A Project Report on Applied Geometry and Special Effects

James Pandey

December 17, 2017

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 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. The functional attributes of curves and B-Spline Curves, and BSpline Surfaces created 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 0.6.9, and the Qt development suite. The basic setup template for the task was taken from gmlibqmldemo application, with the initial setup to work further on the task assigned and is tasked to implement our own curves and surfaces.

1 Introduction

The project was working created on C++ [2] and Qt [3] platform with modeling library GMlib. According to the assignment we have 6 set of tasks to work on, first with the model curve we just take an example for the model curve and try to draw our model curve in using the equations to draw our model curve. I used a model curve from the website of the University of Rhode Island, Department of Mathematics have developed [1] . After implementing the 1st task move on with the Bspline curve using the Bfunction. The model curve was then modified with Bfunction to create a model curve and then it was created as the Bspline curve generating the knot vectors of the curve. The same curve of two different example was created to implement the blending of two curves in task 3. In this the part of the 1st curve was blended with the part of the 2nd curve. For task

4 we have our own GERBS surface which take a model curve and number of the parts for the curve and then the curve is plotted in that amount of number. They can be individually modified and animated using some basic animation features I have just created a simple animation which rotates the curves parts in x,y and z direction. The 5th task was to work on with the surfaces as like curves where we have parts of the surfaces that can be modified individually same as curves we have sub-surfaces of the whole surface in plotted and with the control points of the parts of the surface it can be directed and modified. Task 6 is to work on with the animation of the project that it look good to view on.

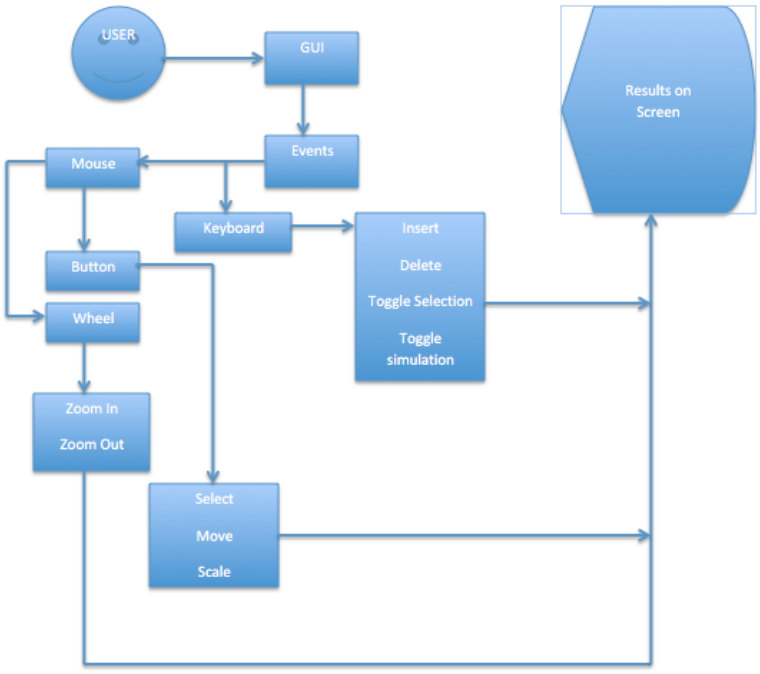

Figure 1: Demonstration the flow of the diagram

2 Task 1

Just with the quick start a model curve was taken and then a example file oc pcircle from GMlib was grabbed and then turned to plot our model curve with the equation taken from the example curve of [1] As it's an individual project the model curve we have taken is all different, needed should focus each and every aspect of the project. A figure below shows the parameters and the attributes of the model curve that I have chosen **??** fig. 3

3 Task 2

In task 2 we focused to make a B-spline curve from our model curve basically B-spline curve is a spline function which has knot vectors at euual distasnce and that it can be used for curve-fitting and numerical-differentiatin of data from experiments. The model curve from task 1 is taken and implemented for the model version of a 2nd degree B-spline curve the class usuage, PCurve class from GMlib as the base class. The evaluator only need to compute the value. And two constructor according to the assigned task is implemented.

