Florida International University
School of Computing and Information Sciences

CIS 4911 - Senior Capstone Project
Software Engineering Focus

Final Deliverable

Touch-midAir-Motion Gesture Input Framework
Team # X

Team Members
Ph.D. Francisco Ortega
Alfredo Zellek
Steven Ignetti

Product Owner: Ph. D. Francisco Ortega
Instructor: Masoud Sadjad

Copyright © and trademark notices, restrictions on copying or distributing the documentation, information for contacting the issuing organization (reader's comments), warranties, contractual obligations or disclaimers, and general warnings and cautions.

Abstract

In this document, we explore the designs of the improved multi-touch visualizer and the new gyroscope visualizer for the Touch mid-Air Motion framework. The sections to follow cover the system design of the visualizers, along with a description of the architectural patterns, and design patterns that were chosen. Diagrams compliment the described models and designs to aid in the understanding of the choices made. Testing methodologies were recorded and the important tests have been elaborated for the benefit of future developers

Furthermore, a breakdown of the user stories and task worked on for the visualizers is described in detail, along with information of the hardwares and softwares used. Great emphasis was made on the work done during the sprints, as well as the way work was divided for each sprint has been recorded, and the team's experiences, both good and bad, have been detailed.

Table of Contents

Introduction

This project will form the basis for an I/O device management framework, designed for developers interested in developing applications for devices outside of the standard mouse and keyboard. Our purpose will be to expand the existing visualizer to have more functionality with multi-touch and introduce a new device: the gyroscope.

Current System

The current system offers a simple touch-screen visualizer, which is an application that receives touch screen input and relays the information visually to the user, either by displaying it on-screen or printing the data in text. The visualizer is designed for developers interested in developing applications for touch-screen devices, and will be useful in gathering information on how the user interacts with the touch screen.

In the previous system's visualizer, it was capable of a few things: drawing the touched location on screen, recording the touch information to a comma-separated value (CSV) file, and replaying recorded touch-screen actions to the user. The visualizer also had extra features, such as changing the size of the brush during drawing, and displaying a console that gives immediate feedback on the actions being detected. The developers would be able to use this application to draw gestures in their touch-screen device and view the recorded results, allowing them to analyze this data for gesture processing. The current system's visualizer also has no framework available to it. This means that the whole of the visualizer is stand-alone.

There are many more limitations to the current system, starting with the constraint that it only serves as a recording utility for developers. Currently, the visualizer is not able to analyze the data in any way, and it is left to the developer to make use of the information. More importantly, the visualizer is limited to only gathering information from touch screen devices. Eventually, the TAM Framework will host a wide variety of I/O devices, and with the added range of devices that the framework will support, the visualizer will also need to take on its scope of devices as well.

Purpose of New System

The purpose of the new system is to give the developers an improved visualization system to use in developing for both multi-touch and gyroscope compatible applications. Developers will be able to use the visualizers to connect to touch screens and the YEI 3-Space Sensor input devices and test the usage of these devices immediately. This improved visualizer also has an adjacent windows to analyze and store the input data for the purpose of aiding developers in the debugging process. The visualizer was also built with the idea in mind that future implementations of this project will abstract these two input devices into a more general form, leading to the goal of a generalized input framework.

In more detail, the future of this generic API framework will deal with abstracting input devices. The API will detect the input devices on a lower level and determine the functionality of the device based on core functionalities, such as a two-dimensional position tracking device like a touch screen, or a three-dimensional position and acceleration tracking device like the YEI

3-Space sensor. This will allow the developer to connect and interact with a variety of devices directly into their application, and provide ways of manipulating and analyzing the data received by the input devices.

User Stories

This section will cover all of the user stories that were worked upon (but not necessarily completed). The main goal of these user stories is to create a more expansive visualizer for both the multi-touch and gyroscope systems. The majority of work that is show below reflects the development of both of these systems.

Implemented User Stories

User Story # 235 - Prepare Device SDK: Gyroscope
As a developer, I would like to be able to handle specific devices using the generic device framework, so that we can demonstrate the the capabilities of the framework. In this case, we can extend the framework to deal with gyroscopes.

User Story # 233 - Change Visual Representation
As a developer, I would like the display of the visualizer interface to show a different display setting, so that the display of fingers during recording and playback will be better represented. The visualizer should show a different visual representation of the finger pressing, such as circular displays or vertical / horizontal line traces.

User Story # 237 - Add New Visualizer File Format
As a product owner, I would like to change the current playback file format from CSV into a more universal format such as JSON, so that we have uniformity in the file formats.

User Story # 270 - Read Documents for Gyroscope
As a product owner, I would like my team to be well informed about gyroscope technology.

User Story # 257 - Improve to Real-time Playback
As a developer, I would like to adjust the playback functionality to play the recorded gestures in real-time, so that the traces are viewed as they were performed.

User Story # 272 - Convert Python Samples from Book to C++
As a developer, I would like to have gyroscope sample code, currently written in python, to be converted to C++, so that it is more efficient and can be combined with existing C++ games or applications.

User Story # 273 - Create Aviation Style Indicator
As a developer, I would like to have all the sensor information as an aviation style attitude indicator, so that I can determine movement of an object in real-time. It should take the pitch/roll and heading (yaw) from a single YEI 3-Space Sensor to render an aviation-style attitude

indicator display and heading indicator display. The attitude indicator should output in text in either continuous text or scrolling text or dashboard format. (Chapter 20, end of chapter)

User Story # 275 - Meet to Plan Sprint 2
As a product owner, I would like to provide user stories for Sprint 2, so that the work is planned for the sprint.

User Story # 276 - Modify JSON Format
As a developer, I want the JSON Format to be readable on an editor, so that it's easier to modify directly.

User Story # 278 - Create Circle-Connection on the Visualizer
As a developer, I would like to view additional primitive shapes (lines, dots) for the multi-touch drawing, so that the traces are better understood.

User Story # 279 - Read about Design Structural Patterns
As a product owner, I would like the team developer to have a proficiency in design structural pattern, so that the visualizer can abstracted for later re-use.

User Story # 280 & 281 - Write Documentation for Sprint 2 (Steven & Alfredo)
As a developer, I would like to continue working on the documentation for the project, so that we have a good reference material for the future.

User Story # 277 - Read about the Qt Threads
As a product owner, I would like the team developer to have a proficiency in C++ Qt threads, so that the visualizer component can work in parallel.

User Story # 258 - Integrate OpenGL into Visualizer
As a product owner, I would like OpenGL to be integrated into the existing visualizer, so that future releases can add more visually-stunning features and maybe even a 3-D visualizer.

User Story # 297 & 322 - Attend the Project Workshop (Alfredo & Steven)
As a product owner, I would like my team to be kept up to date with the technologies we're using in the project, so that we can continue our progress. This workshop will cover C++ 11 threads, templates, and project specifications that we did not cover in the previous workshop.

User Story # 317 - Implement Aviation-style Visualizer for Gyroscope
As a developer, I would like to have all the sensor information as an aviation style attitude indicator, so that I can determine movement of an object in real-time. It should take the pitch/roll and heading (yaw) from a single YEI 3-Space Sensor to render an aviation-style attitude indicator display and heading indicator display using Qt5.

User Story # 274 - Read about C++ Threading

As a product owner, I would like the team developer to have a proficiency in C++ 11 threads, so that the framework can function in parallel.

User Story # 320 & 375 - Meet to Plan Sprint 3 - Alfredo & Steven
As a product owner, I would like to provide user stories for Sprint 3, so that the work is planned for the sprint.

User Story # 330 - Integrate Gyroscope code into Aviation Dashboard Visualizer
As a developer, I would like the gyroscope code to be integrated into the Qt aviation dashboard visualizer, so that I can receive data from the gyroscope and render it onto the application.

User Story # 321 - Prepare Qt Threads
Prepare Qt threading for existing program, so that the user interface runs smoothly with the computationally-intensive calculations for the different display settings.
User Story # 323 / # 319- Write Documentation for Sprint 3 - (Steven & Alfredo)
As a developer, I would like to continue working on the documentation for the project, so that we have a good reference material for the future.
User Story # 338 - Fix Thread Display Error [Defect]
As a product owner, I would like the team member to fix the implementation that uses Qt threading, so that the user interface runs smoothly with the computationally-intensive calculations for the different display settings. The issue reported was that the thread did not display the results on the screen.

User Story # 339 / # 340 - Prepare Basic 3D Visualization for YEI Sensor (Alfredo & Steven)
As a developer, I would like to be able to visualize, in real-time, the YEI Sensor (or similar sensors that include gyroscope, accelerometer, and compass), so that it's easier to debug user interaction with the gyroscope.

User Story # 341 & 342 - Prepare Documentation (Steven & Alfredo)
As a team developer and developer, I would like to have well-documented product for major features, so that I can more easily modify the code.

User Story # 343 - Read about Qt / OpenGL
As a product owner, I would like the team developer to have a proficiency in OpenGL using Qt, so that the framework can have 3D visualizers.

User Story # 344 - Prepare a Basic Utility Window (Part I)
As a developer, I would like to have an additional window that provides log information and additional features outside of the main visualizing window, so that it is easier to debug gestures when using multiple monitors.

User Story # 362 - Fix Gyroscope Visualizer Portability Issues [Defect]

Upon trying to transfer the Gyroscope Visualizer to Alfredo’s laptop, there were a variety of issues that popped up. Drivers could not be installed for the YEI gyroscope, and the visualizer would crash (code was not altered).

User Story # 367 & 368 - Meet to Plan Sprint 4 - Alfredo & Steven
As a product owner, I would like to provide user stories for Sprint 4, so that the work is planned for the sprint.

User Story # 372 / # 371- Meet to Discuss Documentation - (Steven & Alfredo)
As a product owner, I would like to provide the team with help on the documentation, so that the future developers working on my project are able to reference the documentation.

Pending User Stories

User Story # 274 - Read About C++ Threading
As a product owner, I would like the team developer to have a proficiency in C++ 11 threads, so that the framework can function in parallel.

User Story # 234 - Create Generalization for Multi-Touch Visualizer
As a developer, I would like to work with C++ classes (and/or templates) for the multi-touch drawing, so that the classes can be extended and/or modified.

User Story # 232 - Add a Second Visualization Window
As a developer, I would like to have a second window for the visualizer, so that I can control the input and output display on the main visualizer window (usually from a secondary monitor).

User Story # 277 - Read about the Qt Threads
As a product owner, I would like the team developer to have a proficiency in C++ Qt threads, so that the visualizer component can work in parallel.

User Story # 231 - Implement Tasks
As a product owner, I would like to have Tasks implemented, so that the system is able to perform parallel computation.Tasks are asynchronous operations for parallel programming. This implementation should be similar to built in C# tasks library.

User Story # 236 - Implement a Specific Device - Leap Motion
As a product owner, I would like to be able to handle specific devices using the generic device framework, so that we can demonstrate the the capabilities of the framework. In this case, we can extend the multi-touch interface to deal with things like the LeapMotion.

User Story # 258 - Integrate OpenGL into Visualizer
As a product owner, I would like OpenGL to be integrated into the existing visualizer, so that future releases can add more visually-stunning features and maybe even a 3D visualizer.

User Story # 297 - Attend the Project Workshop
As a product owner, I would like my team to be kept up to date with the technologies we're using in the project, so that we can continue our progress. This workshop will cover C++ 11 threads, templates, and project specifications that we did not cover in the previous workshop.

User Story # 259 - Mouse Integration
As a product owner, I would like mouse integration on the visualizer, so that developers that don't have touchscreen devices can still utilize the visualizer for debugging gesture detection.

