EEEE2076 - Software Development Group Design Project
Group Work

P Evans

March 6, 2025

Contents

1 Overview

The group work should follow on from your final individual exercise: the individual worksheets get you to the point where you have a working GUI and VTK install, you now need to add VR to your GUI and implement some features. This document covers two areas:

- Some technical help covering VR implementation and a couple of other VTK topics.
- An explanation of the features required and marking scheme.

2 VTK Support

This section is like a final worksheet, but related to the group work rather than individual. You do not need to add these features to your individual repositories!

2.1 Adding VR to your GUI

As discussed in the lecture, you need a separate thread which handles the VR rendering. This thread can be started when the user clicks a button. The process required is:

- Use clicks start VR button
- New VRRenderThread class created
- A list of actors that will be rendered is added to the VRRenderThread class, this is done using the addActorOffline() function. These actors **cannot** be the same actors as the ones that have been added to the Qt renderer. They must also be linked to new mappers. This means for each ModelPart, a new mapper must be created and linked to the STLReader source, a new actor created and linked to this mapper, the new actor added to the VRRenderThread class.
  If you have modified (or will modify) the properties of the actor in the GUI (e.g. colour, position) and would like these modifications to also be reflected in the VR then you need to link the properties of the old actor to the new actor. This can be done with a line of code similar to newActor−>SetProperty(actor −>GetProperty());
- The VR Thread is started, it creates the VR renderer and adds the list of new actors to the render. You just need to copy code from the previous VR example.

A example VRRenderThread class is available here. You just need to make it work, you will need a button/action to start the VR and to create an instance of the VRRenderClass as a member variable of MainWindow. You'll also need to populate the addNewActor() function in the ModelPart class so that new actors can be generated for each ModelPart.

2.2 Lighting

Lights can be added to the scene in the following way:

```
vtkSmartPointer<vtkLight> light = vtkSmartPointer<vtkLight>::New();
light->SetLightTypeToSceneLight();
light->SetPosition( 5, 5, 15 );
light->SetPositional( true );
light->SetConeAngle( 10 );
light->SetFocalPoint( 0, 0, 0 );
light->SetDiffuseColor( 1, 1, 1 );
light->SetAmbientColor( 1, 1, 1 );
light->SetSpecularColor( 1, 1, 1 );
light->SetIntensity( 0.5 );

// now add actors ...

// Render
ui->vtkWidget->GetRenderWindow()->Render();

// add the light to the renderer after Render was called
renderer->AddLight( light );
```

You can use: vtkLightCollection∗ vtkRenderer::GetLights() elsewhere in your code to retrieve a list of the lights in your *vtkRenderer* and make changes to properties such as light position, direction, intensity or colour.

2.3 Filters

Filters are classes that sit between the mapper and actor, they modify (or filter) the data to apply visual effects.
Typically your code that connects a mapper to actor looks like:

```
MAPPER->SetInputConnection( SOURCE->GetOutputPort() );
```

Filter classes can be inserted into this chain to affect what is rendered. The resulting code could look something similar to:

```
// this will apply a clipping plane whose normal is the x-axis that crosses the x-axis at x=0
FILTER1->SetInputConnection( SOURCE->GetOutputPort() );
FILTER2->SetInputConnection( FILTER1->GetOutputPort() );
MAPPER->SetInputConnection( FILTER2->GetOutputPort() );
```

Obviously the filter classes need to be configured and there a lots of different filter options that you can choose from. Examples include:

2.3.1 Clip Filter

```
// this will apply a clipping plane whose normal is the x-axis that crosses the x-axis at x=0
vtkSmartPointer<vtkPlane>  planeLeft = vtkSmartPointer<vtkPlane>::New();
planeLeft->SetOrigin(0.0, 0.0, 0.0);
planeLeft->SetNormal(-1.0, 0.0, 0.0);

vtkSmartPointer<vtkClipDataSet> clipFilter = vtkSmartPointer<vtkClipDataSet>::New();
clipFilter->SetInputConnection( SOURCE->GetOutputPort() ) ;
clipFilter->SetClipFunction( planeLeft.Get() );

MAPPER->SetInputConnection( clipFilter->GetOutputPort() );
```

