Table of Contents

1 Introduction 2

1.1 Goals and Objectives 2

1.2 Statement of Scope 2

1.3 Software Context 2

1.4 Major Constraints 2

1.5 Definitions, Acronyms and Abbreviations 2

1.6 Document Description 2

2 Data Design 2

3 Architectural and Component Level Design 3

3.1 Overall Design 3

3.1.1 *MVC* Design Pattern 3

3.1.2 AppDelegate 3

3.2 Drawing Engine Design 3

3.2.1 Drawing Engine Structure 3

3.2.2 Drawing Engine Classes 3

3.2.3 Drawing Sequence 3

3.3 Gallery Design 3

3.3.1 Gallery Structure 3

3.3.2 Gallery Classes 4

3.4 Class Descriptions 4

4 User Interface Design 4

5 Security 4

6 Restriction, Limitations and Constraints 4

7 Testing Issues 4

8 Design Process 4

9 Conclusion 4

9.1 What Worked Well 5

9.2 Challenges 5

9.3 Lessons Learned 5

10 Requirements Traceability 5

10.1 Drawing Engine Requirements 5

10.1.1 External Interface Requirements 5

10.1.2 Functional Requirements 5

10.1.3 Performance Requirements 6

10.2 Gallery Requirements 6

10.2.1 External Interface Requirements 6

10.2.2 Functional Requirements 6

10.2.3 Performance Requirements 6

11 References 7

# Introduction

## Goals and Objectives

## Statement of Scope

## Software Context

## Major Constraints

TBD – Need to figure out if this section will be needed or if all constraints will be covered in the section about constraints.

## Definitions, Acronyms and Abbreviations

* *App* – Abbreviation for application.
* *App User* – Someone that uses IdeaStorm
* *IdeaStorm* – The application being developed here.
* *MVC* – Abbreviation for Model-View-Controller a design pattern that separates the user interface components from the data management components.
* *Point Sprite* – TBD
* *UI* – Abbreviation for User Interface.

## Document Description

TBD – Provide a description of the document layout and the

# Data Design

TBD – Need to cover the file structure, how files are saved and the classes used for this.

TBD – Move the explanation and diagram from 3.2.3 of the old document to here and reword to explain better.

TBD – Add an ERD diagram to show the structure and relationship of the data.

TBD – May also want to consider adding a sequence diagram to show how data is saved.

# Architectural and Component Level Design

## Overall Design

### *MVC* Design Pattern

*IdeaStorm* uses the *MVC* design pattern [Cocoa Core Dependencies] to separate the data and data management from the views that the *app user* interacts with. By using the MVC design pattern, *IdeaStorm* can have a centralize data management components (aka. the model) that can be accessed by all other components of the app. In IdeaStorm, the model’s main class is the Database.

The Database class is then accessed directly via two controller components, the GalleryViewController and the DrawingViewController. As the names suggest, these classes also directly manage the views that provide the *UI* for the *app user*.

### AppDelegate

The AppDelegate class is responsible for initializing and setting up the main classes in *IdeaStorm.* The AppDelegate performs the following steps when run:

1. Creates an instance of the Database class.
2. Creates an instance of the GalleryViewController and DrawingViewController classes.
   1. The Database instance is provided to these instances during initialization so each view controller has a reference to the same data component.
3. Provide the instance of the GalleryViewController with a reference to the instance of the DrawingViewController.
4. Sets the GalleryViewController instance as the root view controller for the *app.*
5. Makes the window (including the root view controller) visible.

## Drawing Engine Design

TBD – Will also need a section describing the tool sets.

### Drawing Engine Structure

TBD – Create a JSD describing the Drawing engine. Is this really the best way to describe this. There is a lot of depth in how everything is connected (DrawingViewController creates DrawingEngine nd the Toolbar, Drawing engine then creates the GLView and the ToolbarItems). This would be great for describing the tools actually. If nothing else this could map what creates what in the DrawingEngine. That would be useful on its own and would provide a better overall idea of the structure. It might not be a JSD/ESD though.