User Story # 172 - Implement Device: Touch Device
As a product owner, I want a Touch Device, so that it contains an Input Domain, Manipulation, Output Domain, States, and Events. This touch device shall utilize the system call from the windows interface and generate an input domain this shall be manipulated by the different parts outlined in the other user stories.

User Story # 190 - Implement Generic: Resolution Function
As a product owner, I want Resolution Functions, these are functions that will map the input to the desired output. This creates a mapping between input and output domains. For example, f(x) = x / 2 is a function that takes the original input and divides the input by 2. This can either be a function or a more advanced recognizer. The benefit to the developer is that it allows a very efficient and simple way to define the desired output with a function or if needed a more advanced recognizer.

User Story # 109 - Implement Generic: Connection
As a product owner, I want Connection, which is the ability to connect the device to different host operating systems. The benefit is that it will allow devices to be used with any os, if the connection code is provided.

User Story # 155 - Implement Generic device
As a product owner, I want a Generic Device, so that it contains an Input Domain, Manipulation, Output Domain, States, and Events.

User Story # 266 - Implement Device: Events
As a product owner, I want Events (signal), which are related to the states, use a subscription model to alert other objects that an event happened. For example, if the down state happened, other objects subscribed to this event will receive a callback. The benefit is that it provides developers with event callbacks for multiple objects. This should use a modified observer pattern for the events.

User Story # ??? - Meet to Plan Sprint 5
As a product owner, I would like to provide user stories for Sprint 5, so that the work is planned for the sprint.

User Story # ??? - Write Documentation for Sprint 5
As a developer, I would like to continue working on the documentation for the project, so that we have a good reference material for the future.

Project Plan

Hardware and software resources are used to help develop the current TAM system. The Hardware and Software section describes the individual hardware and software components and provides a brief explanation as to why the respective hardware or software was chosen. The section thereafter details the work that has been accomplished in each of the sprints. The sprints are divided into several user stories that describe the completed tasks, which are then further divided into subtasks. Each user story in the section also includes the acceptance criteria (the specifications that had to be met in order to declare the task complete), and the modeling (a visual representation of the user story's functionality).

Hardware and Software Resources

Hardware Resources

- Macbook Pro, Early 2011
  - Using the Macbook Pro, arbitrarily, for Steven to program the Visualizer

- Acer 21" Touch Screen Display
  - The touch screen is necessary to test the visualizer application. The visualizer is meant to be used by touch screen devices, and so this touch screen provides the best means of testing the application.

- Mini-Display to VGA Adapter
  - This is one of the two devices needed to connect to the touch screen.

- VGA to HDMI Adapter
  - This is one of the two devices needed to connect to the touch screen.

- Multi-touch Dell Laptop
  - Alfredo's hardware for working on the API framework and input devices.

- YEI 3-space Gyroscope Sensor with Wireless Dongle
  - One of the main input devices being worked with for this version of the TAM project. The hardware includes a sensor with compass, accelerometer, and magnetometer.

Software Resources

- Windows 7/8.1 (OS)
    - Windows 8.1 is necessary for hosting Visual Studio 2013, a necessary IDE in programming the TAM project, and only available for Windows.

- Visual Studio 2013 (IDE)
    - A necessary IDE. It was used in the previous system to program the visualizer.

- C++ Programming Language
    - The language of choice by Francisco in the previous system's work and as such is the language for the new system.. C++ was chosen for its efficiency in computer graphics programming.

- Qt Framework
    - Used solely for its capability of working with touch screen devices, as well as its ease in creating an interactive visual window to use in all operating systems.

- OpenGL Framework
    - Will be used in future implementations to visualize (in 3D) input devices that cannot normally be displayed in 2D.

- YEI 3-space sensor API
    - The API necessary to connect and extract data from the YEI 3-space gyroscope.

Sprints Plan

Sprint 1

(05/18/2015 - 05/29/2015)

User Story # 235 - Prepare Device SDK - Gyroscope

Tasks

- Install YEI 3-space sensor (gyroscope) and wireless dongle
- Create test program for gyroscope

Acceptance Criteria

- The device has been Installed and the SDK prepared

User Story # 233 - Change Visual Representation

Tasks

- No subtasks

Acceptance Criteria

- The visualizer displays a different visual representation other than the current finger-painting.

User Story # 237 - Add New Visualizer File Format

Tasks

- Write Code for JSON Implementation in Visualizer
- Research on JSON + Qt Implementation

Acceptance Criteria

- The visualizer can use the reading and writing of the new format for playback as intended.
- The new file format class has been tested to run without issues.

Sprint 2

(05/30/2015 - 06/12/2015)

User Story # 272 - Convert Python Sample Code from Book to C++

Tasks
- Modify Gyroscope-to-Pointer Interface
- Convert Device Streaming Code
- Convert Gyroscope-to-Pointer Interface code
- Convert Navigation and Compassing Code
- Convert Sensor Connection Code

Acceptance Criteria
- Python code converted to C++

User Story # 273 - Create Aviation Style Indicator

Tasks
- Develop Aviation Style Indicator
- Develop Heading Indicator
- Develop Attitude Indicator
- Develop Pitch, Roll, and Yaw Data Collection

Acceptance Criteria
- Obtain rotation, translation, and compass information in all formats (filtered and raw)
- Display (text) rotation, translation, and compass information in all formats (filtered and raw)

User Story # 257 - Improve to Real-time Playback

Tasks
- Understand Previous Code
- Rewrite Visual Playback
- Test the New Playback Functionality

Acceptance Criteria
- The visualizer implements the real-time playback.

- The visualizer (testing only) displays the difference of the initial and final time of the drawing.
- The difference of the initial and final time of the recording should be on the JSON File.
- The difference of time between the original recorded gesture and the playback gesture may have a small time difference.

User Story # 275 - Meet to Plan Sprint 2

Tasks

- Meet with Francisco
- Meet with Francisco for Show and Tell

Acceptance Criteria

- Product owner and team members (Alfredo and Steven) complete planning.

User Story # 276 - Modify JSON Format

Tasks

- Research and Utilize Qt Code to Change Output Format
- Test the JSON Implementation in Visualizer

Acceptance Criteria

- JSON File is well-formatted and reader is able to distinguish between the data-sets recorded.

User Story # 278 - Create Circle-Connection on the Visualizer

Tasks

- Create Averaging Logic for Finger Traces.
- Draw Connection of Fingers
- Add Drawing Functions

Acceptance Criteria

- Create a connection (e.g., circle, lines) between finger traces.
- Create a point (or circle) for the average of all the points being touched.

User Story # 279 - Read about Design (Structural) Patterns

Tasks

- Read Chapter 1
- Read Chapter 2
- Read Chapter 4

Acceptance Criteria

- Read chapter 1, 2, and 4 of “Design Patterns: Elements of Reusable Object-Oriented Software”.

User Story # 270 - Read Documents for Gyroscope

Tasks

- Read Chapter 20
- Read Chapter 26
- Read Gyroscope API documentation

Acceptance Criteria

- Read Chapter 20 from Interaction Design for 3D User Interfaces
- Read Chapter 26 (partial) from Interaction Design for 3D User Interfaces
- Review basic YEI gyroscope API

User Story # 280 - Write Documentation For Sprint 2 - Steven

Tasks

- Write Initial Documentation
- Write User Story Documentation

Acceptance Criteria

- Documentation for Sprint 2 covers all user stories worked on.

User Story # 281 - Write Documentation For Sprint 2 - Alfredo

Tasks

- Write Initial Documentation
- Write User Story Documentation

Acceptance Criteria

- Documentation for Sprint 2 covers all user stories worked on.

Sprint 3

(06/13/2015 - 06/26/2015)

User Story # 268 - Integrate OpenGL into Visualizer

Tasks

- Integrate OpenGL into existing code
- Replace QPaint calls with OpenGL in Visualizer

Acceptance Criteria

- OpenGL code is tested on the visualizer to assure that OpenGL features exist.
- Map the multi-touch input into the 3D environment of OpenGL to be displayed on the Visualizer

User Story # 317 - Implement Aviation-style Visualizer for Gyroscope

Tasks

- Develop Accelerator Gauge for Visualizer
- Develop Attitude Gauge for Visualizer
- Develop Heading Gauge for Visualizer
- Develop Qt GUI layout for Visualizer

Acceptance Criteria

- Working Attitude indicator
- Working Heading indicator

- Working Accelerometer indicator

User Story # 277 - Read About Qt Thread

Tasks

- Read about Qt 5 Threading
- Read Advanced Qt Programming
- Read about C++ GUI Programming with Qt 4

Acceptance Criteria

- Read chapter 14 from the “C++ GUI Programming with Qt 4".
- Read chapters 7 & 8 from “Advanced Qt Programming”.
- Read Web documentation on the qt.io site about Qt 5 Threading

User Story # 330 - Integrate Gyroscope code into Aviation Dashboard Visualizer

Tasks

- Integrate Threading into Gyroscope code
- integrate Gyroscope code into Qt

Acceptance Criteria

- Gyroscope connects to API from Qt dashboard
- Qt program receives data from the gyroscope
- Qt program displays data from the gyroscope

User Story # 323 / #319 - Write Documentation for Sprint 3 (Steven / Alfredo)

Tasks

- Write User Story Documentation

Acceptance Criteria

- Documentation for Sprint 3 covers all user stories worked on.

User Story # 321 - Implement Qt Thread

Tasks
- Integrate Threads into existing code
- Complete intense processes in threads
- Draw computed results from thread process

Acceptance Criteria
- The thread runs in the background without slowing down the main user interface
- The thread performs calculations for the main window to display in real-time

Modeling
- Figure 3 - ProcessorThread User Input Sequence Diagram
- Figure 4 - ProcessorThread Running Sequence Diagram
- Figure 5 - GLWindow Draw Screen Sequence Diagram
- Figure 6 - Setting Display Type Sequence Diagram

User Story # 274 - Read About C++ Threading

Tasks
- Read Chapter 18
- Read Chapter 41
- Read Chapter 42

Acceptance Criteria
- Read chapter 18 from the C++ Standard Library (By Josuttis)
- Read chapter 41 and 42 from Bjorne C++ Programming Language

User Story # 322 / 297 - Attend the Project Workshop (Alfredo / Steven)

Tasks
- Attend the Project Workshop

Acceptance Criteria
- Team members (Alfredo and Steven) show up to the meeting
- Team members(Alfredo and Steven) actively participate in the meeting.

Sprint 4

(06/27/2015 - 07/10/2015)

User Story #340 - Prepare Basic 3D Visualization for YEI Sensor (Alfredo)

Tasks

- Merge Gyroscope with 3D Visualizer Cod
- Create 3D Model for Visualizer

Acceptance Criteria

- Graphical 3D representation of the gyroscope

User Story # 339 - Prepare Basic 3D Visualization for YEI Sensor (Steven)

Tasks

- Display Secondary Window
- Make Secondary Window into Utility Window

Acceptance Criteria

- A second window appears with log information
- Text going to console will appear in the Utility Window

User Story # 343 - Read about Qt / OpenGL

Tasks

- Read Chapter 1 in OpenGL Superbible
- Read Chapter 2 in OpenGL Superbible
- Read Chapter 3 in OpenGL Superbible
- Read Qt OpenGL Documentation

Acceptance Criteria

- Read Qt OpenGL Documentation.
- Read Chapters 1 – 3 of OpenGL Superbible

User Story # 338 - Fix Thread Display Error [Defect]

Tasks

- Find / Fix the bug in threaded program

Acceptance Criteria

- The thread displays the results on screen.

User Story # 362 - Fix Gyroscope Visualizer Portability Issues [Defect]

Tasks

- Fix crashing gyroscope Visualizer
- Modify Installation and Project Settings
- Fix Gyroscope Drivers
- Find Proper Installation Setup for Future Project

Acceptance Criteria

- Gyroscope drivers installed and YEI sensor working.
- Qt Gyroscope Visualizer displays and works as intended.