2.3.2 Shrink Filter

```
vtkSmartPointer<vtkShrinkFilter> shrinkFilter = vtkSmartPointer<vtkShrinkFilter>::New()
    ;
shrinkFilter->SetInputConnection( SOURCE->GetOutputPort() );
shrinkFilter->SetShrinkFactor(.8);
shrinkFilter->Update();

MAPPER->SetInputConnection( shrinkFilter->GetOutputPort() );
```

3 Group Tasks

The following is an introduction to the features you should aim to implement for your group work. These are not expected to be present in your individual repository, however you can use your individual repository as a testing area if you like.

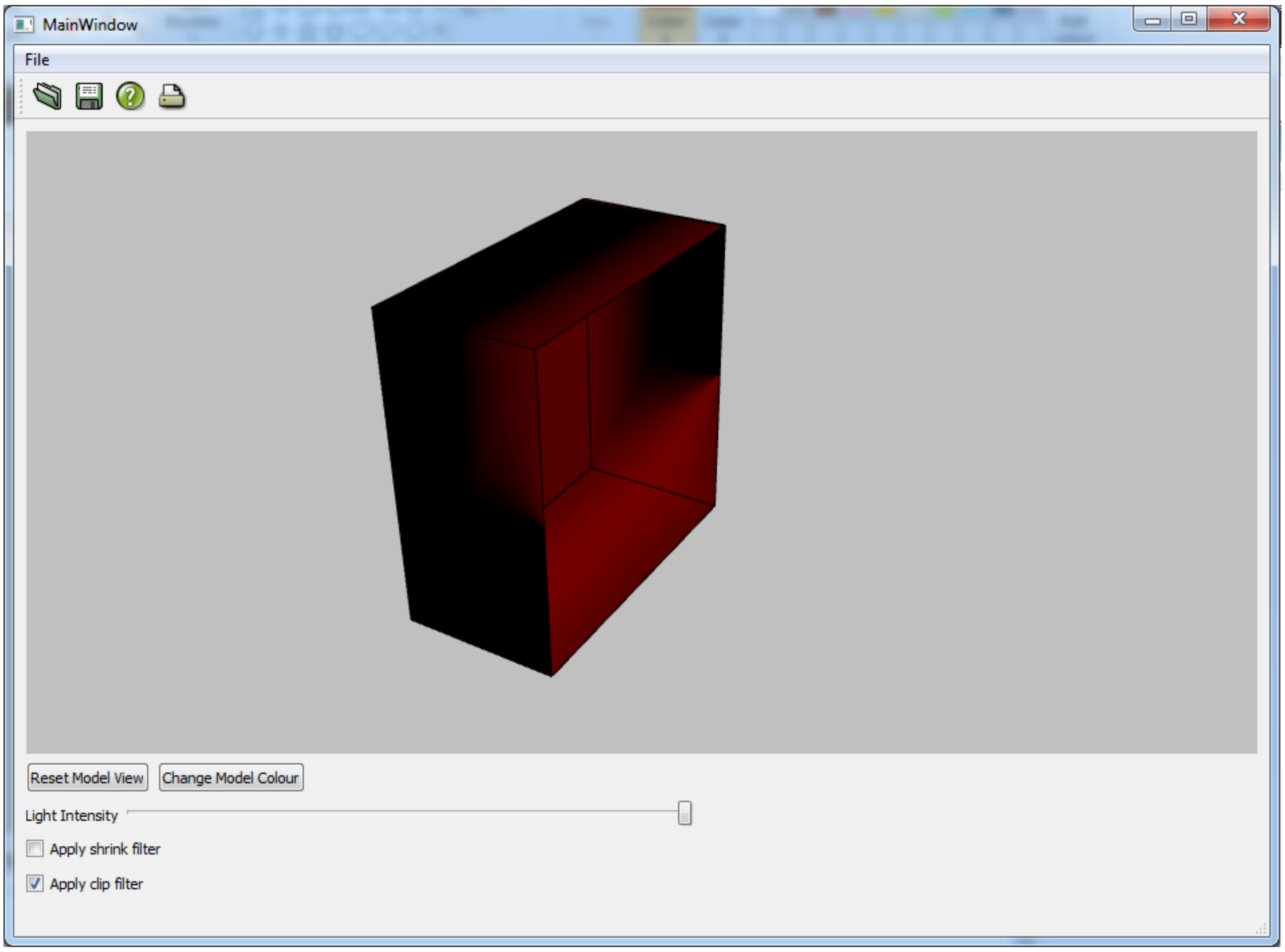

Figure 1: Clip Filter

The output from your Qt and VTK individual worksheets should form the basis for this work.

