A Project Report on Applied Geometry and Special Effects

James Pandey

December 14, 2017

https://github.com/pandeyjames/AppGeoMod

Abstract

This document is a Project Report on the assigned task for STE6247-Applied Geometry and Special Effects.

This report contains the details of how the task is finalized with added creating the support class for the curves, splines and surfaces to demonstrate how it will exactly can be build in Geometrical Modeling and how their mathematical model can be implemented. The task is done using hardcoded mathematical operations rather than advance option that is already available in GMlib/site GMlib. are integrated and demonstrated, including some basic algorithms and methods.

The C++ programming language is used to build the application which was provided as a initial setup from Masters of Computer Science study at UiT, Campus Narvik. The application depends on GMlib: Geometric Modeling Library v 0.6.0, and the Qt development suite. The basic features was provided ready-made, with the initial setup and is tasked to implement and build an application which can be used as a template for new applications throughout the MSc computer science study at UiT - Campus Narvik with added functionality and new features.

1 Introduction

The theoritical background of programming in C++ [1] is achived during earlier studies. Here it is a practical bases of the theories which were studied in the past now has a direct implementation using guide and nomenclature of Qt platform [2] The C++ programming language comes with a standard library, the *Standard Template Library* (STL), which consists of containers, methods, algorithms and more.

Qt (usually pronounced as "cute",) is used mainly for developing application software with graphical user interfaces (GUIs); however, programs without a GUI can be developed, such as command-line tools and consoles for servers. Qt is a cross-platform application framework that is widely used for developing application software that can be run on various software and hardware platforms

with little or no change in the underlying code-base, while still being a native application with the capabilities and speed thereof. Using Qt framework with the Gmlib via GLEW 2.0 an openGl framework a different approach to Graphic Project is carried out.

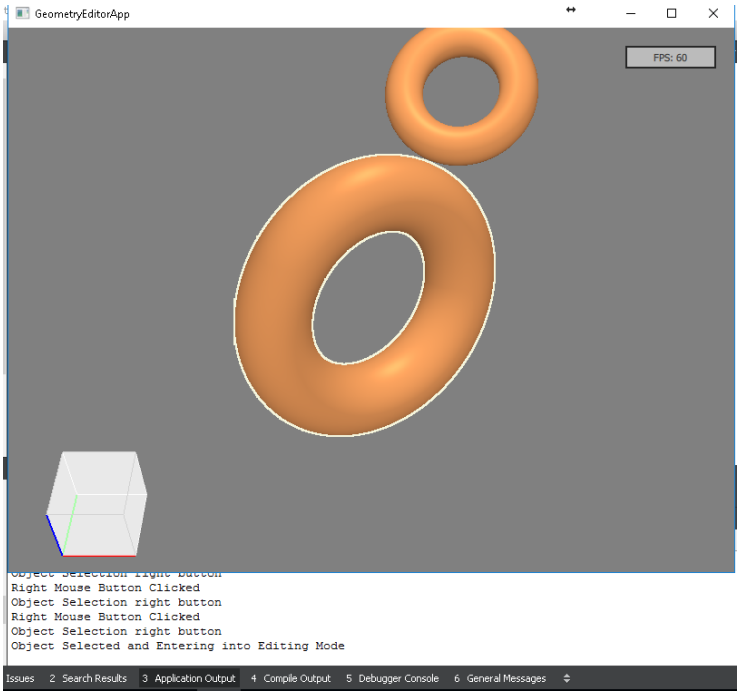

Figure 1: Demonstration of the Application selection of the object

2 Methods

Methods as a way to describe the project includes some various functionality. At the very first default key bind and a simulating Torus object is provided by default in the Application. To develop the application further it is mandatory minimal requirement to bind the keys and mouse events to the default functions for this Qt handles events as signal and slots. The main idea behind this is to connect the events of keyboard and mouse to the application as QEvent and then the identified events can be used to perform different functions. The mouse and keyboard events such as key presses and button clicked and wheel events are distinguished and then additionally, they are used further to call various methods built in the scenario class as per requirement. Here some functions like selection, deselection, camera movement, object rotation, scale, move, insert and delete can be performed. The main theme of task is to provide manipulation

of the objects like a geometrical object such as Torus, sphere, cone, beizier, cylinder, editor, performing the different operations upon.