User Story # 344 - Prepare a Basic Utility Window (Part I)

Tasks

- Display Secondary Window
- Make Secondary Window into Utility Window

Acceptance Criteria

- A second window appears with log information
- Text going to console will appear in the utility window

User Story # 367 / #368 - Meet to Plan Sprint 4 (Alfredo / Steven)

Tasks

- Meet with Francisco.

Acceptance Criteria

- Product owner and team members (Alfredo and Steven) complete planning.

User Story # 371 / # 372- Meet to Discuss Documentation (Alfredo / Steven)

Tasks

- Meet with Francisco

Acceptance Criteria

- Product owner and team members (Alfredo and Steven) complete planning.

User Story # 341- Prepare Documentation - Alfredo

Tasks

- Create Auxiliary Diagrams
- Write User Stories for Sprint 4
- Create Class Diagram of Current Code

Acceptance Criteria

- Create diagrams for major features implemented, which may include sequence diagram, class diagram, architecture diagram, and/or use cases

User Story # 342- Prepare Documentation - Steven

Tasks

- Create Sequence Diagrams
- Write User Stories for Sprint 4
- Create Class Diagram of Current Code

Acceptance Criteria

- Create diagrams for major features implemented, which may include sequence diagram, class diagram, architecture diagram, and/or use cases

System Design

System design defines the strategies taken by the developers to construct the desired project. Here we break up our design goals into the architectural patterns used, the system and subsystem decomposition models, and the design patterns. These make up a large part of the planning that was necessary for putting this project on course. An in-depth look at the choice's for this semester's design are looked at further in the upcoming subsections.

Architectural Patterns

For both the Multi-touch and Gyroscope visualizers, the Model-View-Controller pattern was used.
For these two visualizers, the view and controller and merged, because the user interface not only allows data to be displayed, but also lets the user choose the way the data is visualized on the user interface. We made the choice of the streaming input data to go into the model. This is because for both Touch and Gyroscope data, the system subscribes to the streaming input data.

System and Subsystem Decomposition

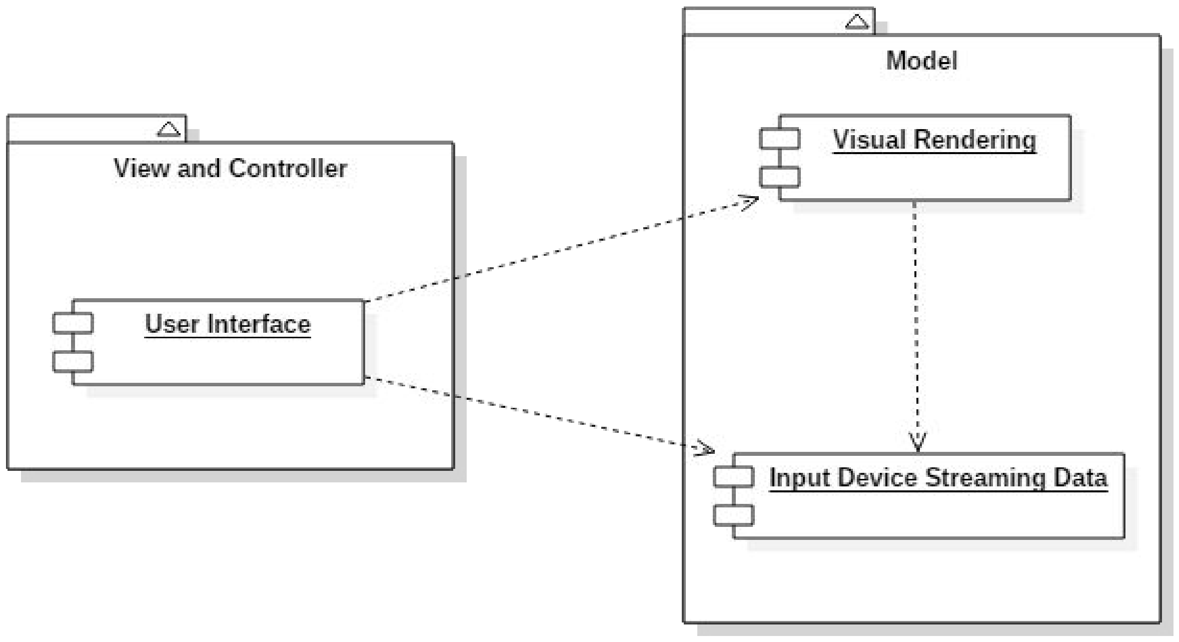


Figure 1 - Model-View-Controller Architecture

Above, in Figure 1, we have a component diagram detailing the subsystem decomposition in terms of the MVC architecture. For the sake of simplicity, the subsystems have been abstracted to encompass both the Touch and Gyroscope visualizers.

The User Interface subsystem covers all of the visual information that the user sees. This may be the touch points on the screen or the aviation style gauges for the gyroscope. The User Interface depends on the Visual Rendering subsystem that is present in both visualizers. This subsystem is in charge of using either the QPainter or OpenGL API’s to render the graphics that are seen by the user in the User Interface. The Visual Rendering of course requires data to display the information, and thus depends on the Input Device Streaming Data Subsystem, which tries to gather data from the devices (Touch, Gyroscope), manipulate the data, and emit the data to the appropriate rendering method in the appropriate class. Also the User Interface relies on the Input Device Streaming Data, since the User Interface may have a textual display (Gyroscope Visualizer) or a debug window(Touch Visualizer).

Deployment Diagram

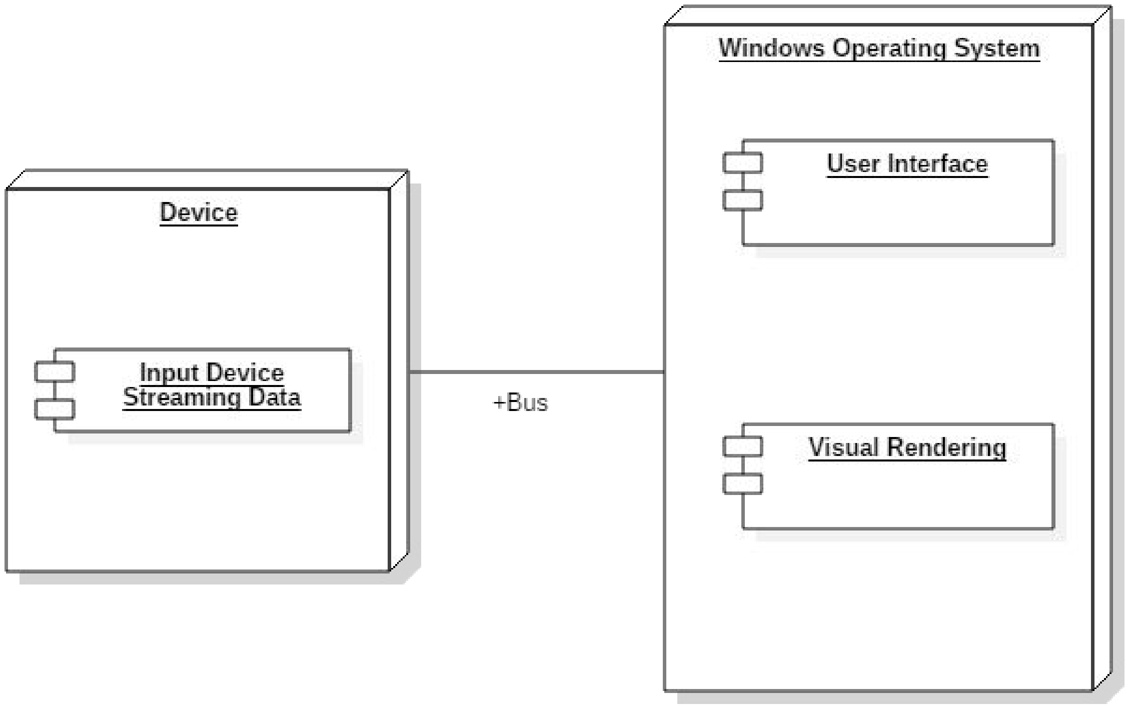


Figure 2 - Deployment Diagram

Design Patterns

Observer Pattern:
The Observer Pattern was chosen in order to effectively use multi-threading with the resource-intensive visualization code. With this pattern, we are able to have a thread collecting input data and doing complex manipulation of the data, and then emitting the data to the visualization software on a separate thread that has subscribed to the data collection thread. This is implemented in the Gyroscope Visualizer’s GyroThread, AviationDashboard, and GLWindow by using the Qt Signal/Slot methods that allow subscriptions to other threads.

Command Pattern:

The Command Pattern was chosen as a way to separate the process between the UI thread and the processor-thread. With the Command pattern, the processor-thread can be fed data (from the event of touching the screen, or moving the gyroscope, or any future input device) and perform the intense calculations that is needed to produce the results to display on screen, then send the commands to the UI thread on what to draw on the screen. Since the processor-thread will be sending the commands to the UI thread, it keeps the calculation process in a separate thread from the user interface process, allowing the UI to run seamlessly. The command is delivered via an overloaded method of the TAMShape interface called the draw() method. Depending on the TAMShape class delivered by the processor-thread, the UI thread will proceed to draw that given shape based on the specifications in the respective draw() method.

System Validation

Both visualizers used the Bottom-up approach to testing the software. The project leverages a lot of external APIs and all of these have been tested, hence, our testing was minimal. The classes were written and tested individually, and as the sprints went on, we integrated the already tested components with the newer components and performed testing on those. We also included unit-tests for the stories requiring intensive computation, since these are prone to error.

User Story # 237 - Add New Visualizer File Format

System Tests

- Test #001 - Open a JSON File
- Test #002 - Open a JSON File with garbage data
- Test #003 - Touch screen, press Save, save as new JSON File, open saved File
- Test #004 - Touch screen, press Save, save as existing JSON File, open saved File
- Test #005 - Press Save (w/o touching screen), save as new JSON File, open saved File
- Test #006 - Open a CSV File
- Test #007 - Open a CSV File with garbage data
- Test #008 - Touch screen, press Save, save as new CSV File, open saved File
- Test #009 - Touch screen, press Save, save as existing JSON File, open saved File
- Test #010 - Press Save (w/o touching screen), save as new JSON File, open saved File
- Test #011 - Open a file that is not JSON or CSV

User Story # 257 - Improve to Real-time Playback

System Tests

- Test #016 - Play metronome to 100 bpm, touch screen to the beat, press Save, open saved File

User Story # 278 - Create Circle-Connection on the Visualizer

System Tests

- Test #020 - getCircumcenter method test #1: Basic invalid input test
- Test #021 - getCircumcenter method test #2: Three same points
- Test #022 - getCircumcenter method test #2: Two same points
- Test #023 - getCircumcenter method test #2: Three points in a line

- Test #024 - getCircumcenter method test #2: Right triangle
- Test #025 - getCircumcenter method test #2: Equilateral Triangle
- Test #026 - getCircumcenter method test #2: Obtuse Triangle
- Test #027 - getCircumcenter method test #2: Acute Triangle