3.1 GUI Assessed Via Installer uploaded to Moodle (Also see (1) in marking rubrics)

You need to upload your final installer on Moodle
There should be at least one submission per group before the deadline, in the event that multiple group members submit a different commit ID before the deadline then I will contact you to see which to mark. If there is a submission before the deadline from anyone then this will be marked, I will not mark a late submission from someone else made after the deadline. If all submissions are late then the first submission will be marked.

You should also have this as a release, but to avoid any confusion: I will mark what you upload to Moodle. This was it is clear what you want marking in the event you have multiple releases on Github, it also finalises your submission before the deadline in a way that is clearly recorded.

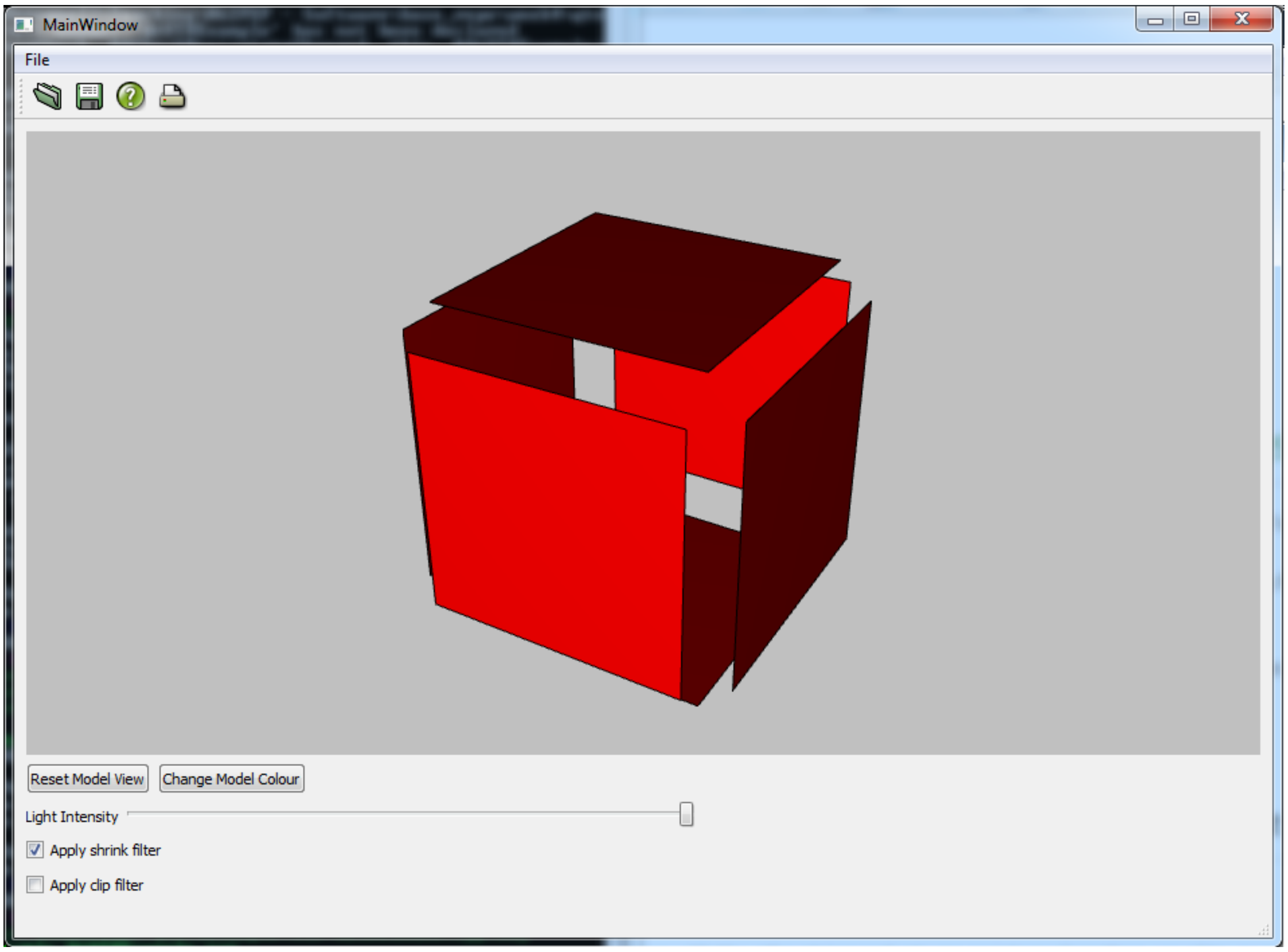

Figure 2: Shrink Filter

3.1.1 Installer (40%)

- The installer contains the program, irrespective of any other libraries / examples.
- The installer is a typical Windows type as generated by NSIS.
- There is a startmenu shortcut which includes a link to the uninstaller
- The uninstaller works and removes all trace of the program and anything else installed with it.
- The program works standalone - by this I mean that I don't have to have installed Qt/VTK/OpenVR myself - these are packaged in the installer. A good way to test this is to give it to one of your friends who does not have these libraries installed and get them to test. Note the VR function will not work if they don't have a VR system connected, but the program should still run and allow the model to be viewed/edited in the GUI.

3.1.2 Application (60%)

- Loading CAD models: You can load multiple CAD models and have them displayed together (this is a carry-over from the individual work). For the group work, there are extra marks if you can add a feature that loads all files from a directory simultaneously (see the example program provided on Moodle which implements this)
- The treeview functionality is the same as the individual worksheet requirements - new models / sections of model can be added at any point in the tree as children of existing items and the names of the items reflect the loaded filenames. For the group work you should also be able to remove selected items.
- A context menu (right click in tree) can be used to edit item properties. These properties include:
  - Filters - two working filters can be applied to any item in any order. This will be tested in the GUI only at this stage (not VR).
  - The colour of any item can be changed independently
