EECS 393: Software Enginnering

Design Document

for

Android 3D Photocopier App

Authors:
Austin Hacker, Joshua Tang, Callum Grant, Alex Campbell

October 12, 2015

Contents

Revision History

Name	Date	Reason for Change	Version
Austin Hacker	10/12/2015	initial draft	1.0 draft
Austin Hacker	10/12/2015	changes due to inspection reports	1.0 draft 2

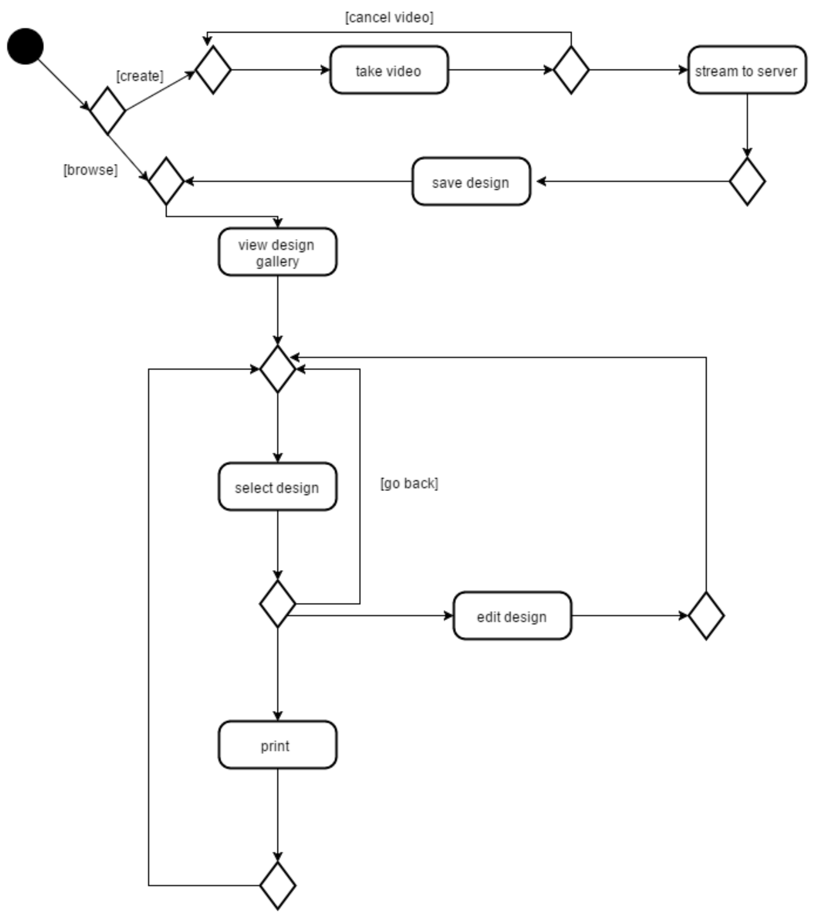
Part I

Introduction

1 Purpose

This purpose of this document is to outline the essential design of the 3D photocopier app for Android OS. An object oriented design is proposed. This design choice is based off of the maintainability this design would allow and the restriction on Android development to Java. A very high-level activity diagram is provided below. This is meant to describe basic functionality of the software and its components. Each component is described in further detail in the following sections.

User Activity Diagram



2 Scope

This document will depict the architecture of the Android 3D Photocopier App currently in preproduction by the aforementioned authors.

3 Overview

The following parts will each detail the architecture of a given module within the app. Each module is created to reduce coupling as much as possible and so they operate nearly independently of each other. Each part begins with a detailed activity diagram for the specific module. The classes and methods within the module are then listed. The modules are described in the following way:

Part 2 Creation of a 3D mesh (App side)

Part 3 Creation of a 3D mesh (Server side)

