used to limit data sizes and thereby improve interaction rates.

In this example, models from the open-source movie *Big Buck Bunny* are used to demonstrate decimation (models copyright the Blender Foundation | www.bigbuckbunny.org). VTK provides several different decimation algorithms with varying levels of speed and fidelity. The fastest algorithm is vtkQuadricClustering, the algorithm that generates models with the best fidelity is vtkQuadricDecimation. In the example above, vtkDecimatePro is used. (See the included documentation for more information.) Since the source code is provided, interested readers may want to try different algorithms.



7.10. File Browser



The next example is a useful application for quickly locating large files and directories in a directory tree. The application uses VTK's tree map class and other information visualization classes. A tree map is a layout scheme that represents the "size" of each node in a tree (including the node's children) by a rectangle. In turn, the children are embedded in the parent's rectangle in a recursive process. The meaning of size varies depending on the application. In this example, size means the disk space usage of a file (if a leaf node) or the disk usage of the files and subdirectories contained within it (if an intermediate, directory node).

To use the application, simply load the directory you are interested in. Depending on the size of the directory and the subdirectories and files contained within it, the program may take anywhere from one or two seconds to minutes to execute. Once the image (shown) is generated, you can interact with the display. Labels are placed in the center of each rectangle and dynamically resize based as the user zooms in and out towards the tree map. Further, by moving the mouse pointer over the tree map a label will appear indicating the current directory or file at which you are pointing. Selecting (left mouse button) brings up a file browser. Use the right mouse button to zoom, and the center button to pan across the tree map.

(A cautionary note: if you run this example from Visual Studio in debug mode in very large directories, timeouts may occur that produce errors. The solution is not to run in debug mode.)

8. For More Information

The following section lists additional resources to learn more about VTK and Kitware.

8.1. Manual Pages

The ActiViz .NET product is distributed with extensive documentation on a per class basis. Refer to this documentation for details regarding the use of the examples shown here. A good way to view this is through Visual Studio's object browser view. Choose "Object Browser" from the view menu, and then browse through the Kitware.VTK classes.

8.2. VTK.org Web Site

The VTK open-source community maintains resources at http://www.vtk.org. Besides on-line documentation, user's and developer's mailing lists are available for posting and receiving answers to questions. Refer to vtk.org for more information on joining mailing lists.

8.3. More Examples

VTK has hundreds of additional examples available from which to learn about it. These examples are written in C++ or other VTK wrapping languages such as Python. In most cases these examples may be directly translated into a .NET supported programming language. A large number of tests, used to ensure the quality of VTK, are also useful references for learning about VTK. These tests and examples are great jumping off points for ActiViz users who wish to write their own .NET applications.

VTK examples: https://examples.vtk.org/site/Cxx

VTK tests are available in VTK source code, most tests are located in `<kit>/<module>/Testing/Cxx` for example https://gitlab.kitware.com/vtk/vtk/-/tree/master/Rendering/Core/Testing/Cxx.

8.4. VTK Books

VTK was created in 1993 as part of a textbook published by Prentice-Hall. Because the software was useful, and because of its open-source license, a community quickly grew up around the system. Since that time, the textbook has evolved and a supplemental *VTK User's Guide* has been written. These two books, listed below, are available through Kitware at http://www.kitware.com/products/books.html and Amazon.com.

- *The Visualization Toolkit An Object-Oriented Approach to 3D Graphics*. Schroeder, Martin, Lorensen — this is principally a theory book on visualization, although it contains many practical examples.
- *The VTK User's Guide* — shows how to use VTK through an extensive set of examples.

8.5. Related Software

Kitware creates and distributes many open-source and proprietary software systems. A synopsis of some of these systems follows.

Open Source Software (BSD or BSD-style licenses)

- The Visualization Toolkit (VTK) — provides 3D visualization capabilities including information visualization, volume rendering, modeling, data processing, and human-computer interaction tools (http://www.vtk.org).
- The Parallel Visualization Application (ParaView) — built on VTK, ParaView is a distributed, scalable visualization application designed for data sizes ranging from small to very large (http://www.paraview.org).
- The Insight Segmentation and Registration Toolkit (ITK) — provides a comprehensive suite of image processing, registration and segmentation tools for biomedical and other imaging tasks (http://www.itk.org).
- CMake, CTest, CPack, and CDash — these systems form the core of Kitware's quality software process. CMake is used to manage cross-platform development. CTest and CDash are the software testing client and server, respectively. CPack is used to package and distribute software across multiple computing platforms (http://www.cmake.org, http://www.cdash.org).
- 3D Slicer — is a biomedical application built on VTK and ITK. It has been successfully employed for medical and biological imaging applications (http://www.slicer.org).

ActiViz .NET OpenSource Edition

ActiViz .NET Supported Edition