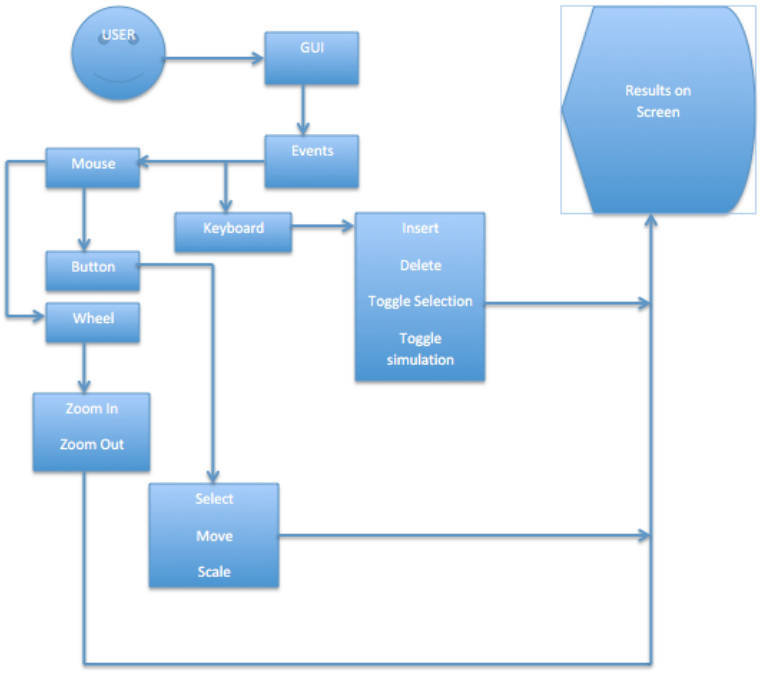

Figure 2: Demonstration the flow of the diagram

3 Discussion

As of now the project concludes with the implementations of basic functionality. A head-start with the application is to first setup the system which require a graphic card supports openGl version above 4, experienced with version 3 so it is recommended above v 4. If you are a developer and want to further develop the program you need to prepare your system first with Glew v 2.0 and Gmlib v 0.6.0 and of-course have a Qt mingw32 v 5.7 installed. For the user of this program only needs to run the supplied released executable file with supported libraries and dll's.

As it's an individual project the features which were to be added can't be implemented properly cause one should focus each and every aspect of the project. If it was a group project then more outstanding outcome could be seen, whereas one must only focus on one module and enhance it as the best. This is the main difficulty of the project and lack of references and proper documentation and examples of the implementation of GMlib library was a new thing to approach, which obviously increase the complexity of the task. And limited time frame of only couple of weeks also hardens the way for the accomplishment. Some key points of topics for this section:

- The application is only developed and tested for windows may not be compatible for other system.
- The version of Glew and the Gmlib pre-match is mandatory otherwise the program may crash with failure.
- Qt framework is a very user friendly with the documentation and help provided online and offline.
- For now the program only runs under windows system future enhancement may include running under other systems.
- A full featured application which can work with the geometrical objects like the software, Auto Cad, Solid work, is an example of the Geometrical Editor. This project may be enhanced with such features in future.

4 Results

As we can see the results of the program outcome various output of the application are listed herewith. Every function can't be present out properly here in the report as a pictorial form. So, here some prime functions are included as pictures.

A figure below demonstrates a new object sphere insertion in the scene with the main torus deleted. fig. 3.