Part 4 Editing a 3D mesh

Part 5 Printing a 3D mesh

Part 6 Requirements Traceability Matrix

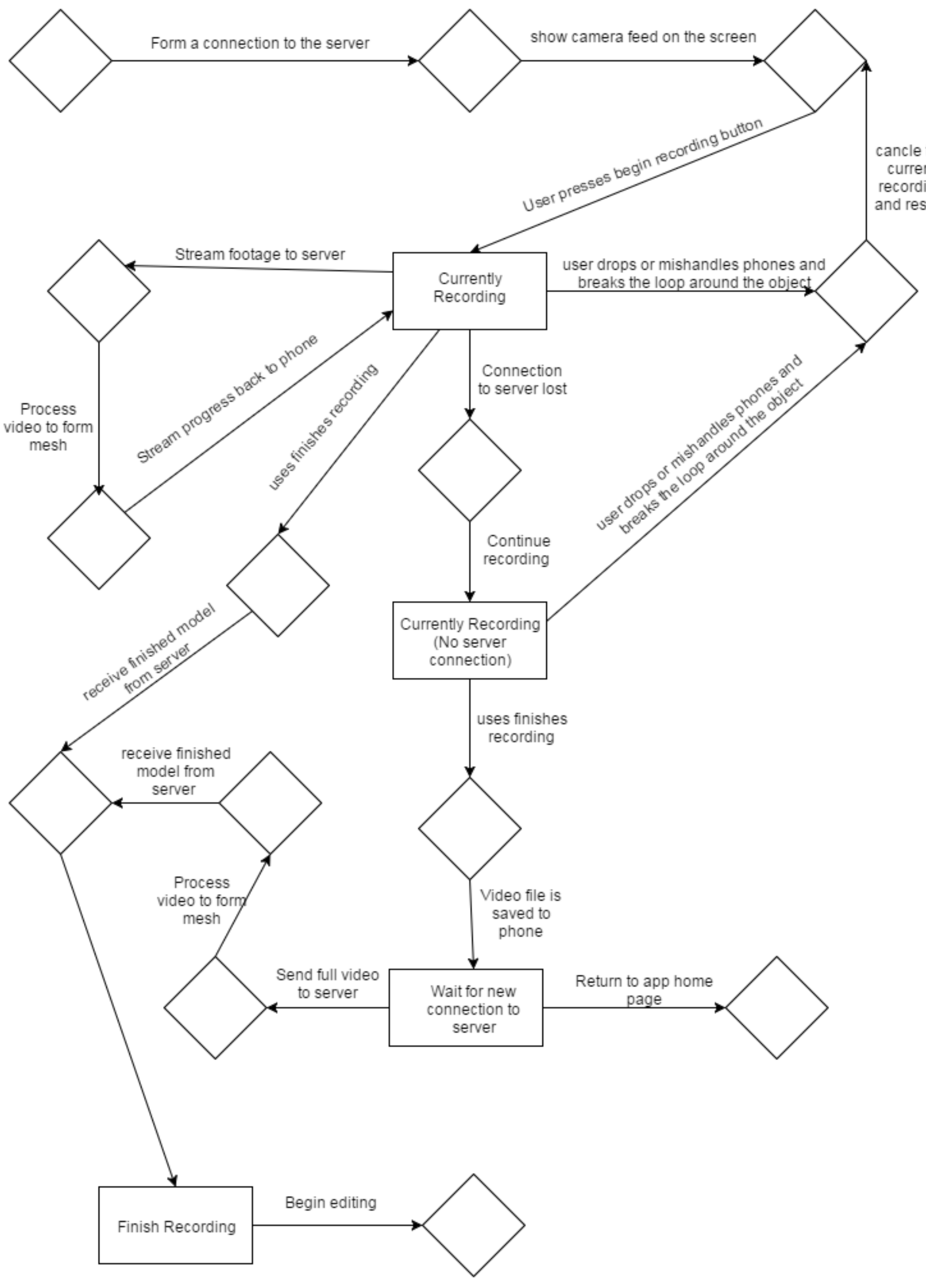
4 Acronyms

- SLAM: Simultaneous localization and mapping
- ORB: Oriented Fast and Rotated BRIEF
- BRIEF: : Binary robust independent elementary features
- SURF: Speeded up Robust Features
- SIFT: Scale-invariant feature transform

Part II
Video Capture Module

This section goes over all the steps and events that can happen while capturing video and sending that video to the server to process into the 3d model.

Detailed Video Capture Activity Diagram


Form a connection to the server
show camera feed on the screen
User presses begin recording button
cancle the current recording and restart
Stream footage to server
Currently Recording
user drops or mishandles phones and breaks the loop around the object
Process video to form mesh
Stream progress back to phone
uses finishes recording
Connection to server lost
user drops or mishandles phones and breaks the loop around the object
Continue recording
receive finished model from server
Currently Recording (No server connection)
uses finishes recording
receive finished model from server
Process video to form mesh
Video file is saved to phone
Send full video to server
Wait for new connection to server
Return to app home page
Begin editing
Finish Recording

1 Class Interfaces

1.1 ServerConnection

The first thing the phone will do is find the server and connect to it. ServerConnection will access the IP of a viable server and establish a persistent connection with that server.