### Drawing Engine Classes

TBD – Create a class diagram showing all the class relationships

### Drawing Sequence

Figure 3‑1 describe the sequence of events that occurs when the *app user* draws using the *IdeaStorm* drawing engine. The DrawingTool is placed in angle brackets to represent that it is a protocol.

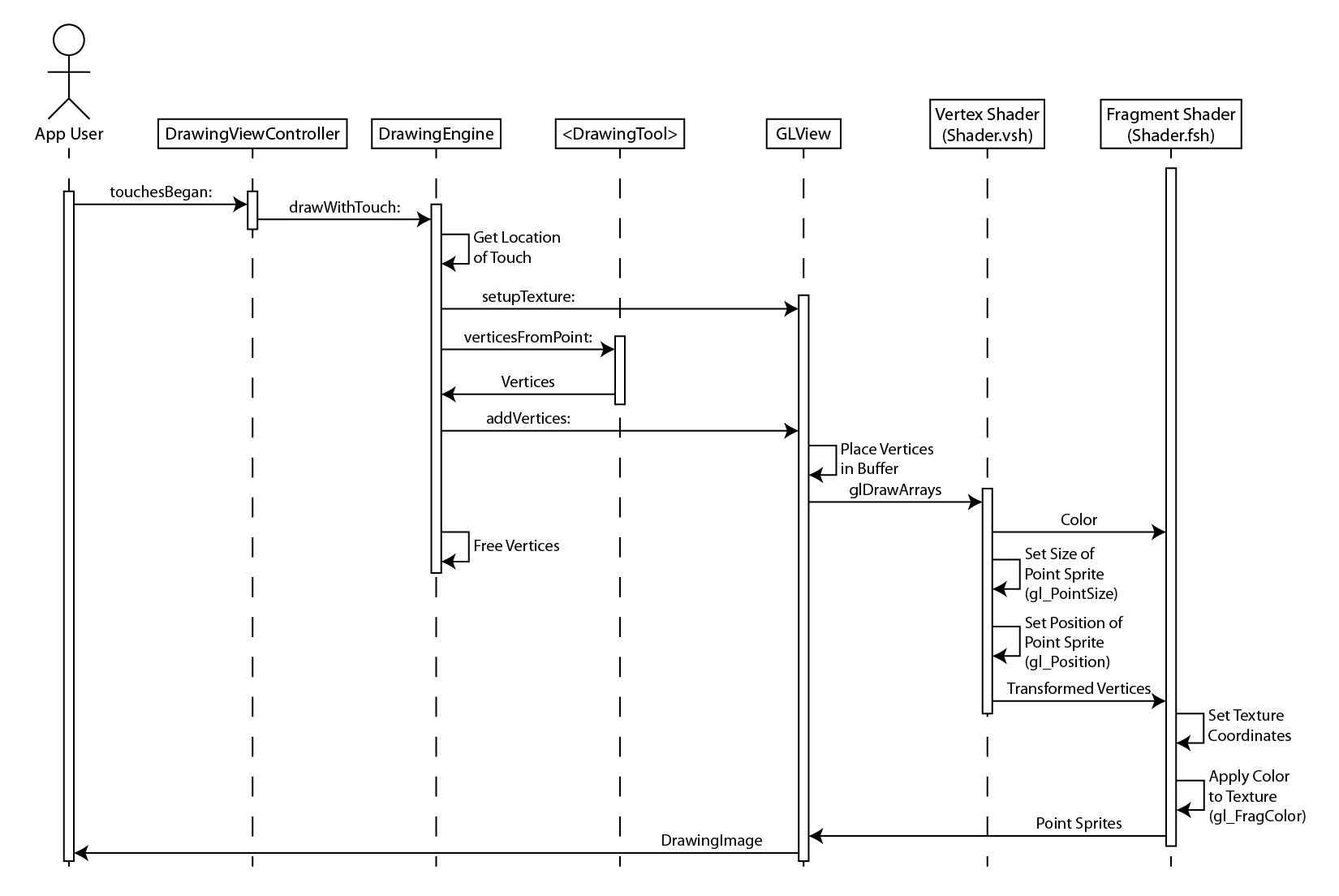


Figure 3‑1 Drawing Sequence Diagram

Below is a detailed description of some of the events that are illustrated in the diagram.