  - The visibility of any item in the 3D model can be changed independently.
  - The name of any item can be changed.
- There are marks for the design of the GUI - use of toolbars/menus, layout that scales properly, appearance that is typical of a Windows application
- Stability - Does it crash when I test it? The application should also give be information by status bar - e.g. tell me what I just added, tell me if I try to do something I can't (e.g. add something when I don't have a selection in the tree).

3.2 Code assessed Via Github (Also see (2) in marking rubrics)

Your repository is marked after the final project week.

You need to submit the commit ID to Moodle
Instructions for how to do this are on the Moodle page. There should be at least one submission per group before the deadline, in the event that multiple group members submit a different commit ID before the deadline then I will contact you to see which to mark. If there is a submission before the deadline from anyone then this will be marked, I will not mark a late submission from someone else made after the deadline. If all submissions are late then the first submission will be marked.

I will mark the code on Github, but the commit ID submission is used to clearly mark which version of the code you want marking and records your submission time on university systems.

3.2.1 Code (25%)

Quality of code written, correct use of Github.

- Code compiles, using CMake, without me needing to fix things.
- Code structure - basic OOP principles use of classes, formatting of code, consistent variable naming style, consistent file naming style
- GUI - Code working properly, can I run the program from Visual Studio without errors.

3.2.2 Commenting (15%)

Use of Doxygen, and code commenting in general, should be self-explanatory.

- Doxygen comments for all member variables of classes, describing variable usage
- Doxygen comments for all member functions of classes, describing usage, arguments and return values.
- Doxygen file header comments, describing purpose of all source files.
- Explanatory comments (non-Doxygen) included in code where appropriate

3.2.3 Online Documentation (20%)

- Documentation on Github.io - There is a documentation website on Github.io that accurately describes your code. This does not have to generated using actions for these marks (you can upload to your gh-pages branch directly if you have an issue with actions).
- As above, but this time you have got Github Actions to generate it automatically when your master/main branch is pushed.
- Creative use of Doxygen - this just means you have made some effort to customise the template.