1.1.1 Public Method GetServerIP

$getServerIP()$

The getServerIp() method figures out the IP address of the server it needs to connect to.

1.1.2 Public Method ConnectToServer

$connectToServer(String\ IP)$

The connectToServer() method takes the given IP address and makes a connection to that server.

1.1.3 Public Method FilesToSend

$filesToSend()$

The filesToSend() method is used to check if there are any previous video files on the phone that need to be sent to the server to process because they were unable to be sent when recorded. It is checked every time the phone connects to the server.

1.1.4 Public Method SendData

$sendData(video)$

The sendData method is used to send a given video to the server for processing. This is used if the connection to the server is interrupted or unavailable during creation of the video.

1.1.5 Public Method ReceiveMesh

$receiveMesh()$

The receiveMesh method listens to the server and waits for the server to send a mesh object back to the phone.

1.2 Video Recording

Once video recording starts the user has to circle around the object to make sure that a full 3D model can be made for the object. This will continue until the user is satisfied that they have gotten enough of the object recorded.

1.2.1 Public Method StreamData

$streamData(videoSegment)$

The streamData() method is used to continuously stream the recorded data the phone is generating to the server so that it can render it in real time.

1.2.2 Public Method ReceiveData

$receiveData()$

The receiveData() method is continuously running in the background on the phone while it is recording video so that it can report the progress of the mesh to the user and give feedback on what parts of the object still need to be recorded.

1.2.3 Public Method UserError

$userError()$

The userError() method is called when the user makes some kind of mistake such as dropping the phone or turning the phone away from the object that would cause the video to not be suitable for generating a 3D model. In the event that this happens the video recording will start and the user will be prompted to start again.

1.2.4 Public Method ServerLost

$serverLost(String\ IP)$

The serverLost() method is called if the phone happens to lose its connection to the server for any reasons, such as hitting a dead spot in coverage or the access point it was using lost power, it will not stop recording but will save the video file to the phone with a flag on it so that the next time it does connect to the server it will send that video file to it to be processed.