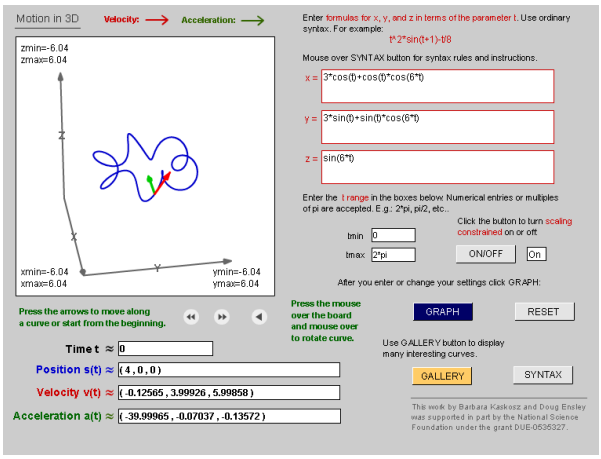

Figure 2: Attributes of the model curve

4 Task 3

Blending of two curves, the first part of a curve is blended with the part of the second curve, we have used the B-function which slighetly changes the original curve to blend it to another curve Bfunction only changes the value of the curves but it is approximately same as the first curve. Here the 1st and second curve is inserted as parameter and how much part want to be blended.

Implementation of the curve resulting from blending two curves with a B-function over a given part of the two curves for e.g. 0.3 of 1st curve and 0.7 of 2nd curve.

5 Task 4

In task 4 there is to be implmented own version of GERBS curve of the blending spline type where we this gerbs parts and animate them individually. For the subcurves of the model curve GMlib's PSubCurve class was used to make the local curves. And the same model curve from the 1st part was modeled as GERBS.

6 Task 5

As of in task 4 here we used surfaces to build the GERBS surface of blending spline type. But we have used SimpleSubSurf class externally provided to make it the more simpler version than the existing one in GMlib. We just calculated 1st derivative of the B-function for this. And it was done for both open and closed surfaces. A plane is an example of open surface while Cylinder is closed in one direction and open in other direction. Simple transformation of the surfaces

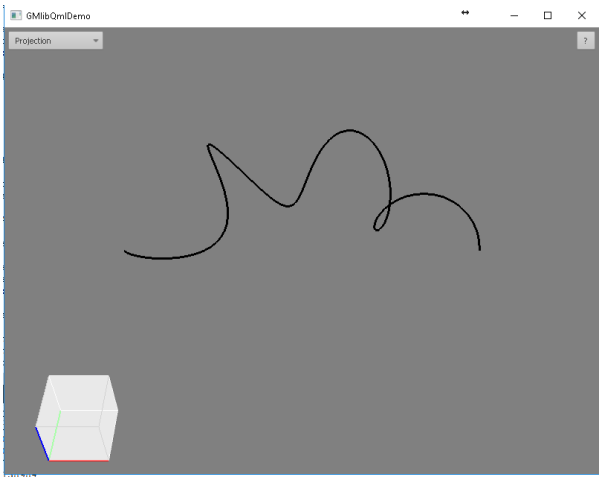

Figure 3: Curve plotted in the project as open curve from t=0 to i

is possible, where the control points like functions as replacers, small cube boxed on the surface is enabled when when we set the flag setCollapsed() to true for the sub-surfaces. We can select that receptor points and edit our local surfaces and they behave according to the changes.

7 Task 6

For the animation portion we can just rotate the parts of the curves and surfaces to give a feel like nice random movements. The part of the local curve and surface they have their own local axis and they can be manipulated within their axis, it just gives us the option to create animated object of the model. If there were no sub-surfaces or sub-curves then to animate them is not possible, if we apply rotation or translation to the model curve then it will rotate or translate around its own axis, and within it local boundary their is its sub-curves and surfaces which also rotates with it. But if we apply rotation and translation only to the sub surfaces then the whole model is fixed in the frame and the childs surfaces or the curves will rotate or translate.

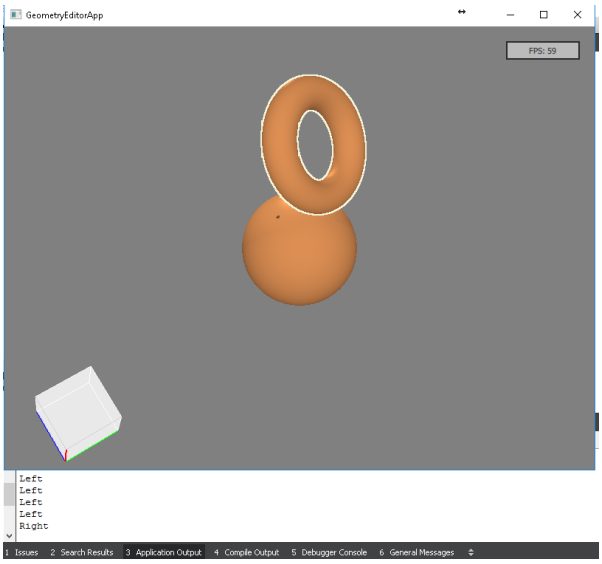

Figure 4: Demonstration of the Application insertion and deletion of the object

8 Conclusion

Each and every task are completed from task 1 to 6, works for both open and closed curves and open and closed surfaces with 1st derivative to replot surfaces. 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 surfaces added and various editing functions like scale, resize, delete, insert, with multiple type of object. A features of saving and loading of the curves and surfaces that are in the seen to the computer and then retrieving them back to scene will be so exciting for add-up.

