LEARNING vtk

#vtk

Table of Contents

About

Chapter 1: Getting started with vtk

Remarks

This section provides an overview of what vtk is, and why a developer might want to use it.

It should also mention any large subjects within vtk, and link out to the related topics. Since the Documentation for vtk is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

Building and Installation on Windows 7

Prerequisites

- If you want to build VTK from latest sources you need git from Here or you can download a snapshot of the code as a zip and unzip on to your disk drive
- CMake
- Microsoft Visual Studio 2015
- Plenty of free space - atleast a couple of GB to be on the safe side, really depending on what all you want to build

Getting Ready

- I like to keep things clean so I usually create 3 folders like so:

```
c:\vtk                #
c:\vtk\src            # 'code base' folder
c:\vtk\build          # 'out of source' build folder
c:\vtk\install        # 'install folder' where the 'installed' files will reside
```

- If using the git method,
    - open a command prompt
    - change working directory `cd c:\vtk\src`
    - clone the git repository `git clone https://gitlab.kitware.com/vtk/vtk.git`. This could take a while depending on your internet connection speed
    - If you are working behind a proxy, you will need to setup git to use it. See this question on how to do that.

- If using the zip method, unzip the source code into `c:\vtk\src`

Configuration

- Launch the CMake GUI
- Select `c:\vtk\src` for `Where is the source code:`
- Select `c:\vtk\build` for `Where to build the binaries:`
- Hit `Configure` and select `Visual Studio 2015` as the required generator
- You will be presented with a number of configuration options
- I generally use the following settings for a minimal build
  - `CMAKE_INSTALL_PREFIX` = `c:\vtk\install`
  - `BUILD_SHARED_LIBS` ticked
  - `BUILD_DOCUMENTATION` unticked
  - `BUILD_TESTING` unticked
  - `CMAKE_CXX_MP_FLAG` ticked. This will use all the CPU cores (on multicore/multiprocessor systems) to speed up the build
- Keeping Hitting `Configure` correcting any errors until all the RED entries become WHITE
- Hit `Generate`
- Close CMake GUI

Building

- If the generation was successful there should be
  - A Visual Studio Solution :

    ```
    c:\vtk\build\vtk.sln
    ```

  - A bunch of project files -

    ```
    ALL_BUILD.vcxproj
    INSTALL.vcxproj
    vtkCompileTools.vcxproj
    VTKData.vcxproj
    ZERO_CHECK.vcxproj
    ```

- You can build this using either from a command line or using the IDE
- I prefer the command line as it is generally faster and uses less RAM
- Using the command line
  - Launch `Developer Command Prompt For Visual Studio 2015`
  - Change working directory: `cd c:\vtk\build`
  - Launch msbuild:
    - for debug builds
      - `msbuild /p:Configuration=Debug ALL_BUILD.vcxproj`
      - `msbuild /p:Configuration=Debug INSTALL.vcxproj`
    - for release builds
      - `msbuild /p:Configuration=Release ALL_BUILD.vcxproj`
      - `msbuild /p:Configuration=Release INSTALL.vcxproj`
- Using the IDE
  - Open the `VTK.sln` with Visual Studio 2015 and build the `INSTALL.vcxproj`
  - This technique is usually slower as the IDE will start building intellisense for each of the

projects listed in the solution

- `c:\vtk\install` should now have some new folders
    - `bin` # contains the dll files
    - `lib` # contains the lib files
    - `cmake`
    - `share`
    - `include` # contains the header files