1.2.5 Public Method FinishRecording

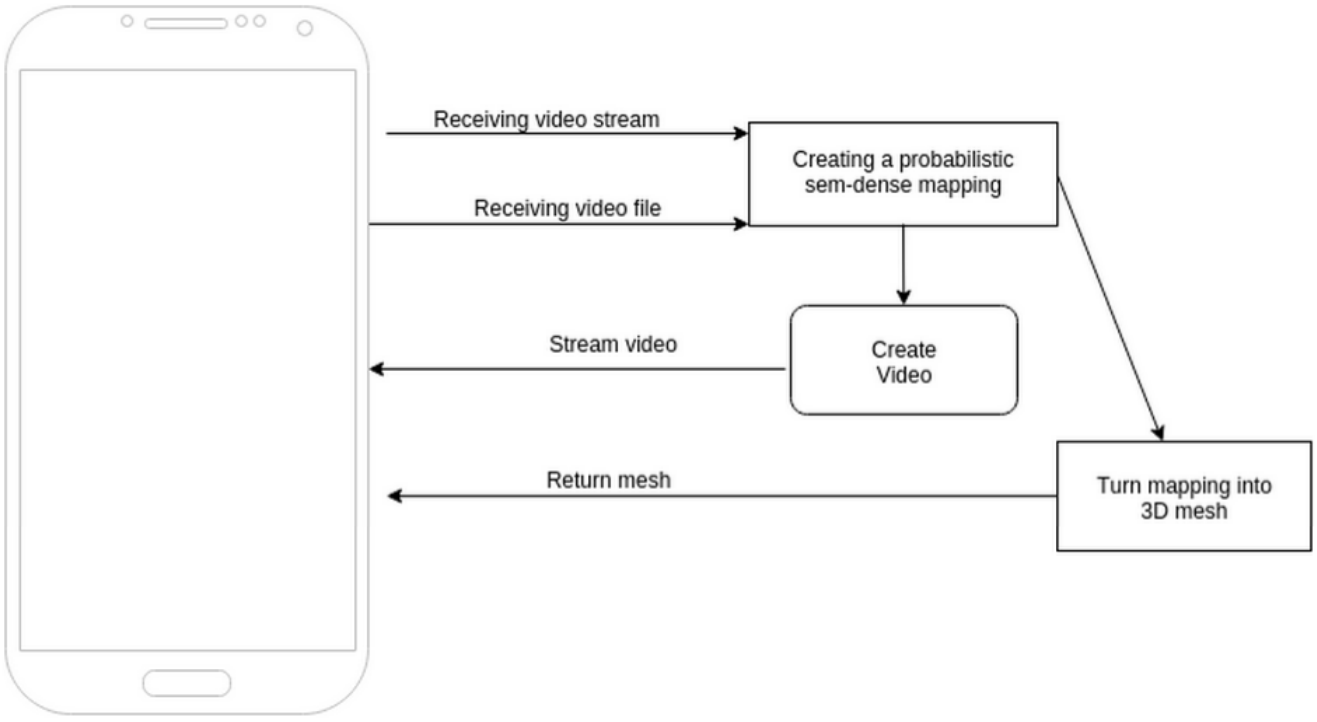
$finishRecording()$

The finishRecording() method happens when the user indicates that they are done recording the object. If the phone still has its connection to the server than it will wait for the finished 3D model to be sent to it so that the user can then proceed to editing. If the phone does not still have a connection with the sever it will instead just save the video file and then proceed back to the applications home page.

Part III
3D Smoothing Module

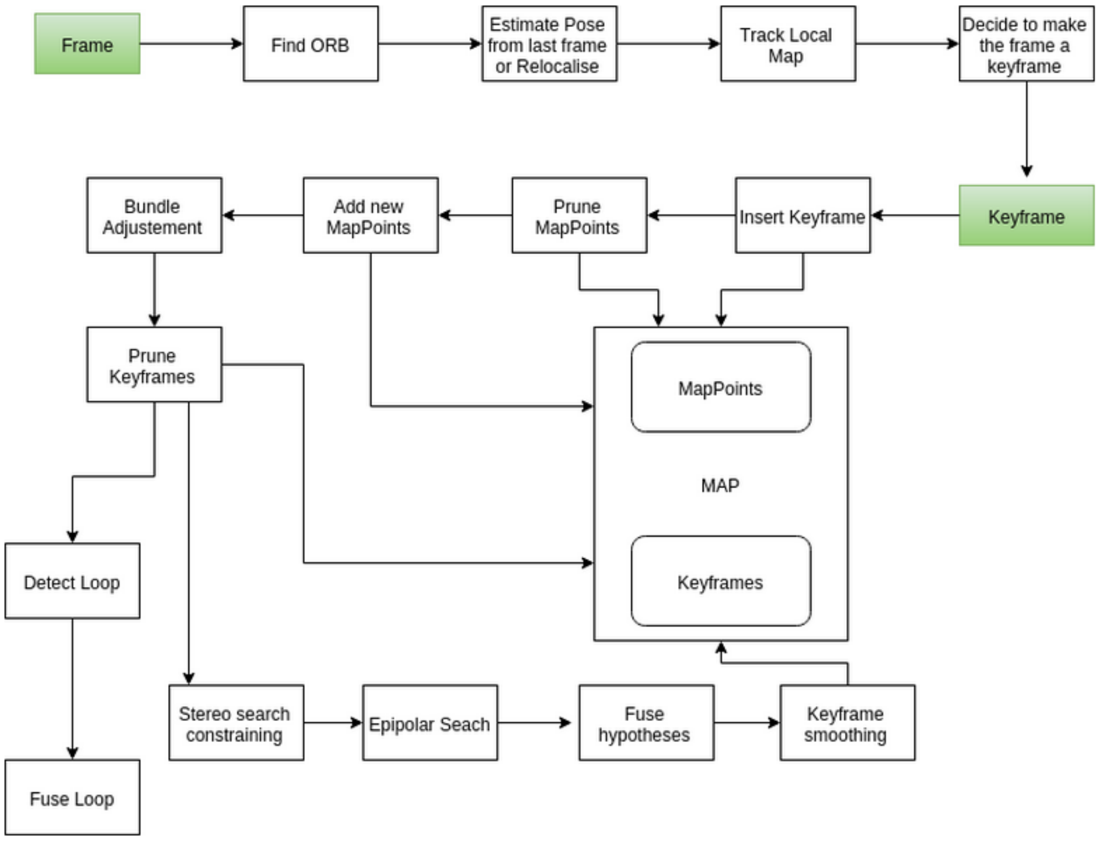
1 Server side Processes

Everything that happens in the server can be resumed in the following diagram, but this level of abstraction doesn't give a good idea of how the process works. We will explore in depth how we can map real world objects and how we can turn that map into a mesh.