User Story # 258 - Integrate OpenGL into Visualizer

System Tests

- Test #035 - frustum() method test #1 : Basic invalid input test
- Test #036 - frustum() method test #2 : Near input checks
- Test #037 - frustum() method test #2 : Window width / height input test
- Test #038 - screenSize() method test #1 : Width / height input test
- Test #039 - Line Constructor method test #1 : Basic invalid input test
- Test #040 - Circle Constructor method test #1 : Basic invalid input test
- Test #041 - Finger Constructor method test #1 : Basic invalid input test
- Test #042 - SimpleCube Constructor method test #1 : Basic invalid input test
- Test #043 - Touch the screen, press Playback
- Test #044 - Touch the screen, press Clear Screen, press Playback
- Test #045 - Touch the screen, press Open, cancel, press Playback
- Test #046 - Touch the screen, resize window, touch the screen, press Playback
- Test #047 - Touch the screen, change Display, touch the screen, press Playback
- Test #048 - Touch the screen, change Brush size, touch the screen, press Playback
- Test #049 - Touch the screen rapidly, press Playback
- Test #050 - Touch the screen, change Brush Size, touch the screen
- Test #051 - Touch the screen, change Display, touch the screen
- Test #052 - press Playback, playback finished
- Test #053 - press Playback, playback finished, press Save, save to file
- Test #054 - press Playback, touch the screen, playback finished
- Test #055 - press Playback, minimize, maximize, playback finish
- Test #056 - press Playback, change window size, playback finished
- Test #057 - press Playback, move window, playback finished
- Test #058 - press Playback, exit application
- Test #059 - Touch the screen, press Clear Screen, press Playback
- Test #060 - Press multiple menu buttons at once (with multi-touch screen)
- Test #061 - Slow down computer performance, touch the screen, press Playback
- Test #062 - Slow down computer performance, press multiple menu buttons at once

User Story # 321 - Prepare Qt Threads

System Tests

- Test #070 - getShortestHamiltonianPath method test #1 : Empty input test

- Test #071 - getShortestHamiltonianPath method test #1 : Single input test
- Test #072 - getShortestHamiltonianPath method test #2 : Swap of vertices test
- Test #073 - getShortestHamiltonianPath method test #3 : 15 random vertices test

User Story # 273 - Create Aviation Style Indicator (Textual)

System Tests

- Test #080 - Connected Gyroscope and ensured proper connection through API calls
- Test #081 - With Gyroscope connected, ensured data is streaming onto the console.
- Test #082 - Verified Pitch, Roll, and Heading streaming data on console
- Test #083 - Verified Gyroscope streaming terminates when gyroscope is disconnected.

User Story # 317 - Implement Aviation-Style Visualizer for Gyroscope

System Tests

- Test #090 - Test pitch gauge with debug keyEvents (up and down)
- Test #091 - Test Roll gauge with debug keyEvents (left and right roll)
- Test #092 - Test Compass with debug keyEvents (rotating through all directions)
- Test #093 - Test accelerometer gauge with debug keyEvents (increase/decrease)

User Story # 330 - Integrate Gyroscope code into Aviation Dashboard Visualizer

System Tests

- Test #100 - Open Visualizer with Gyroscope connected to ensure proper connection
- Test #101 - Open Visualizer with Gyroscope not connected to ensure proper error message
- Test #102 - Open Visualizer with Gyroscope connected and verify data with debug console
- Test #103 - Open Visualizer, with Gyroscope connected, and then disconnect it while the program is still running to ensure that we get the proper error message.

User Story # 340 - Prepare Basic 3D Visualization for YEI Sensor

The Unit Tests for this user story verify the rotations for the 3D visualization of the gyroscope.

For the following Unit Tests, the initial position for all tests begin with the gyroscope facing the dongle and then rotating according to the orientation on the test.

System Tests (Unit Tests)

- Test #110 - Gyroscope laying flat without motion

- Test #111 - Gyroscope pointing upward
- Test #112 - Gyroscope pointing downward
- Test #113 - Gyroscope pointing right
- Test #114 - Gyroscope pointing left

Glossary

- **UI:** Short for “User Interface”. What enables the user to interact with the software in use.
- **TAM:** Short for “Touch-midAir-Motion”, the name of the project discussed in this document.
- **Qt:** A framework widely used for developing applications with graphical user interfaces.