Using the build

- To Use VTK in a Visual C++ project, one has to
    - Configure the compiler header file search path to include `c:\vtk\include\vtk-<version>`
    - Configure the linker library file search path to include `c:\vtk\lib`
    - Configure the linker to link to the required `.lib` files
    - Copy the required DLLs to output folder
- I have put together a small props file to handle all the four tasks `c:\vtk\vtk.vsprops`

```
<?xml version="1.0" encoding="UTF-8"?>
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
    <PropertyGroup>
        <VTK_ROOT_DIR>$(MSBuildThisFileDirectory)</VTK_ROOT_DIR>
        <VTK_BIN_DIR>$(VTK_ROOT_DIR)\bin</VTK_BIN_DIR>
        <VTK_INC_DIR>$(VTK_ROOT_DIR)\include\vtk-7.0</VTK_INC_DIR>
        <VTK_LIB_DIR>$(VTK_ROOT_DIR)\lib</VTK_LIB_DIR>
    </PropertyGroup>

    <PropertyGroup>
        <BuildDependsOn>CopyVTKBinariesList;$(BuildDependsOn);</BuildDependsOn>
    </PropertyGroup>

    <Target Name="CopyVTKBinariesList">
        <ItemGroup>
            <VtkBinaries Include="$(VTK_BIN_DIR)\*.dll" />
        </ItemGroup>
        <Copy SourceFiles="@(VtkBinaries)"
              DestinationFiles="@(VtkBinaries-
>'$(OutDir)\%(RecursiveDir)%(Filename)%(Extension)')"
              SkipUnchangedFiles="true" />
    </Target>

    <ItemDefinitionGroup>
      <ClCompile>

<AdditionalIncludeDirectories>$(VTK_INC_DIR);%(AdditionalIncludeDirectories)</AdditionalIncludeDirector

      </ClCompile>
      <Link>

<AdditionalLibraryDirectories>$(VTK_LIB_DIR);%(AdditionalLibraryDirectories)</AdditionalLibraryDirector

        <AdditionalDependencies>vtkalglib-7.0.lib;vtkChartsCore-7.0.lib;vtkCommonColor-
7.0.lib;vtkCommonComputationalGeometry-7.0.lib;vtkCommonCore-7.0.lib;vtkCommonDataModel-
7.0.lib;vtkCommonExecutionModel-7.0.lib;vtkCommonMath-7.0.lib;vtkCommonMisc-
7.0.lib;vtkCommonSystem-7.0.lib;vtkCommonTransforms-7.0.lib;vtkDICOMParser-
7.0.lib;vtkDomainsChemistry-7.0.lib;vtkDomainsChemistryOpenGL2-7.0.lib;vtkexoIIc-
7.0.lib;vtkexpat-7.0.lib;vtkFiltersAMR-7.0.lib;vtkFiltersCore-7.0.lib;vtkFiltersExtraction-
```

```
7.0.lib;vtkFiltersFlowPaths-7.0.lib;vtkFiltersGeneral-7.0.lib;vtkFiltersGeneric-
7.0.lib;vtkFiltersGeometry-7.0.lib;vtkFiltersHybrid-7.0.lib;vtkFiltersHyperTree-
7.0.lib;vtkFiltersImaging-7.0.lib;vtkFiltersModeling-7.0.lib;vtkFiltersParallel-
7.0.lib;vtkFiltersParallelImaging-7.0.lib;vtkFiltersProgrammable-7.0.lib;vtkFiltersSelection-
7.0.lib;vtkFiltersSMP-7.0.lib;vtkFiltersSources-7.0.lib;vtkFiltersStatistics-
7.0.lib;vtkFiltersTexture-7.0.lib;vtkFiltersVerdict-7.0.lib;vtkfreetype-7.0.lib;vtkGeovisCore-
7.0.lib;vtkglew-7.0.lib;vtkhdf5-7.0.lib;vtkhdf5_hl-7.0.lib;vtkImagingColor-
7.0.lib;vtkImagingCore-7.0.lib;vtkImagingFourier-7.0.lib;vtkImagingGeneral-
7.0.lib;vtkImagingHybrid-7.0.lib;vtkImagingMath-7.0.lib;vtkImagingMorphological-
7.0.lib;vtkImagingSources-7.0.lib;vtkImagingStatistics-7.0.lib;vtkImagingStencil-
7.0.lib;vtkInfovisCore-7.0.lib;vtkInfovisLayout-7.0.lib;vtkInteractionImage-
7.0.lib;vtkInteractionStyle-7.0.lib;vtkInteractionWidgets-7.0.lib;vtkIOAMR-7.0.lib;vtkIOCore-
7.0.lib;vtkIOEnSight-7.0.lib;vtkIOExodus-7.0.lib;vtkIOExport-7.0.lib;vtkIOGeometry-
7.0.lib;vtkIOImage-7.0.lib;vtkIOImport-7.0.lib;vtkIOInfovis-7.0.lib;vtkIOLegacy-
7.0.lib;vtkIOLSDyna-7.0.lib;vtkIOMINC-7.0.lib;vtkIOMovie-7.0.lib;vtkIONetCDF-
7.0.lib;vtkIOParallel-7.0.lib;vtkIOParallelXML-7.0.lib;vtkIOPLY-7.0.lib;vtkIOSQL-
7.0.lib;vtkIOVideo-7.0.lib;vtkIOXML-7.0.lib;vtkIOXMLParser-7.0.lib;vtkjpeg-7.0.lib;vtkjsoncpp-
7.0.lib;vtklibxml2-7.0.lib;vtkmetaio-7.0.lib;vtkNetCDF-7.0.lib;vtkNetCDF_cxx-
7.0.lib;vtkoggtheora-7.0.lib;vtkParallelCore-7.0.lib;vtkpng-7.0.lib;vtkproj4-
7.0.lib;vtkRenderingAnnotation-7.0.lib;vtkRenderingContext2D-
7.0.lib;vtkRenderingContextOpenGL2-7.0.lib;vtkRenderingCore-7.0.lib;vtkRenderingFreeType-
7.0.lib;vtkRenderingImage-7.0.lib;vtkRenderingLabel-7.0.lib;vtkRenderingLOD-
7.0.lib;vtkRenderingOpenGL2-7.0.lib;vtkRenderingVolume-7.0.lib;vtkRenderingVolumeOpenGL2-
7.0.lib;vtksqlite-7.0.lib;vtksys-7.0.lib;vtktiff-7.0.lib;vtkverdict-7.0.lib;vtkViewsContext2D-
7.0.lib;vtkViewsCore-7.0.lib;vtkViewsGeovis-7.0.lib;vtkViewsInfovis-7.0.lib;vtkzlib-
7.0.lib;%(AdditionalDependencies)</AdditionalDependencies>
      </Link>
    </ItemDefinitionGroup>
    <ItemGroup />

</Project>
```