2 Building a Probabilistic Semi-Dense Point Map

We will use a Monocular SLAM algorithm in order to map a real world object. We have decided to implement the algorithm used in Probabilistic Semi-Dense Mapping from Highly Accurate Feature-Based Monocular SLAM, by Raul Mur-Artal and Juan D. Tardos (July 2015). This paper uses the ORB-SLAM algorithm which utilizes ORB [ORB: an efficient alternative to SIFT or SURF, Ethan Rublee et al. (2011)] to find features in each frame the server receives. The mapping is created by matching the same ORB from different keyframes and using epipolar geometry to figure out where the ORB points go in 3D space. The following diagram explains in detail how the algorithm works:

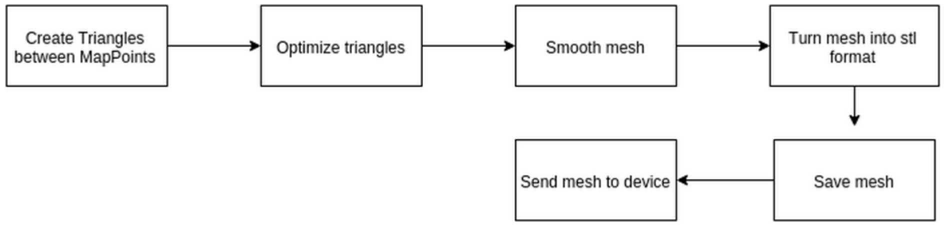


The loop detection feature of the algorithm will detect when a loop is completed and adjust the map to reflect it. This will remove a lot of noise from imperfect localisation. This is the most we can do to correct for failure in this phase of the process. This step will always return a map. Once we obtain the semi-dense map we need to turn it into a mesh that can actually be 3D printed.

3 Creating the 3D mesh

We will now turn the map we have into a mesh using an algorithm based on Triangulating Point Set Surfaces with Bounded Error by Carlos E. Scheidegger, Shachar Fleishman, Cludio T. Silva(2005). This method creates a mesh from the MapPoints by triangulating them. We will be using the maps keyframes to define the bounded area. Once we have this mesh, we need to smooth it. We will use Laplacian Smoothing in order to get a smooth mesh.