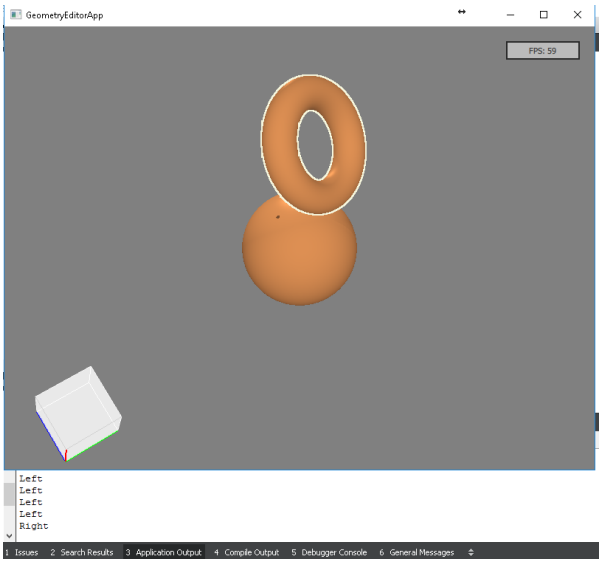

Figure 3: Demonstration of the Application insertion and deletion of the object

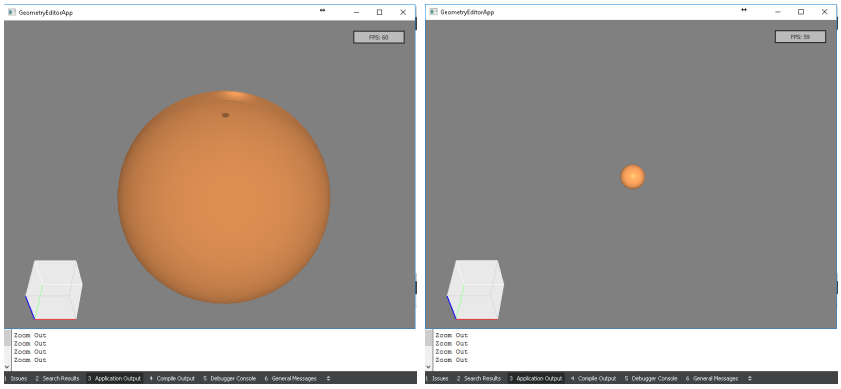

Figure 4: Demonstration of the Application zoom in zoom out of object

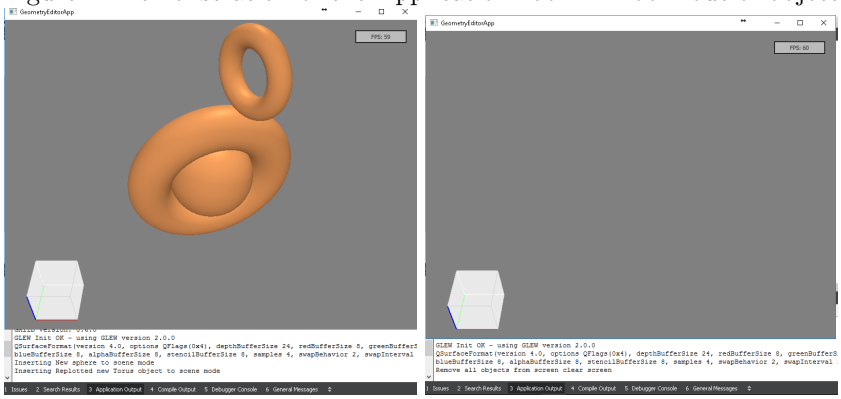

Figure 5: Demonstration of the Application inserting sphere and clearing screen

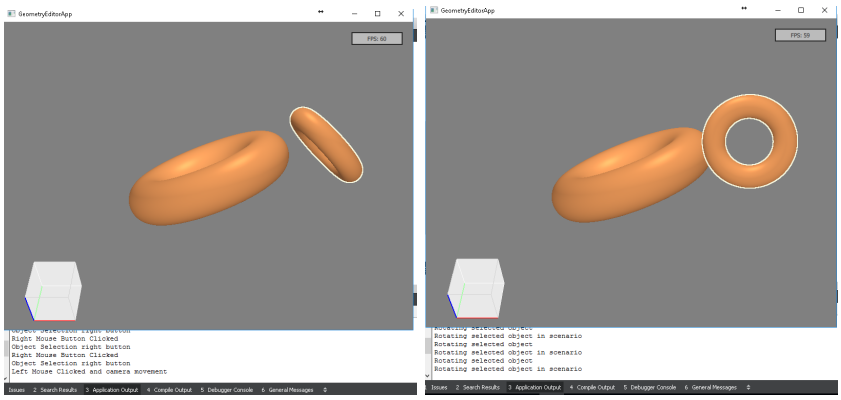

Figure 6: Demonstration of the Application zoom in zoom out of object

```
//move camera
void Scenario::moveCamera(const QPoint& c, const QPoint& p) {

    auto previous       = convertQtToGMlibViewPoint(*_camera.get(), p);
    auto current        = convertQtToGMlibViewPoint(*_camera.get(), c);

    auto tmp = previous - current;
    GMlib::Vector<float,2> d (tmp(0),-tmp(1) ); // "inverted y"
    d = d * 0.0001;

    _camera->move(d);
}
// Converts Qt Point to GMlib View
GMlib::Point<int, 2> Scenario::convertQtToGMlibViewPoint(const GMlib::Camera& cam, const QPoint& pos) {
    // Height of the camera's viewport
    int h = cam.getViewportH();

    // QPoint
    int q1 {pos.x()};
    int q2 {pos.y()};

    // GMlib Point
    int p1 = q1;
    int p2 = h - q2 - 1;

    return GMlib::Point<int, 2> {p1, p2};
}
```

Figure 7: Codes to handle camera movement

```
void Scenario::rotateSceneObject( const QPoint& c, const QPoint& p )
{
    stopSimulation();
    //if(Scenario::_sceneObj->isLocked()==true)
    //{
    if( _selectedObject and _objSelectedBool == true ) {

        auto previous       = convertQtToGMlibViewPoint(*_camera.get(), p);
        auto current        = convertQtToGMlibViewPoint(*_camera.get(), c);
        auto tmp = previous - current;
        // Compute rotation axis and angle in respect to the camera and view.
        const GMlib::UnitVector<float,3> rot_v =
                float( tmp(0) ) * _camera->getUp() -
                float( tmp(1) ) * _camera->getSide();
        const GMlib::Angle ang(
                    M_2PI * sqrt(
                        pow( double( tmp(0) ) / _camera->getViewportW(), 2 ) +
                        pow( double( tmp(1) ) / _camera->getViewportH(), 2 ) ) );
        //_sceneObj->editPos();
        _selectedObject->rotateGlobal( ang, rot_v);
        qDebug() <<"Rotating selected object in scenario";
    }

}
```

Figure 8: Codes to rotate scene object