1. The DrawingViewController receives the initial touch event, which is passed to the DrawingEngine.
2. The DrawingEngine is responsible for passing the touch location as a CGPoint to the active DrawingTool.
3. The DrawingTool uses the current point and previously received point to create a series of vertices for rendering.
4. The DrawingEngine receives passes the vertices created by the DrawingTool to the GLView for rendering.
5. The GLView passes the vertices to a buffer and calls the glDrawingArrays method to draw them.
6. The vertex shader passes the color for each vertex to the fragment shader and sets the size and position of the *point sprite* by defining gl\_PointSize and gl\_Position respectively.
7. The vertex shader passes the transformed vertices down the graphics pipeline.
8. The fragment shaders sets the texture coordinates for each *point sprite* using the texture2D() method and sets the color for each *point sprite* by setting the gl\_FragColor variable.
9. The *point sprite* is then rendered to the GLView, which is displayed to the *app user*.

### Drawing Engine Algorithms

#### Point Sprites

#### Curve Interpolation

##### Calculating the Control Points

##### Estimating the Curve Length

##### Calculating Curve Points

#### Changing Orientation

TBD – Need to make sure the issues with this approach are discussed and how this will be corrected in the future.

#### Saving Drawing Images

## Gallery Design

### Gallery Structure

TBD – This section should provide a high level overview of the gallery.

TBD – Move the JSD describing the Gallery’s design here from section 3.1 of the old document (This diagram also needs to be corrected, remove strokes under the Drawing element and replace with Drawing Thumbnail). The explanation that goes along with the diagram would need to move here as well.

### Gallery Classes

TBD – Create a class diagram showing all the class relationships for the gallery

## Class Descriptions

TBD – Need a table or some other way to describe the classes in detail that is much more compact and organized then the old method. Can separate this into separate tables for

# User Interface Design

TBD – I maybe able to reference the user interface mockups in the SRS for this.

TBD – Pull (or reference) the UI mockups from the SRS

TBD – Create storyboards were and if needed to describe the rest of the UI.

# Security

TBD – This is simple, there is no security other then what Apple provides on its devices.

# Restriction, Limitations and Constraints

TBD – Explain the platforms and OS versions that it will be limited to. Also explain what the software cannot do (load previous drawings, rearrange drawings).

# Testing Issues

TBD – Explain the testing process for IdeaStorm (Manual Test Scripts) and the test scripts that were developed.

# Design Process

TBD – Explain the process of designing IdeaStorm including this rewrite.

TBD – Need to explain that both major components were built in different semesters which explains some of the differences in the design approach (heavy use of delegate and protocols in last semester for gallery)

# Conclusion

TBD – Explain each of the below sections using bullet points

## What Worked Well

## Challenges

## Lessons Learned

# Requirements Traceability

TBD – Need to separate these by the requirement type and the major part of the software they belong to (drawing engine or gallery).

## Drawing Engine Requirements

### External Interface Requirements

|  |  |
| --- | --- |
| Requirement | Document(s), section(s) and/or process(es) that meets the requirement |
| **EIR005** |  |
| **EIR018** |  |
| **EIR001** |  |
| **EIR002** |  |
| **EIR012** |  |
| **EIR022** |  |
| **EIR025** |  |
| **EIR019** |  |
| **EIR011** |  |
| **EIR021** |  |
| **EIR028** |  |
| **EIR026** |  |
| **EIR027** |  |
| **EIR029** |  |

### Functional Requirements

|  |  |
| --- | --- |
| Requirement | Document(s), section(s) and/or process(es) that meets the requirement |
| **FR001** |  |
| **FR002** |  |
| **FR003** |  |
| **FR013** |  |
| **FR033** |  |
| **FR032** |  |
| **FR004** |  |
| **FR005** |  |
| **FR017** |  |
| **FR020** |  |
| **FR023** |  |
| **FR021** |  |

### Performance Requirements

|  |  |
| --- | --- |
| Requirement | Document(s), section(s) and/or process(es) that meets the requirement |
| **PR001** |  |
| **PR002** |  |
| **PR005** |  |