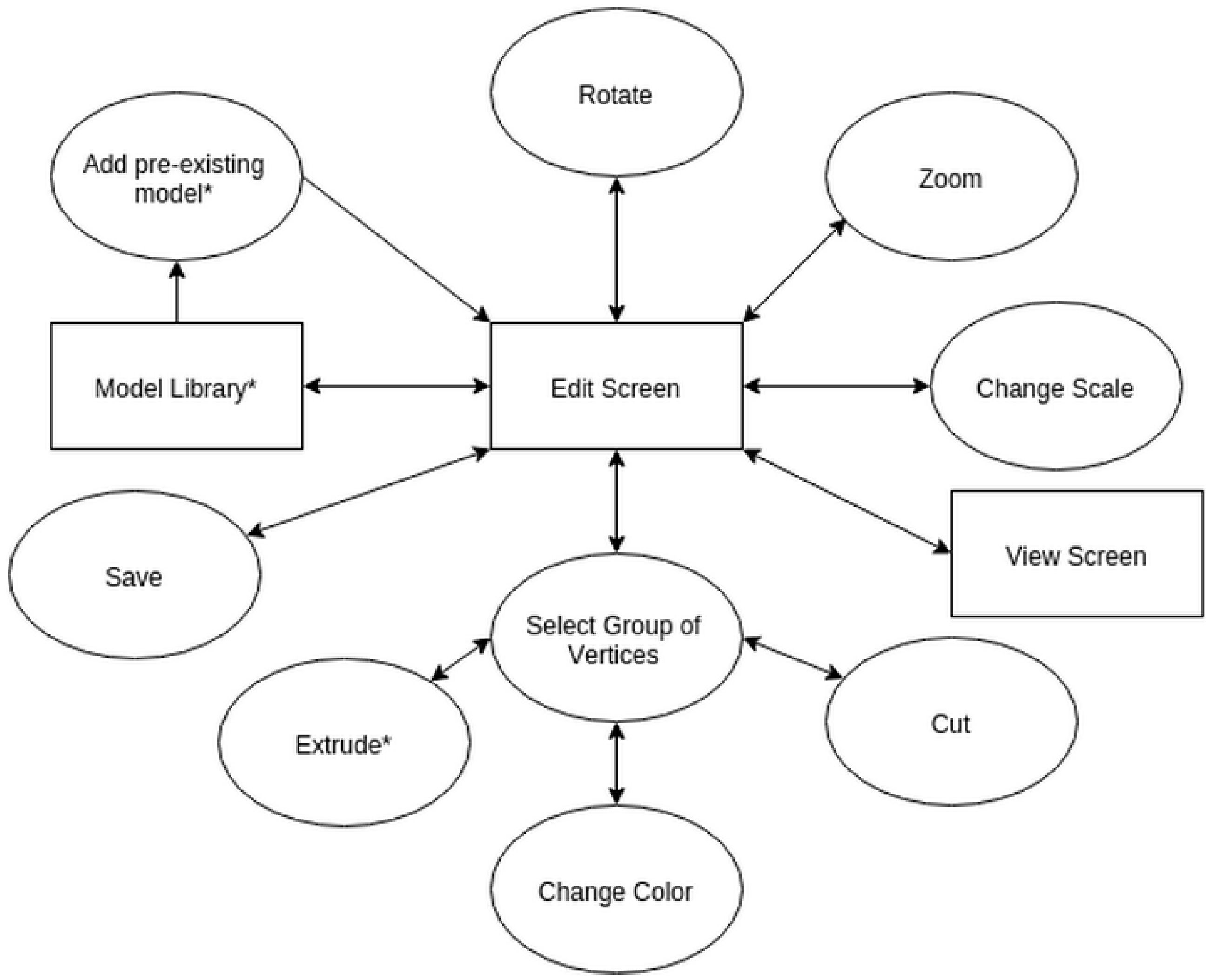
Once we have created a smooth mesh, we will turn it into the stl format (we have selected this format because it is usable by 3D printers that are available to us and because libraries that help us convert meshes into the stl format are easy to use). We then need to save the mesh and send it back to the device.