Appendix

Appendix A - UML Diagrams

Static UML Diagrams

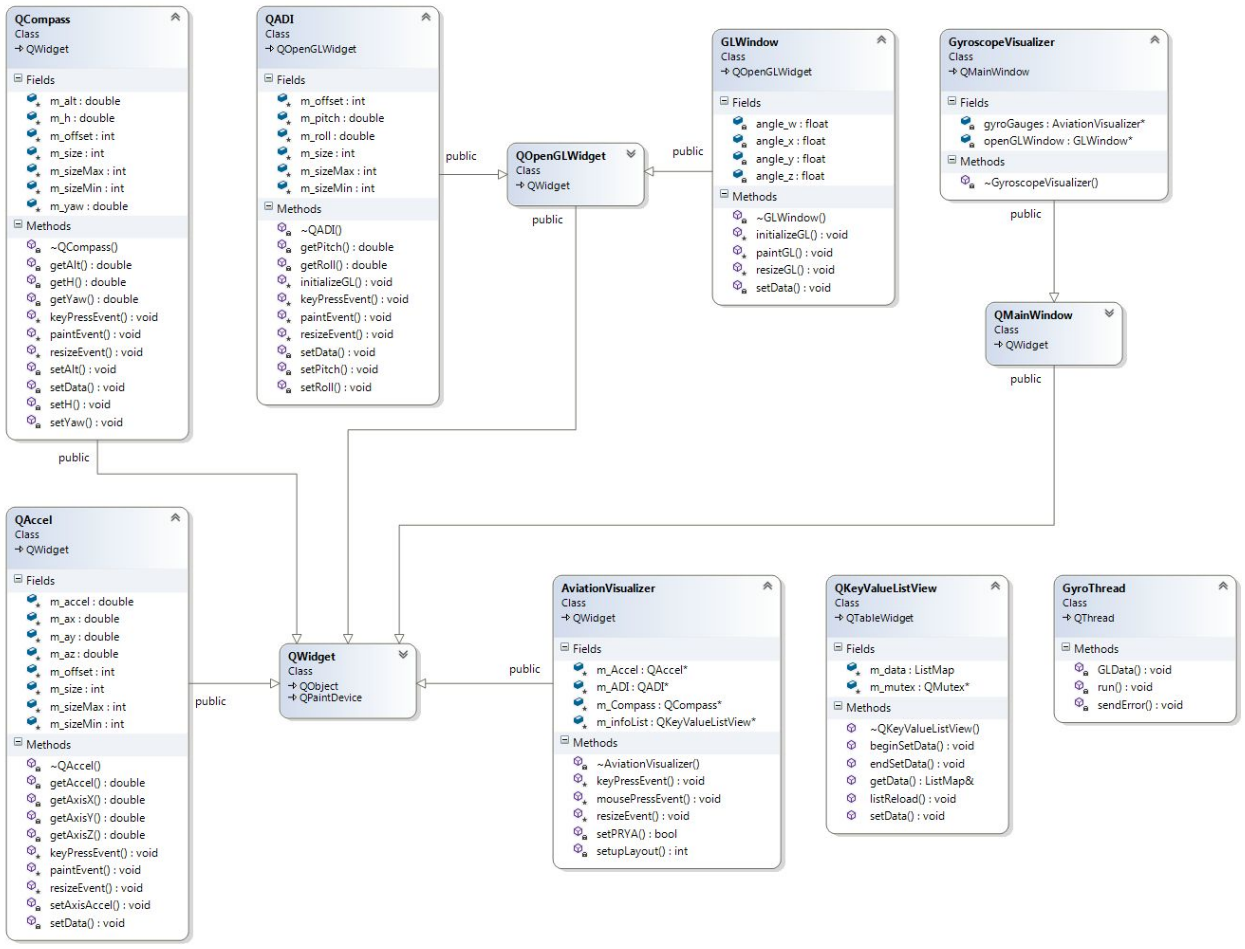


Figure 3 - Gyroscope Visualizer Class Diagram

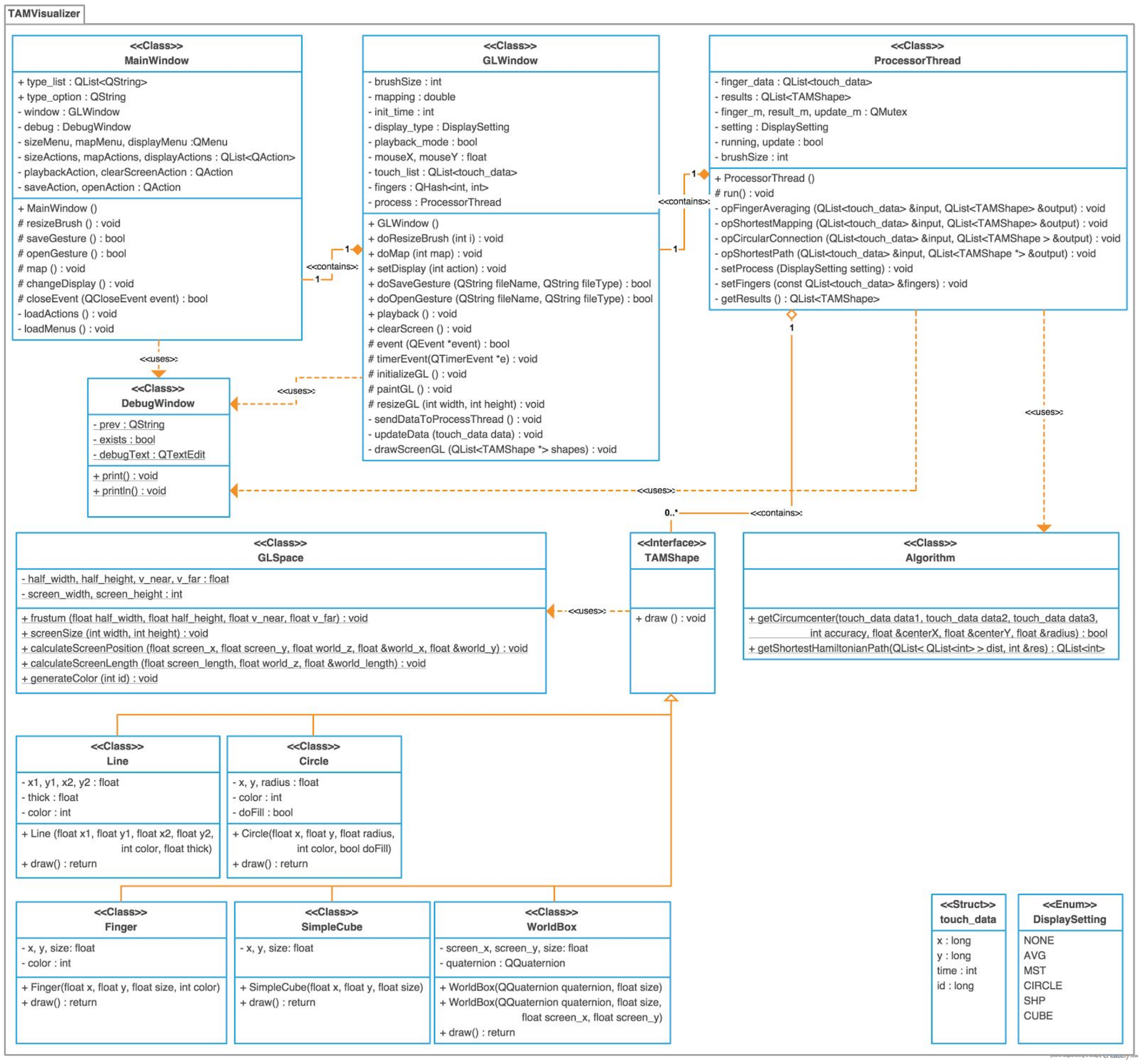

TAMVisualizer
<<Class>>
MainWindow
+ type_list : QList<QString>
+ type_option : QString
- window : GLWindow
- debug : DebugWindow
- sizeMenu, mapMenu, displayMenu :QMenu
- sizeActions, mapActions, displayActions : QList<QAction>
- playbackAction, clearScreenAction : QAction
- saveAction, openAction : QAction
+ MainWindow ()
# resizeBrush () : void
# saveGesture () : bool
# openGesture () : bool
# map () : void
# changeDisplay () : void
# closeEvent (QCloseEvent event) : bool
- loadActions () : void
- loadMenus () : void
<<Class>>
GLWindow
- brushSize : int
- mapping : double
- init_time : int
- display_type : DisplaySetting
- playback_mode : bool
- mouseX, mouseY : float
- touch_list : QList<touch_data>
- fingers : QHash<int, int>
- process : ProcessorThread
+ GLWindow ()
+ doResizeBrush (int i) : void
+ doMap (int map) : void
+ setDisplay (int action) : void
+ doSaveGesture (QString fileName, QString fileType) : bool
+ doOpenGesture (QString fileName, QString fileType) : bool
+ playback () : void
+ clearScreen () : void
# event (QEvent *event) : bool
# timerEvent(QTimerEvent *e) : void
# initializeGL () : void
# paintGL () : void
# resizeGL (int width, int height) : void
- sendDataToProcessThread () : void
- updateData (touch_data data) : void
- drawScreenGL (QList<TAMShape *> shapes) : void
<<Class>>
ProcessorThread
- finger_data : QList<touch_data>
- results : QList<TAMShape>
- finger_m, result_m, update_m : QMutex
- setting : DisplaySetting
- running, update : bool
- brushSize : int
+ ProcessorThread ()
# run() : void
- opFingerAveraging (QList<touch_data> &input, QList<TAMShape> &output) : void
- opShortestMapping (QList<touch_data> &input, QList<TAMShape> &output) : void
- opCircularConnection (QList<touch_data> &input, QList<TAMShape > &output) : void
- opShortestPath (QList<touch_data> &input, QList<TAMShape *> &output) : void
- setProcess (DisplaySetting setting) : void
- setFingers (const QList<touch_data> &fingers) : void
- getResults () : QList<TAMShape>
<<contains>>
1
1
<<contains>>
1
1
<<uses>>
<<uses>>
<<uses>>
<<uses>>
<<contains>>
1
0..*
<<Class>>
DebugWindow
- prev : QString
- exists : bool
- debugText : QTextEdit
+ print() : void
+ println() : void
<<Class>>
GLSpace
- half_width, half_height, v_near, v_far : float
- screen_width, screen_height : int
+ frustum (float half_width, float half_height, float v_near, float v_far) : void
+ screenSize (int width, int height) : void
+ calculateScreenPosition (float screen_x, float screen_y, float world_z, float &world_x, float &world_y) : void
+ calculateScreenLength (float screen_length, float world_z, float &world_length) : void
+ generateColor (int id) : void
<<uses>>
<<Interface>>
TAMShape
+ draw () : void
<<Class>>
Algorithm
+ getCircumcenter(touch_data data1, touch_data data2, touch_data data3,
int accuracy, float &centerX, float &centerY, float &radius) : bool
+ getShortestHamiltonianPath(QList< QList<int> > dist, int &res) : QList<int>
<<Class>>
Line
- x1, y1, x2, y2 : float
- thick : float
- color : int
+ Line (float x1, float y1, float x2, float y2,
int color, float thick)
+ draw() : return
<<Class>>
Circle
- x, y, radius : float
- color : int
- doFill : bool
+ Circle(float x, float y, float radius,
int color, bool doFill)
+ draw() : return
<<Class>>
Finger
- x, y, size: float
- color : int
+ Finger(float x, float y, float size, int color)
+ draw() : return
<<Class>>
SimpleCube
- x, y, size: float
+ SimpleCube(float x, float y, float size)
+ draw() : return
<<Class>>
WorldBox
- screen_x, screen_y, size: float
- quaternion : QQuaternion
+ WorldBox(QQuaternion quaternion, float size)
+ WorldBox(QQuaternion quaternion, float size,
float screen_x, float screen_y)
+ draw() : return
<<Struct>>
touch_data
x : long
y : long
time : int
id : long
<<Enum>>
DisplaySetting
NONE
AVG
MST
CIRCLE
SHP
CUBE

Figure 4 - TAM Visualizer Class Diagram

Dynamic UML Diagrams

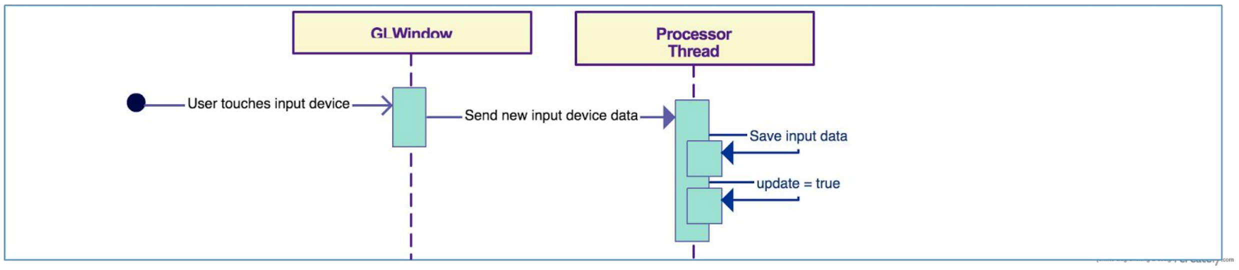


Figure 5 - ProcessorThread User Input Sequence Diagram for Multi-Touch

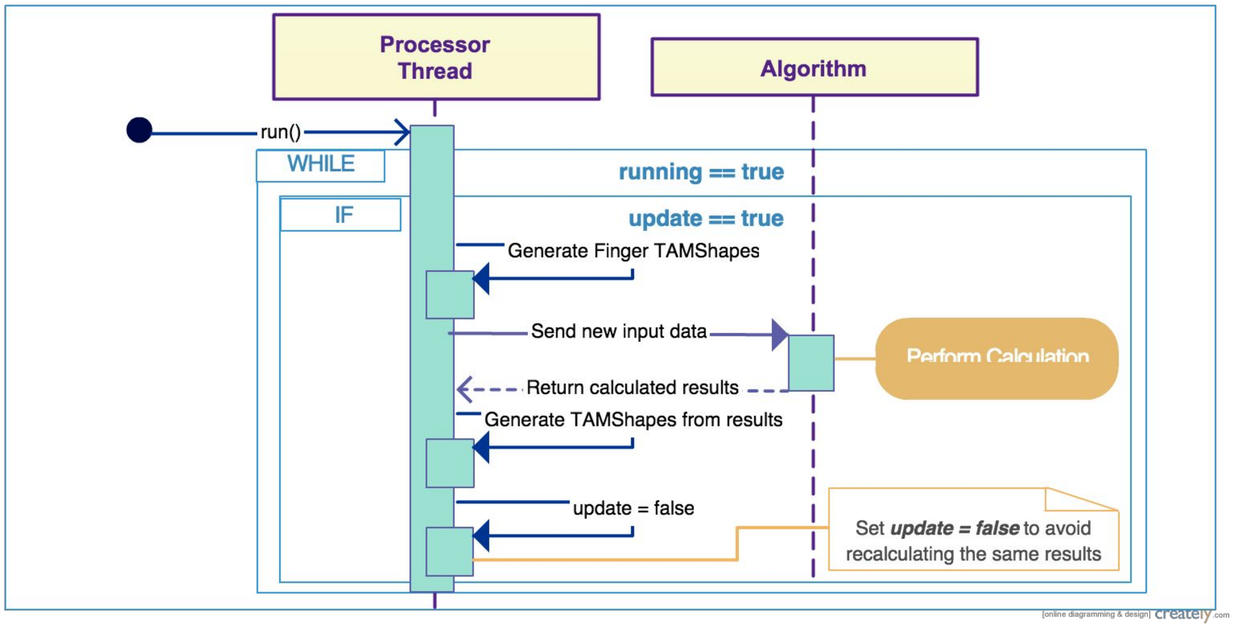

Processor Thread
Algorithm
run()
WHILE
running == true
IF
update == true
Generate Finger TAMShapes
Send new input data
Perform Calculation
Return calculated results
Generate TAMShapes from results
update = false
Set update = false to avoid recalculating the same results
[online diagramming & design] creately.com