## Gallery Requirements

### External Interface Requirements

|  |  |
| --- | --- |
| Requirement | Document(s), section(s) and/or process(es) that meets the requirement |
| **EIR044** |  |
| **EIR038** |  |
| **EIR039** |  |
| **EIR030** |  |
| **EIR031** |  |
| **EIR032** |  |
| **EIR033** |  |
| **EIR034** |  |
| **EIR035** |  |
| **EIR036** |  |
| **EIR037** |  |
| **EIR041** |  |
| **EIR045** |  |
| **EIR047** |  |

### Functional Requirements

|  |  |
| --- | --- |
| Requirement | Document(s), section(s) and/or process(es) that meets the requirement |
| **FR041** |  |
| **FR035** |  |
| **FR042** |  |
| **FR036** |  |
| **FR037** |  |
| **FR014** |  |
| **FR038** |  |
| **FR040** |  |

### Performance Requirements

|  |  |
| --- | --- |
| Requirement | Document(s), section(s) and/or process(es) that meets the requirement |
| **PR006** |  |

# References

Bullock, B. (2009, September 30). Approximate length of a cubic bezier curve. *LeMoDa.net*. Retrieved September 13, 2011, from http://www.lemoda.net/maths/bezier-length/index.html

Chung, C., & Bucanek, J. (2011). *Pro Objective-C design patterns for iOS*. New York: Friends of ED/Apress.

Cocoa Core Competencies: Model-ViewController. (2012, February 16). *Mac OSX Developer Library*. Retrieved June 1, 2012, from https://developer.apple.com/library/mac/#documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

Drawing a curve in OpenGLES - how? - GameDev.net. (2009, May 7). *GameDev.net*. Retrieved August 26, 2011, from http://www.gamedev.net/topic/534082-drawing-a-curve-in-opengles---how/

Harmon, T. (2009, October 12). Resize a UIImage the right way ‚Trevor’s Bike Shed. *Trevor's Bike Shed*. Retrieved February 22, 2012, from http://vocaro.com/trevor/blog/2009/10/12/resize-a-uiimage-the-right-way/

iPhone - GLPaint Save Image - Stack Overflow. (2009, December 22). *Stack Overflow*. Retrieved April 22, 2012, from http://stackoverflow.com/questions/946700/glpaint-save-image

Lamarche, J. (2010, November 2). iPhone Development: OpenGL ES 2.0 for iOS, Chapter 4 - Introducing the Programmable Pipeline. *iPhone Development*. Retrieved August 12, 2011, from http://iphonedevelopment.blogspot.com/2010/11/opengl-es-20-for-ios-chapter-4.html

Lazuka, K. (2009, November 19). Life Kid / Point Sprites and OpenGL ES 2.0. *Life Kid*. Retrieved September 13, 2011, from http://klazuka.tumblr.com/post/249698151/point-sprites-and-opengl-es-2-0

McCaughey, D. (2010, September 30). Able Pear Software: Creating a GUID or UUID in Objective-C. *Able Pear Software*. Retrieved March 11, 2012, from http://blog.ablepear.com/2010/09/creating-guid-or-uuid-in-objective-c.html

Pinto, A. (2009, August 1). The Generic UWF Maintenance Process - Short Version. *Generic Analysis Report Standard*. Retrieved August 28, 2011, from www.cs.uwf.edu/~seadmin/GumpShort/anastan.htm

Pressman, R. S. (2010). *Software engineering: a practitioner's approach* (7th ed.). New York: McGraw-Hill Higher Education.

Smooth line connecting CGPoints. (2010, May 13). Developer Forums. Retrieved May 5, 2011, from https://devforum.apple.com/message/41732?start=0&start=0

Wenderlich, R. (2010, September 7). How To Save Your App Data With NSCoding and NSFileManager | Ray Wenderlich. *Ray Wenderlich | Tutorials for iPhone / iOS Developers and Gamers*. Retrieved March 7, 2012, from http://www.raywenderlich.com/1914/how-to-save-your-app-data-with-nscoding-and-nsfilemanager