Part IV

Editing Module

Detailed Edit Activity Diagram

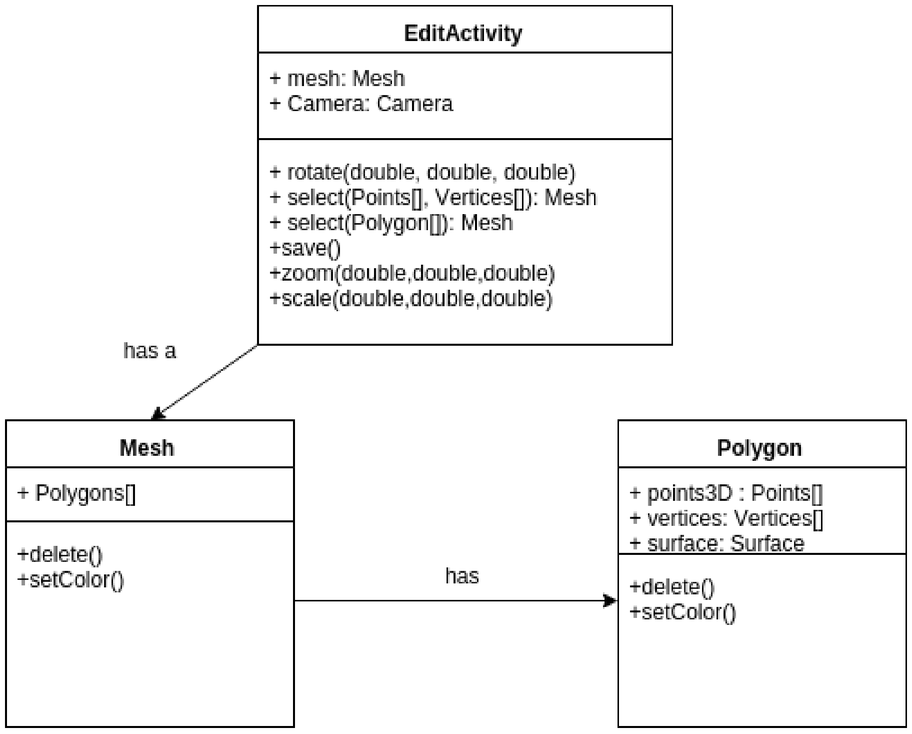


*not a priority, will not be in V1.0

1 Android Processes

Once the app receives a mesh from the server, the user needs to be able to edit it to their specification. From the view activity, the user can get to the edit activity, where they can modify the mesh as much as they want. These modifications include cutting, extruding and changing the color of any selected set of vertices and the surfaces between them. As in the view activity, we need to be able to rotate the model in every direction and zoom in and out. We can also edit the scale of the mesh, which will set the scale at which the mesh is to be printed if the user decides to print it. The user can save the changes that they have made.

2 Class Interfaces



2.1 EditActivity

The EditActivity is an Android Activity which is used to render the mesh being edited. It can be interacted with in order to edit the Mesh it is displaying.

2.1.1 Public Method Rotate

$rotate(double\ x,\ double\ y,\ double\ z)$

This method changes the position of the camera to that of the first 2 arguments and it rotates at the speed of the third argument.

2.1.2 Public Method Select

$select(Point[]\ points,\ Vertex[]\ vertices)$

$select(Polygon[]\ polygons)$

This method returns a mesh which consists of all the argument polygons or points and vertices.

2.1.3 Public Method Save

$save()$

This method saves the mesh as an stl file.

2.1.4 Public Method Zoom

$zoom(double\ x,\ double\ y,\ double\ z)$

$zoom(Point\ p,\ double\ d)$

This method will change the distance of the camera from the given point(or 2 first arguments) by the last argument.

2.1.5 Public Method Scale

$scale(double\ x,\ double\ y)$

This method sets the scale of the resulting model to the inputs.

2.2 Mesh

A mesh is a collection of polygons. Packaging the polygons together in one object allows for easier use by the app user as they can then delete the entire mesh simply or change the color if their 3D printer supports color

2.2.1 Public Method Delete

delete()

This method deletes the all polygons making up the mesh. Effectively, this method will call the delete method of each underlying polygon to avoid coupling.

2.2.2 Public Method setColor

setColor()

This method sets the color of all polygons making up the mesh. Effectively, this method will call the setColor method of each underlying polygon to avoid coupling.

2.3 Polygon

A Polygon is the base make-up of a mesh. A Polygon is easier to work with than a series of points and vertices.

2.3.1 Public Method Delete

delete()

This deletes all instances of the points and vertices contained in the polygon.