Figure 6 - ProcessorThread Running Sequence Diagram for Multi-Touch

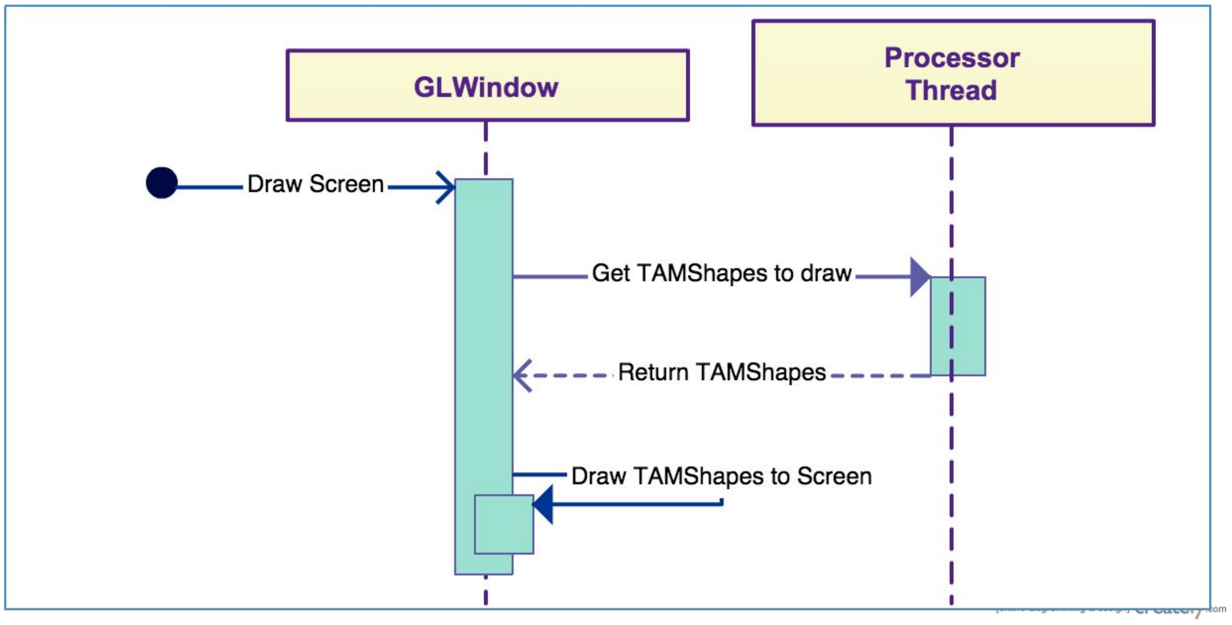


Figure 7 - GLWindow Draw Screen Sequence Diagram for Multi-Touch

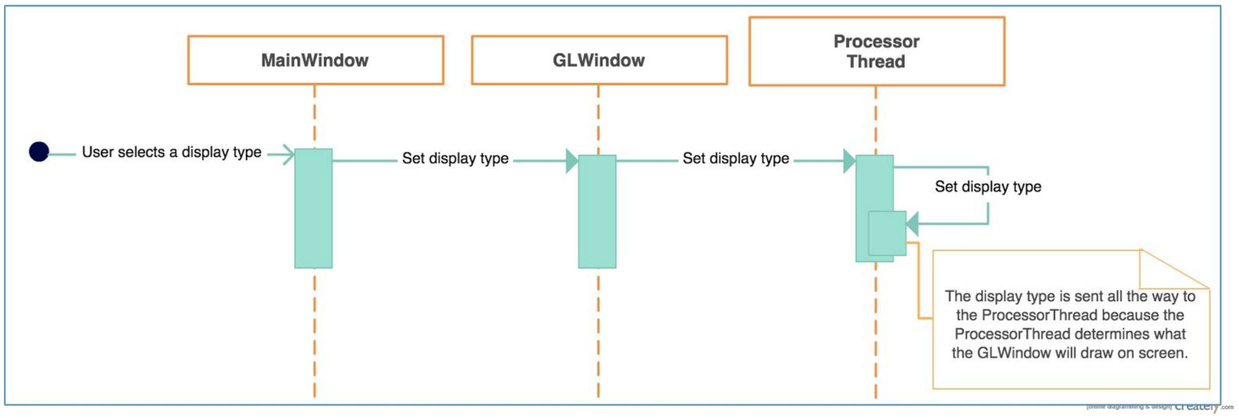
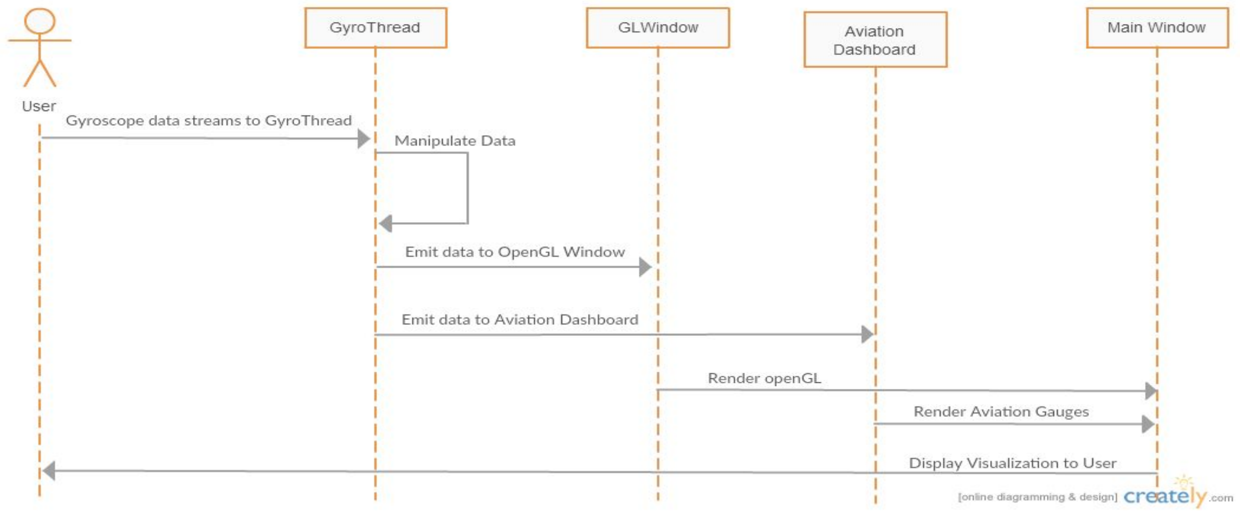
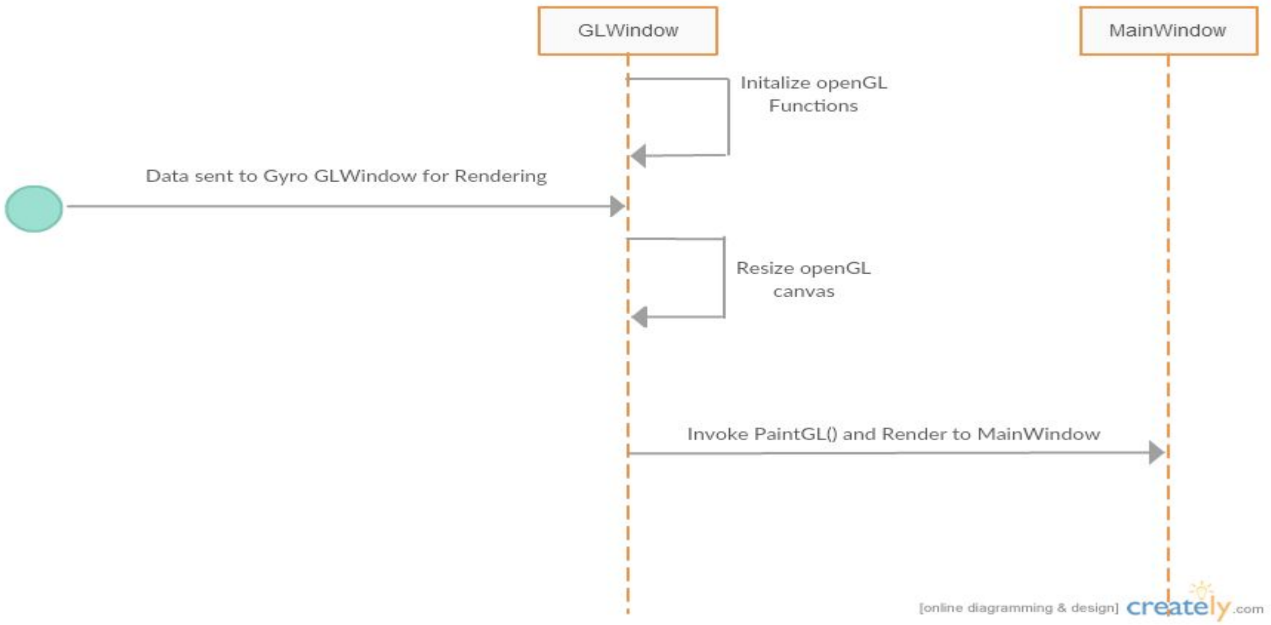

MainWindow
GLWindow
Processor Thread
User selects a display type
Set display type
Set display type
Set display type
The display type is sent all the way to the ProcessorThread because the ProcessorThread determines what the GLWindow will draw on screen.

Figure 8 - Setting Display Type Sequence Diagram for Multi-Touch



User
GyroThread
GLWindow
Aviation Dashboard
Main Window
Gyroscope data streams to GyroThread
Manipulate Data
Emit data to OpenGL Window
Emit data to Aviation Dashboard
Render openGL
Render Aviation Gauges
Display Visualization to User
[online diagramming & design] creately.com

Figure 9 - User Input Sequence Diagram for Gyroscope



GLWindow
MainWindow
Initalize openGL Functions
Data sent to Gyro GLWindow for Rendering
Resize openGL canvas
Invoke PaintGL() and Render to MainWindow
[online diagramming & design] creately.com

Figure 10 - GLWindow interaction for Gyroscope

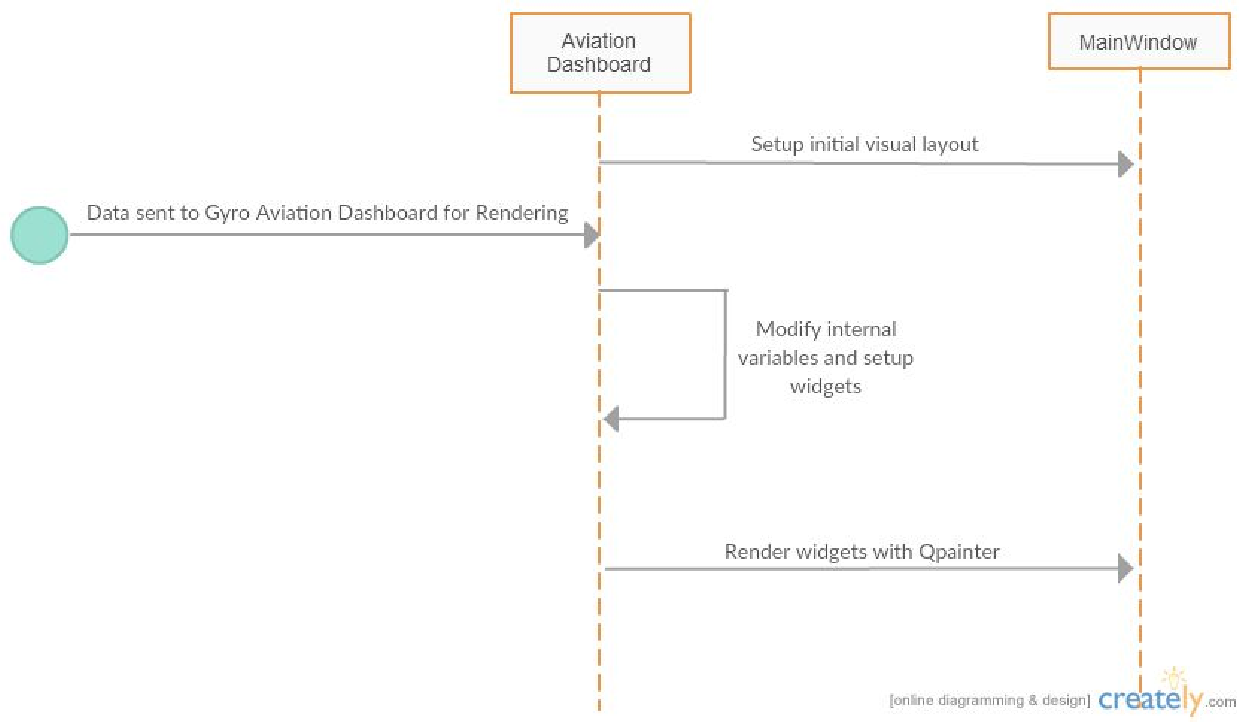
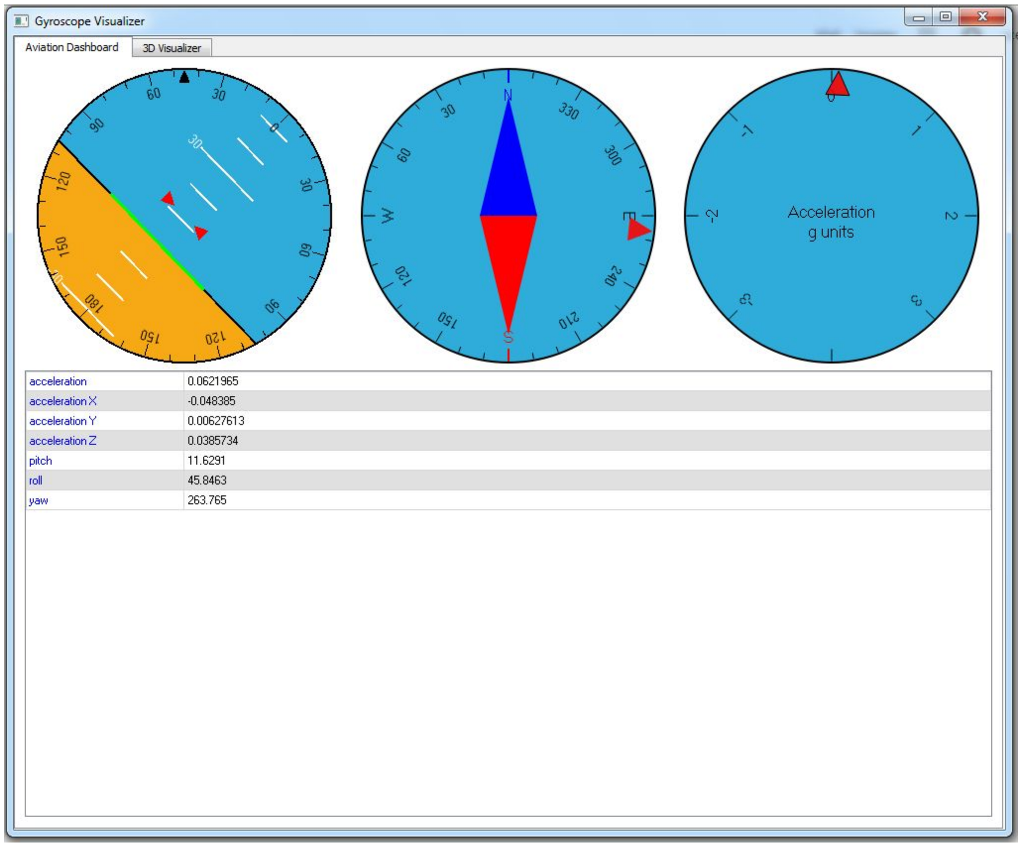

Aviation Dashboard
MainWindow
Setup initial visual layout
Data sent to Gyro Aviation Dashboard for Rendering
Modify internal variables and setup widgets
Render widgets with Qpainter
[online diagramming & design] creately.com

Figure 11 - Aviation Dashboard interaction for Gyroscope

Appendix B - User Interface Design



acceleration	0.0621965
acceleration X	-0.048385
acceleration Y	0.00627613
acceleration Z	0.0385734
pitch	11.6291
roll	45.8463
yaw	263.765

Figure 12 - TAM Gyroscope 2D Aviation Dashboard. Gauges from left to right, Pitch/Roll, Heading, Accelerometer.