3.2.4 Git (15%)

- General structure - appropriate use of sub-directories: repository is easy to read, code in a sensible sub-directory structure, not just all source dumped in the root folder.
- README files in folders, these should briefly explain folder contents, how to compile, etc. If someone went to your repository, could they work out how to use/compile your code?
- A number of commits from each group member, evidence of a sustained development process starting towards the end of the 2nd project week. Not just everything uploaded on the final Friday by one group member.
- A good, working .gitignore
- No binaries or build folders and a repository that does not contain 100's of MB of history due to added & deleted build files.
- Installer added as release.

3.2.5 Installer (25%)

- A working installer can be created from within Visual Studio via the INSTALL target.
- Your installer pulls in the "required system libraries" using the appropriate CMake routine. This should pull in the Microsoft Visual C++ runtime libraries. Two points to note: 1) These are DLLs that do not automatically existing on Windows but are installed with Visual Studio, if you try to run your software on a computer without VS and you haven't bundled these libraries then it won't work! 2) There are both "Release" and "Debug" versions of the runtime, if you bundle the wrong one then it won't work.

- The installer bundles Qt DLLs, as a minimum this means that if I build the installer, the necessary Qt DLLs will be included within it. At a basic level, I might have to tell CMake where my Qt DLLs are and the installer might just bundle all Qt DLLs. A better CMake script will find a better way of doing this - automatically finding the DLLs. An even better script will be able to determine exactly which DLLs are needed and only bundle these, not just all of them. Note - you need more than just the DLLs for it to work properly, there are some other files from Qt that are needed. You may want to read this: https://doc.qt.io/qt-6/windows-deployment.html
- As above, but for VTK.
- Open VR library is also bundled automatically.

3.3 VR Assessed Via Demonstration (See also (4) in marking rubrics)

You demonstrate your software as a group.

3.3.1 Setup (20%)

I want you to download your installer and run it. Initially Qt,VTK, OpenVR will be removed from the path on the test PC - this means your installer needs to install these.

- Marks if the installer works installs the main program and the program will run (with or without Qt, VTK OpenVR on the path - if the program won't run to begin with, they will be readded to the path)
- The installer installs everything required for the demo (e.g the model) so nothing else has to be downloaded independently.
- You can load a model
- Standalone - these marks given if we didn't have to add Qt, VTK, OpenVR back onto the path.

3.3.2 Basic Functionality - does it run? (20%)

Show that you can load a model and start the VR.

- VR can be started
- The model loaded in displayed in VR correctly.
- The hand controller works - you can drag sections of model. See demo provided on Moodle for example.
- Ability to stop/restart VR multiple times without it crashing or any other bugs!

3.3.3 VR (50%)

Show your VR features, explain the menus, buttons, etc and demonstrate that these work in VR.

- Demonstrate two filters working. These can be applied to parts of the model independently (e.g. only to a wheel), and can be applied in any combination. (as for GUI marked in Moodle submission, but this time for VR)
- As above but for colour
- You can change things in the GUI and the effect is seen in VR, while it is running. E.g. changing filters and colour. Ideally any extra STL files added in the GUI will also immediately appear in VR and be editable.

- Virtual environment - can you add a floor, scenery, etc either manually or ideally with a Skybox.
- Add some animation: e.g. the rotation hinted at in the renderThread class for a basic implementation but if you can do anything better (explode or return dragged parts to original locations) then this would be a perfect solution.
- Marks for a good demonstration - use of a model with appropriate levels of detail (lots of parts so it is interactive, but not too slow), and a demonstration that works and doesn't need to be continually restarted because of crashes / bugs.
- The marks listed above will get you 70% of the VR category marks. The rest can come from your own ideas: See what you can find in the VTK: additional VTK filters, maybe overlaying simulated air-flows, interactivity other than the basic dragging feature that is already enabled, impressive lighting. You can get up to 50% of the category marks here, so the total available is 120%, obviously it must be capped at 100% but the idea is that a good idea of your own can compensate for some of the other items.