```
// Object Selection
GMlib::SceneObject* Scenario::findSceneObject( const QPoint& pos ) {

    if(!_select_renderer)
        _select_renderer = std::make_shared<GMlib::DefaultSelectRenderer>();

    _select_renderer->setCamera(_camera.get());
    {
        GMlib::Vector<int,2> size { _viewport.width(), _viewport.height() };
        _select_renderer->reshape( size );
        _select_renderer->prepare();
        _select_renderer->select(-GMlib::GM_SO_TYPE_SELECTOR);

        auto coord = convertQtToGMlibViewPoint(*_camera, pos);

        //qDebug()<<"Type Id-> "<<_sceneObj->getTypeId();

        _sceneObj = _select_renderer->findObject( coord(0), coord(1) );

    }
    _select_renderer->releaseCamera();

    if( _sceneObj != 0) {

        return _sceneObj;
    }

    else return nullptr;
}
```

Figure 9: Codes to select object on the screen

```
void Scenario::setSelected( GMlib::SceneObject* _obj) { ...}

//object deletion concept is to hide object from the scene
void Scenario::deleteSelectedObject(GMlib::SceneObject *_obj)
{
    if( _obj ) {

        if( _objSelectedBool == true and _selectedObject and _setEditObjectBool==true) {

            //_sceneObj->remove(_obj);
            _sceneObj->setVisible(false,0);

            qDebug() <<"Selected Object removed";
            _setEditObjectBool=false;
        }
        else {

            qDebug () <<"Edit Mode not activated or object not selected";
        }
    }
}
```

Figure 10: Codes to delete object on the screen

5 Concluding remarks

Future enhancements may include array of object insertsion to the scene manipulation with ease and creating a live object like a car, snowman, with the objects added and various editing functions like scale, resize, delete, insert, with multiple type of object. The saving and loading of the scene objects is carried out with openddl data sctucture, where all the properties of the scene objects can be loaded to a structure and can be retrieved afterwards also it is a means of sharing files or objects created by other friends around, but need to use the same structure format.

References

[1] Addision Wesley. *A Tour of C++*. PEARSON, 1st edition, 2013.

[2] Witold Wysota and Lorenz Haas. *Game Programming Using Qt Beginner's Guide*. PACKT Publishing, 1st edition, 2016.