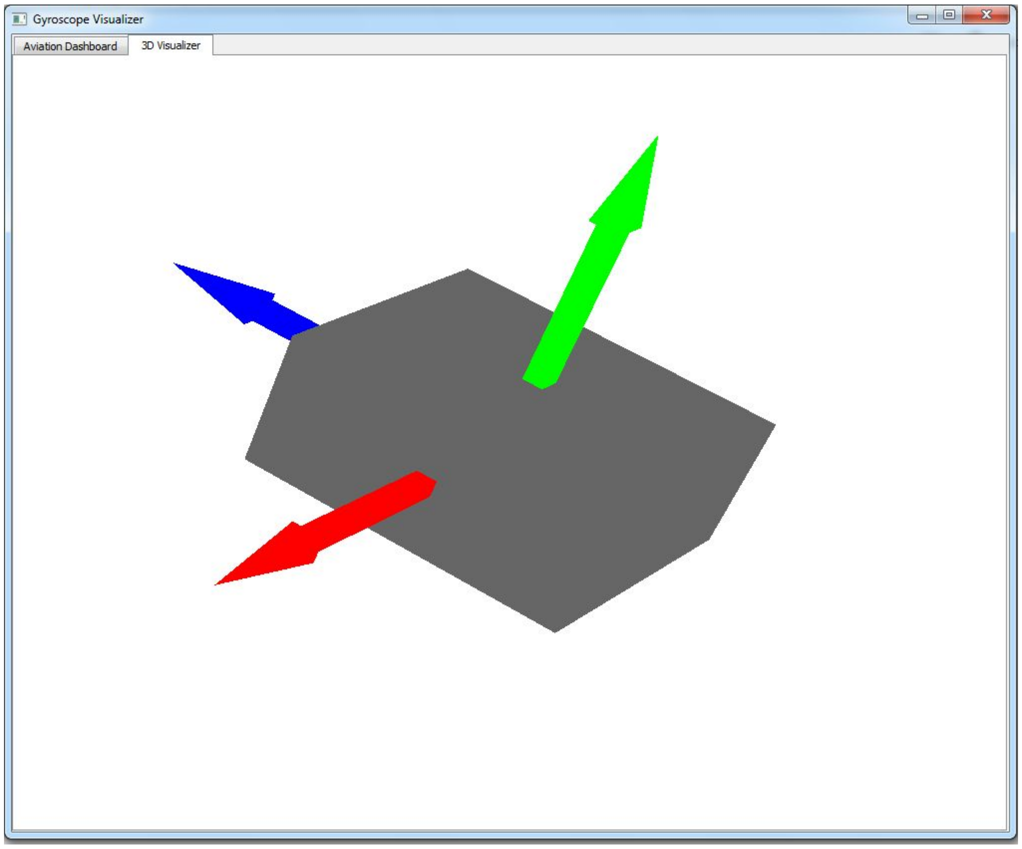


Figure 13 - TAM Gyroscope 3D Visualizer

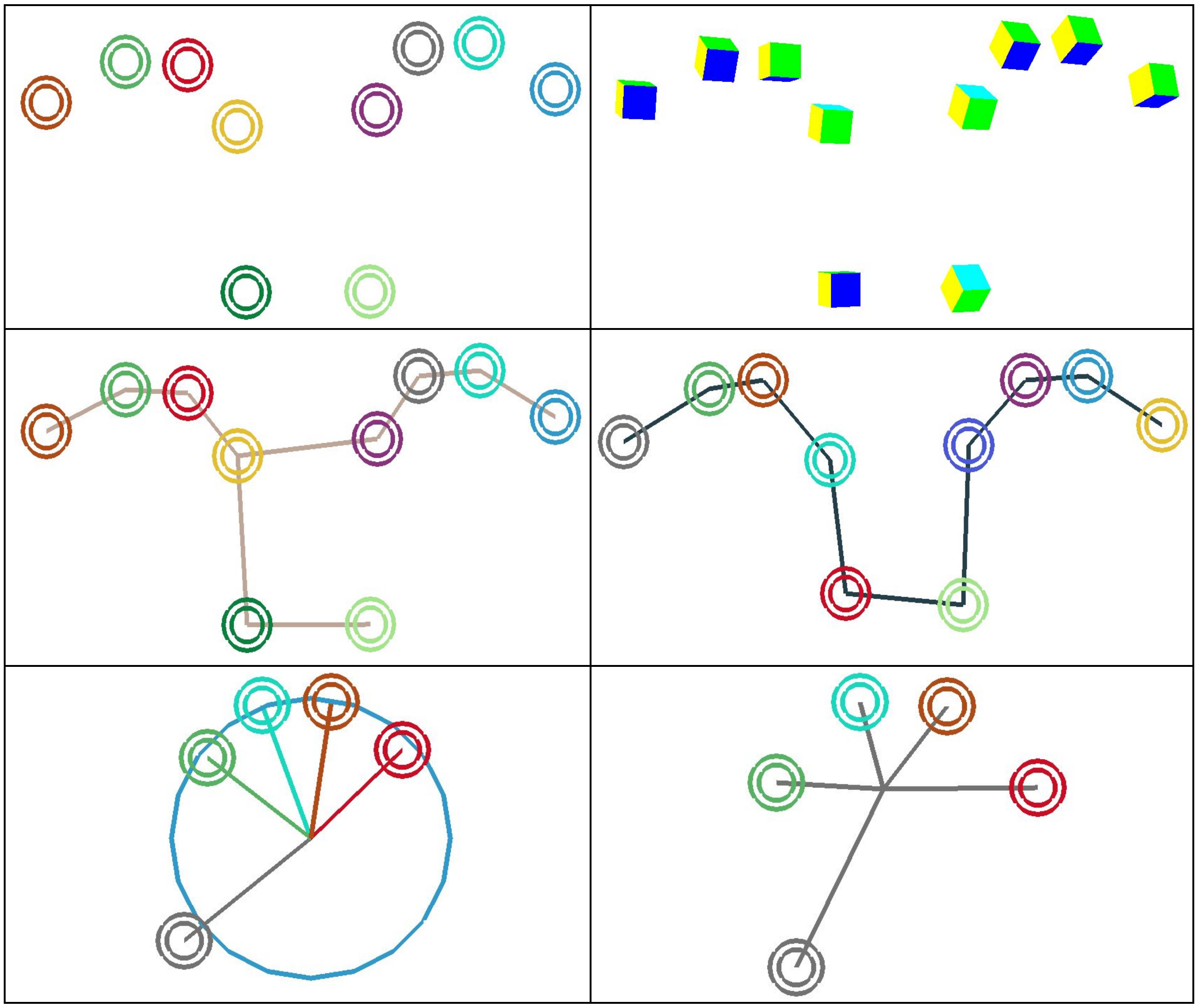


Figure # - The different display settings of the TAM Multi-Touch Visualizer

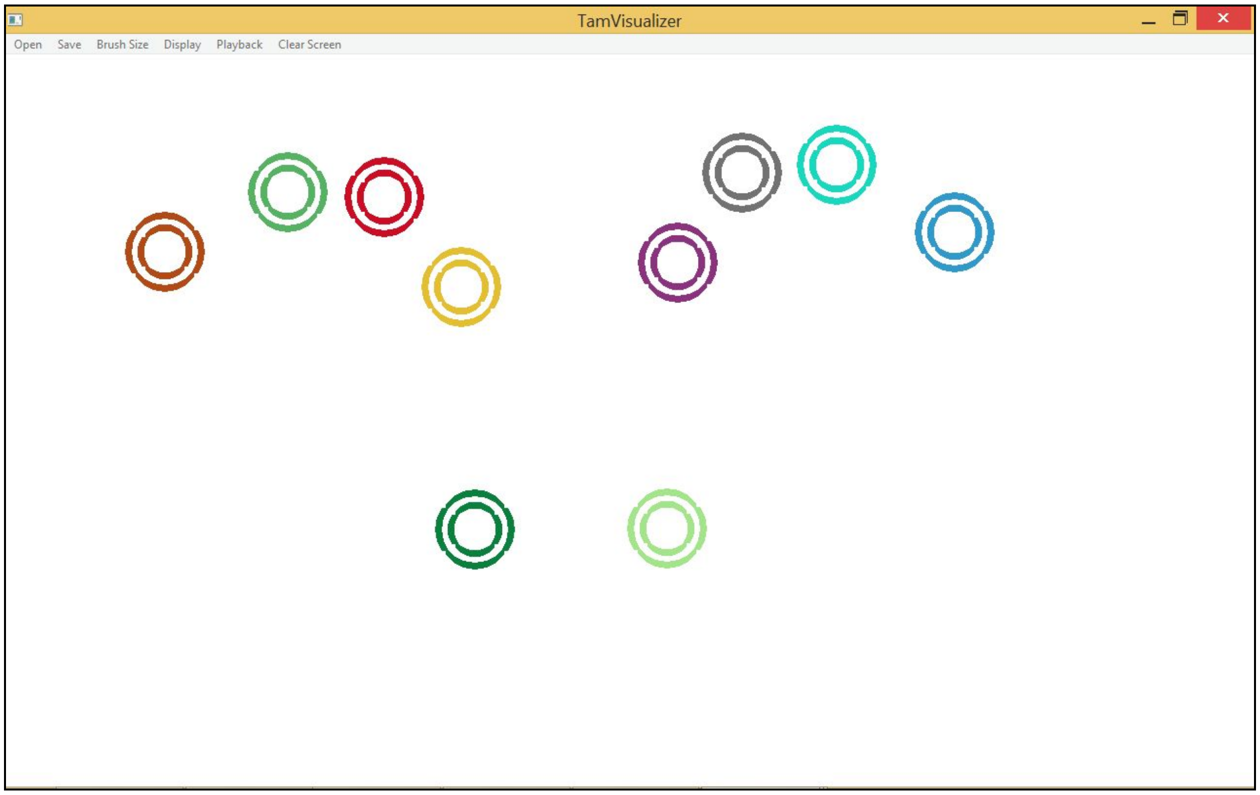


Figure 14 - The TAM Multi-Touch Visualizer

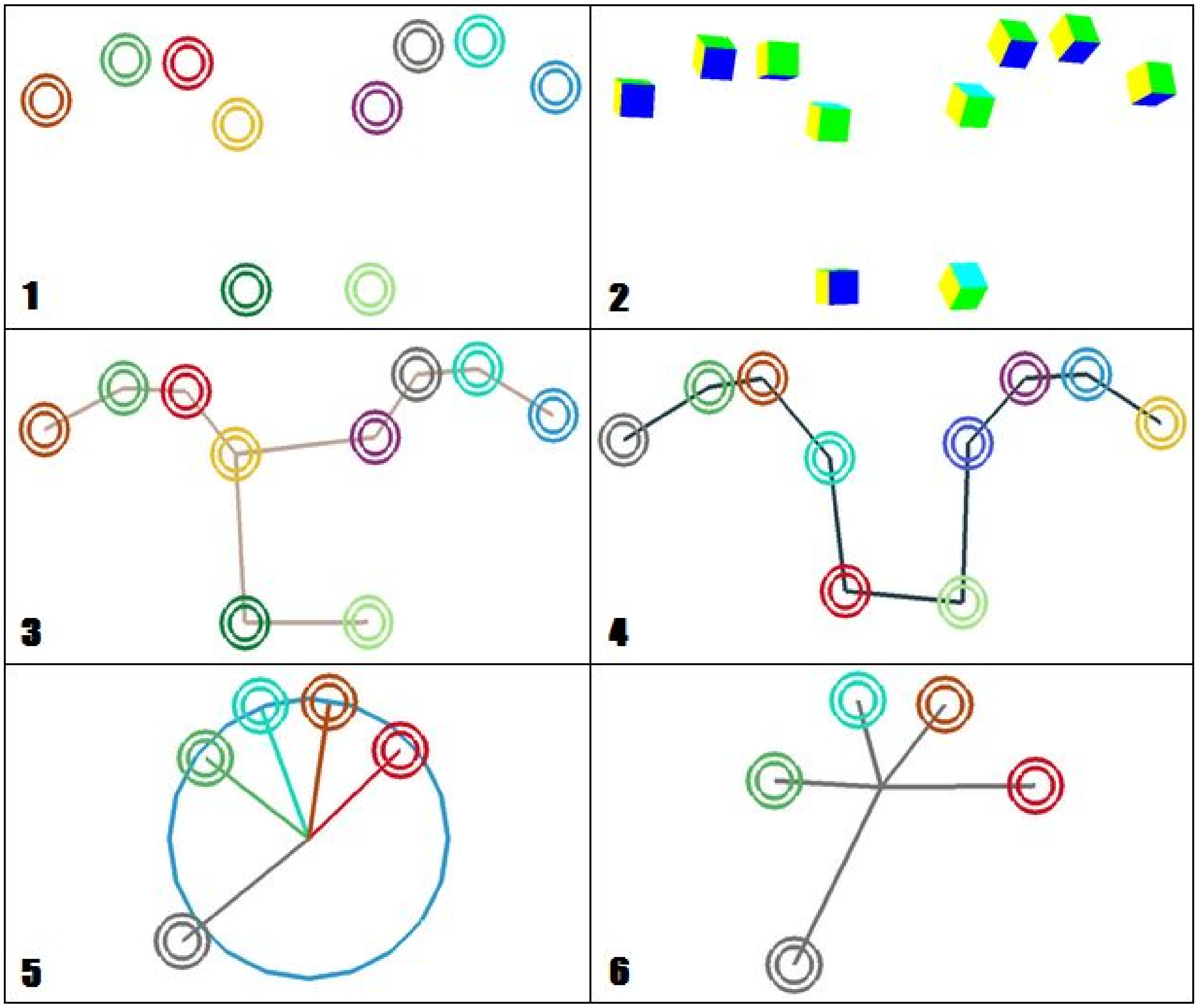


Figure 15 - The six different display settings of the TAM Visualizer: (1) No display, (2) OpenGL Cube, (3) Shortest Mapping, (4) Shortest Path, (5) Circular-Connection, (6) Finger Averaging.

Appendix C - Sprint Review Reports

Sprint 1 Report

Date: May 29, 2015
Attendees: Alfredo Zellek, Steven Ignetti, Francisco Ortega
Discussed Topics:

- Achieved:
    - Added a new visualizer format. (Steven)
    - Changed Visual Representation. (Steven)
    - Prepared Development Environment for Visualizer and Gyroscope (Alfredo)
- Not Achieved:
    - N/A
- Product Backlog Changes:
    - Split the JSON User Story into the portion of the user story that was completed and the additional feature that wanted to be implemented (Legible JSON format)

Sprint 2 Report

Date: June 12, 2015
Attendees: Alfredo Zellek, Steven Ignetti, Francisco Ortega
Discussed Topics:

- Achieved:
    - Added real-time playback to visualizer (Steven)
    - Made JSON Format legible (Steven)
    - Added new display types to visualizer, circular connection and finger averaging (Steven)
    - Converted Python code for gyroscope to C++ (Alfredo)
    - Created text-based aviation style indicator (Alfredo)
- Not Achieved:
    - N/A
- Product Backlog Changes:
    - Added new user stories which involved development of both team member's knowledge for the current sprint goals, such as reading, research, meetings with the product owner, and other miscellaneous activities not involved directly with the project. The following user stories were added:
        - Read Gyroscope Documentation (Alfredo)
        - Read Design / Structural Patterns (Steven)
        - Meet for Sprint 2 Planning
        - Write Documentation for Sprint 2
    - A few priority changes to the existing user stories to accommodate for the current and future sprint goals.

Sprint 3 Report

Date: June 26, 2015
Attendees: Alfredo Zellek, Steven Ignetti, Francisco Ortega
Discussed Topics:

- Achieved:
    - Integrated OpenGL into Visualizer, converting existing code into OpenGL code (Steven)
    - Added a separate thread to the visualizer (Steven)
    - Implemented 2D aviation-style visualizer for gyroscope (Alfredo)
    - Integrated gyroscope code from sprint 2 into aviation-style visualizer (Alfredo)
- Not Achieved:
    - Displaying thread data to the visualizer screen.
        - The reason this is incomplete was because of a few seemingly harmless OpenGL code that was added / modified, but not tested prior to the meeting.
- Product Backlog Changes:
    - Added new user stories which involved development of both team member's knowledge for the current sprint goals, such as reading, research, meetings with the product owner, and other miscellaneous activities not involved directly with the project. The following user stories were added:
        - Read and research about C++ Threading (Alfredo)
        - Read and research about Qt Threading
        - Write Documentation for Sprint 3
        - Added Sprint Planning Meeting with Francisco
        - Added C++ / Project Workshop Meeting

Sprint 4 Report

Date: July 10, 2015
Attendees: Alfredo Zellek, Steven Ignetti, Francisco Ortega
Discussed Topics:

- Achieved:
    - Introduced 3D Visualization through OpenGL into Gyroscope Visualizer (Alfredo, Steven)
    - Properly achieved multi-threading within the Gyroscope Visualizer (Alfredo)
    - Properly set up Gyroscope Visualizer environment to allow development portability (Alfredo)
        - Visual Studio paths were corrected and made general using environment Macros.
    - Created a debugging window to replace the windows console (Steven)
- Not Achieved:

        - Fully functional rotations due to Gimbal Lock in the gyroscope code.
- Product Backlog Changes:
    - Added new user stories which reflected problems encountered throughout the sprint.
        - Gyroscope Visualizer Environment had to be added mid sprint, since this was a major issue and needed immediate attention.
        - A visual bug was present in the multi-touch visualizer when adding a new thread to the environment. The bug was later fixed during this Spri
    - Added new user stories which involved development of both team member's knowledge for the current sprint goals, such as reading, research, meetings with the product owner, and other miscellaneous activities not involved directly with the project. The following user stories were added:
        - Read and research about OpenGL in Qt (Alfredo)
        - Write Documentation for Sprint 4
        - Added Sprint Planning Meeting with Francisco
        - Added Documentation Meeting with Francisco

Sprint 5 Report

Date: July 24, 2015
Attendees: Alfredo Zellek, Steven Ignetti, Francisco Ortega
Discussed Topics:

- Achieved:
    - Finalized the Multi-Touch Visualizer (Steven)
    - Finalized the Gyroscope Visualizer (Alfredo)
    - Wrote personal unit testing frameworks for the visualizers (Steven, Alfredo)
    - Completed documentation (Steven, Alfredo)
- Not Achieved:
    - N/A… yet...
- Product Backlog Changes:
    - Added user stories which involved development of both team member's knowledge for the current sprint goals, such as reading, research, meetings with the product owner, and other miscellaneous activities not involved directly with the project. The following user stories were added:
        - Finish Documentation
        - Complete Testing
        - Finalize Gyroscope Visualizer
        - Finalize Multi-touch Visualizer

Appendix D - Sprint Retrospective Reports

Sprint 1 Retrospective

Date: May 29, 2015
Attendees: Alfredo Zellek, Steven Ignetti, Francisco Ortega
Discussed Topics:

- Both demonstrations went well, no issues with displaying the work we have done.
- For now on, break down larger user stories into smaller user stories that can be completed during the course of the Sprint.
- For now on, work only on tasks that are "musts" during the course of the sprint.
- For now on, test the completed work for a user stories before confirming it as complete.

Sprint 2 Retrospective

Date: June 12, 2015
Attendees: Alfredo Zellek, Steven Ignetti, Francisco Ortega
Discussed Topics:

- Both demonstrations went well, no issues with displaying the work we have done.
- Learning the previous user code worked out well. Being the first major revision of code in the project, modifying the previous programmer's code to the desired outcome did not produce any major difficulties. (Steven)
- Building a new visualizer for the gyroscope from the start was a difficult task. It took a lot of reading, and messing around to get used to the API, and this should set the groundwork for the future gyroscope visualizer. (Alfredo)

Sprint 3 Retrospective

Date: June 26, 2015
Attendees: Alfredo Zellek, Steven Ignetti, Francisco Ortega
Discussed Topics:

- Demonstrations did not go so well, as a glitch disrupt the displaying of important results. (Steven)
- Second major revision of visualizer code went well. The major revision for OpenGL implementation resulted in a massive deletion of code from the previous programmer and many new classes to be implemented in its place. (Steven)
- Aviation Dashboard demonstration went well, no issues with displaying the work done. (Alfredo)
- Learning enough Qt to develop the dashboard took a lot of effort. This required a good ability to manipulate the QPainter object in order to design the aviation gauges. (Alfredo)

Sprint 4 Retrospective

Date: July 10, 2015
Attendees: Alfredo Zellek, Steven Ignetti, Francisco Ortega
Discussed Topics:

- Both demonstrations went well, no issues with displaying the work we have done.
- We should avoid falling behind on documentation, as we spent a lot of time catching up on it.
- Working together closely for the first time went decent. Encountered problems working out a schedule to meet up. Fortunately, we were able to merge the two projects.
- For now on, both of us need to test new features and document testing properly.
- From now on, properly set up development environment to allow portability (Alfredo).

Sprint 5 Retrospective

Date: July 24, 2015
Attendees: Alfredo Zellek, Steven Ignetti, Francisco Ortega
Discussed Topics:

- [To be discussed]

References

The Shortest Hamiltonian Path Algorithm, used to draw the shortest path between all fingers, was retrieved from the Google Source Code and converted to C++:

❏ https://sites.google.com/site/indy256/algo/shortest_hamiltonian_path

Credit to John Ryding (strife25 in github) for offering a solution to drawing circles in OpenGL:

❏ https://gist.github.com/strife25/803118

Sample code made by Sorj offered a template into how to design a Qt Application with an OpenGL environment. The code also provided OpenGL code for drawing a simple cube, which was later used to test the OpenGL feature integrated into the multi-touch screen visualizer for the first time:

❏ http://sorj.de/?p=192

qFlightInstruments provided by bushuhui. Allowed easy integration into the Qt project and visualizer. The flight instruments were modified and expanded upon.

❏ https://github.com/bushuhui/qFlightInstruments