- The above vsprops file copies all the available dlls in the `c:\vtk\bin` folder.
- An alternative way to make sure the DLLs can be located is to use alter the PATH environment variable for the debugging session and put the VTK binaries path as the first directory to be searched when loading dependencies. The below fragment can be instead of the `CopyVTKBinariesList` task to do this.

  ```
  <PropertyGroup>

  <LocalDebuggerEnvironment>PATH=$(VTK_BIN_DIR);%PATH%;$(LocalDebuggerEnvironment)</LocalDebuggerEn

  </PropertyGroup>
  ```

- For final deployment you might want to use a tool like Dependency Walker to track down which dlls and their dependencies are used and only bundle only those for redistribution.
- To use the props file in a Visual C++ project you can either use the Property Manager tool within Visual Studio (Menu: View => Property Manager) or edit the vcxproj using a text editor and add this following line `<Import Project="C:\vtk\vtk.vsprops" />` below the other project imports.

Cleaning Up

- If you like to recover some disk space, you can delete the `c:\vtk\build` folder but the downside is you cannot debug into vtk

Uninstallation

- Simply delete the `c:\vtk` folder if you dont want to VTK anymore

MacOSX and Unix:

1. Install the latest version of CMake available here
2. Download the latest VTK here.
3. Create a build directory for VTK `mkdir <path_to_build_directory`
4. Configure with `ccmake <path_to_VTK_directory -G "UNIX Makefiles" \ -DVTK_USE_QVTK:BOOL=ON \ -DVTK_USE_CARBON:BOOL=ON \ -DCMAKE_INSTALL_PREFIX=/usr/local \ -DVTK_USE_GUISUPPORT:BOOL=ON` or use the GUI to do so with `ccmake <path_to_VTK_directory`
5. Get into the build directory and use `make -j`(you don't have to use `-j` but compilation is really long.
6. Finally use `make install`

Read Getting started with vtk online: https://riptutorial.com/vtk/topic/5182/getting-started-with-vtk

Chapter 2: Hello World

Examples

Hello World Example

```
#include <vtkAutoInit.h>

VTK_MODULE_INIT(vtkRenderingOpenGL2);
VTK_MODULE_INIT(vtkRenderingFreeType);
VTK_MODULE_INIT(vtkInteractionStyle);

#include <vtkSmartPointer.h>
#include <vtkTextActor.h>
#include <vtkRenderer.h>
#include <vtkRenderWindow.h>
#include <vtkRenderWindowInteractor.h>

int main(int /*argc*/, char ** /*argv*/)
{
    auto textActor = vtkSmartPointer<vtkTextActor>::New();
    textActor->SetInput("Hello World");

    auto renderer = vtkSmartPointer<vtkRenderer>::New();
    renderer->AddActor(textActor);
    renderer->ResetCamera();

    auto interactor = vtkSmartPointer<vtkRenderWindowInteractor>::New();

    auto renderWindow = vtkSmartPointer<vtkRenderWindow>::New();
    renderWindow->AddRenderer(renderer);
    renderWindow->SetInteractor(interactor);

    interactor->Start();

    return 0;
}
```

Breakdown:

```
#include <vtkAutoInit.h>

VTK_MODULE_INIT(vtkRenderingOpenGL2);
VTK_MODULE_INIT(vtkRenderingFreeType);
VTK_MODULE_INIT(vtkInteractionStyle);
```

VTK design uses a factory method design pattern to create new instances of vtkObject derived classes using the `<ClassName>::New()` method. This allows a platform specific implementation to be selected during runtime to satisfy a required interface.