References

[1] Motion in 3d. http://www.math.uri.edu/ bkaskosz/flashmo/as3/motion3d/motion3d.html. Accessed: 2017-11-14.

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

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

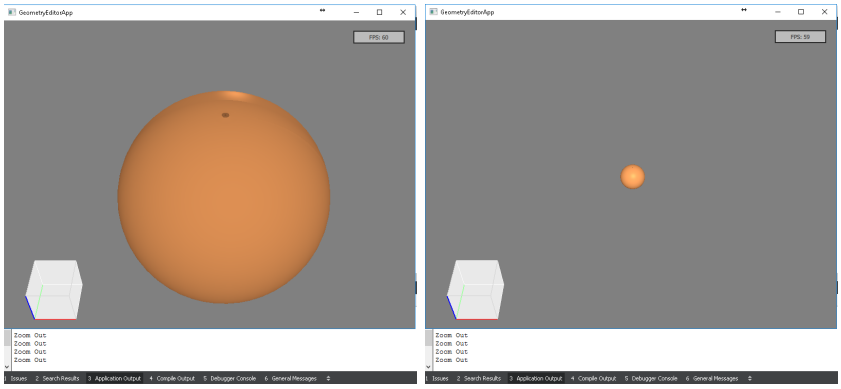

Figure 5: Demonstration of the Application zoom in zoom out of object

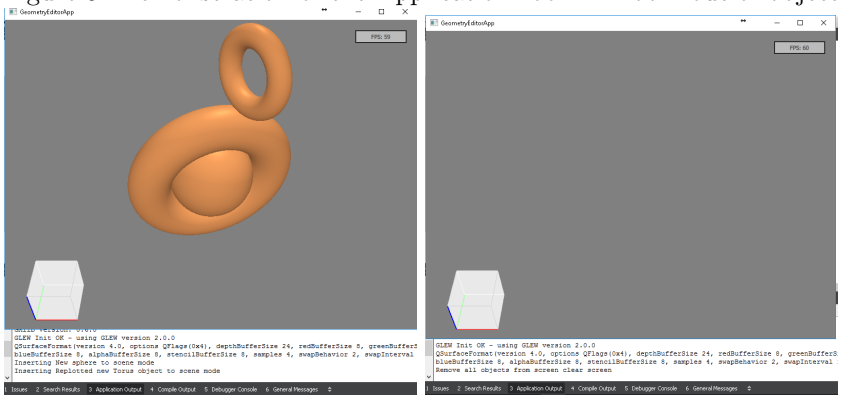

Figure 6: Demonstration of the Application inserting sphere and clearing screen

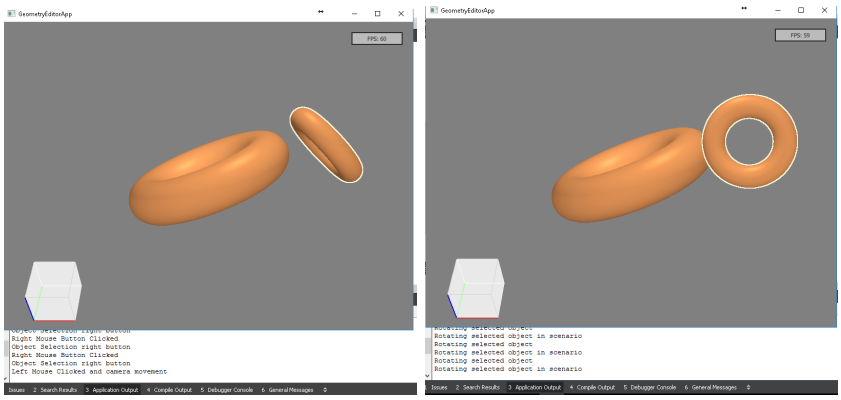

Figure 7: Demonstration of the Application zoom in zoom out of object

```
//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 8: 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 9: 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 10: 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 11: Codes to delete object on the screen