For this mechanism to work, factory classes need to "register" themselves so that they can be selected by the vtk infrastructure. Details on this topic is available here.

`VTK_MODULE_INIT` is a macro used to automatically initialize the required modules/library(ies)(`vtkRenderingOpenGL2`, `vtkRenderingFreeType`, `vtkInteractionStyle` in this example). Failure to initialize the modules will result in `<ClassName>::New()` calls to return `NULL` and therefore runtime errors.

```
#include <vtkSmartPointer.h>
#include <vtkTextActor.h>
#include <vtkRenderer.h>
#include <vtkRenderWindow.h>
#include <vtkRenderWindowInteractor.h>
```

`vtkSmartPointer` role is similar to that of a `std::unique_ptr` in that it manages the reference count that controls the lifetime of `vtkObject` derived class instances.

`vtkTextActor` is a simple class that can be used to display strings on the screen.

`vtkRenderer` is a class responsible for managing a scene's contents. Specifically it manages the collection of

- 2D actors derived from `vtkActor2D`
- 3D actors derived from `vtkProp3D`
- Volumes : `vtkVolume`
- Camera : `vtkCamera`
- Lights : `vtkLight`

`vtkRenderWindow` is a class that provides platform independent interface for

- managing a collection of renderers.
- handling user input and forwarding it to `vtkRenderWindowInteractor` for further processing

`vtkRenderWindowInteractor` is a class responsible for mapping the user input (mouse/keyboard/timing) events to a corresponding action. Internally it uses a `vtkInteractorStyle` to provide different mapping behaviors.

```
auto textActor = vtkSmartPointer<vtkTextActor>::New();
textActor->SetInput("Hello World");
```

Create text actor and set the string to display

```
auto renderer = vtkSmartPointer<vtkRenderer>::New();
renderer->AddActor(textActor);
renderer->ResetCamera();
```

- Create a renderer
- Add the text actor to it
- Resets the camera position to make sure the actor is visible in the screen.

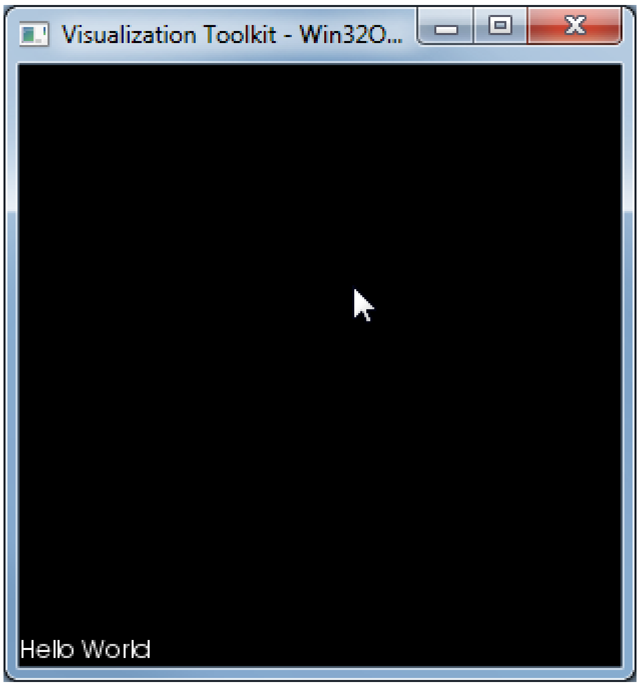
```
auto interactor = vtkSmartPointer<vtkRenderWindowInteractor>::New();
auto renderWindow = vtkSmartPointer<vtkRenderWindow>::New();
renderWindow->AddRenderer(renderer);
renderWindow->SetInteractor(interactor);
```

Create a window to render into, add the renderer to it and set the interactor. The factory function will automatically pick a suitable implementation based on the available/registered factory classes

```
interactor->Start();
```

This is a blocking call that returns only when the user requests a quit (`q` key) or closes the window. Runs a message loop and dispatches the messages.

Running this should create a window that looks like this



Hello World

Notes

This list of DLLs that were used by this exe are:

`VTKCommonCore-7.0.DLL`

`VTKInteractionStyle-7.0.DLL`

`VTKRenderingCore-7.0.DLL`

`VTKRenderingFreeType-7.0.DLL`

`VTKRenderingOpenGL2-7.0.DLL`

Read Hello World online: https://riptutorial.com/vtk/topic/5974/hello-world

Credits

S. No	Chapters	Contributors
1	Getting started with vtk	Community, LBes, Shreyas Murali
2	Hello World	Shreyas Murali