2.3.2 Public Method setColor

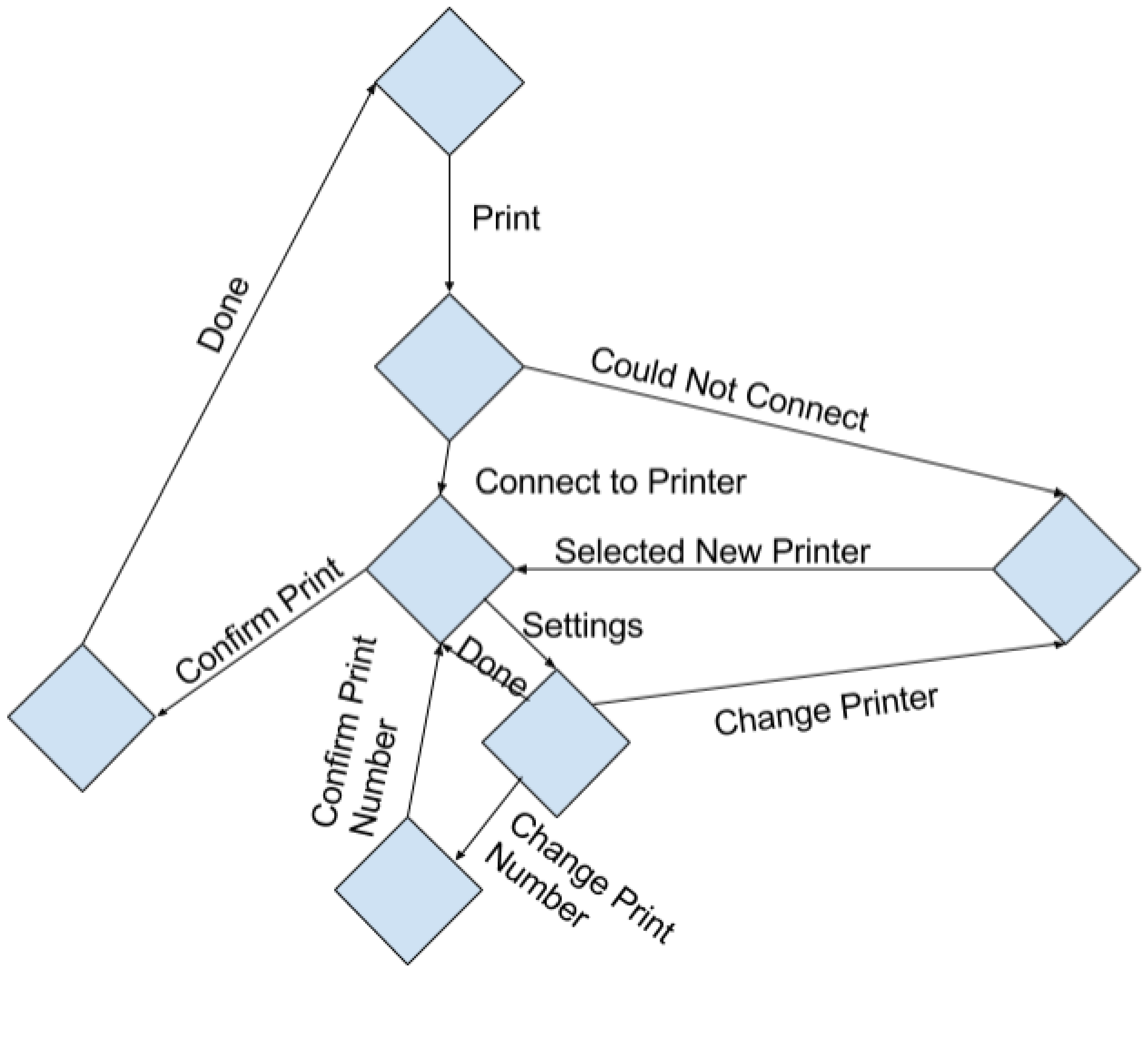
setColor()

This method sets the color of the surface of the polygon.

Part V

3D Print Module

Detailed Print Activity Diagram



1 Responsibilities

Requirements for the 3D printing portion of the app are as follows:

1. View 3D printers on the network.
2. Connect to a 3D printer.
3. Verify successful connection to a printer.
4. Store printer details.
5. Send 3D model to printer.
6. Send print command.
7. Notify user when the print is finished.

2 Inter-Agent Protocols

The print portion of the app must be able to interface with the edit function, as well as the stored set of 3D models. From there, we should be able to print out the given model. To do this we need a class, PrinterController, to store the model and details we are preparing to print, and choose a printer. To assist with the printer we are using, we need a class, PrinterSettings, to manage the details of the printer we are using, and a class 3DPrinter to store the details of a given printer.

3 Class Interfaces

3.1 PrinterController

An instance of class PrinterController stores the information necessary to print a model to a 3D printer. The current print settings are stored in an object of type PrinterSettings. Printer details are stored in an object of type 3DPrinter.

3.1.1 Public Method SetModel

$SetModel(String\ modelPath)$

SetModel stores the location of a model on the phone, to be used when the time comes to print.

3.1.2 Public Method GetModel

$GetModel()$

GetModel returns the model currently stored for printing.

3.1.3 Public Method GetSettings

$GetSettings()$

GetSettings returns the PrinterSettings.

3.1.4 Public Method GetPrinters

$GetPrinters()$

GetPrinters searches the network for 3D Printers and returns their types and Addresses, contained within objects of type 3DPrinter.

3.1.5 Public Method ChoosePrinter

$ChoosePrinter(3DPrinter\ printer)$

ChoosePrinter stores a printer in the PrinterSettings object.

3.1.6 Public Method PrintModel

$PrintModel()$

PrintModel waits for user confirmation of current print settings, allowing them to change if necessary, before printing.

3.2 3DPrinter

The 3DPrinter class stores the details of a Printer.

3.2.1 Public Method GetName

$GetName()$

GetName returns the name of the 3DPrinter.

3.2.2 Public Method GetAddress

$GetAddress()$

GetAddress returns the address of the 3DPrinter on the network.

3.3 PrinterSettings

PrinterSettings keeps track of the current Print settings for the app.

3.3.1 Public Method GetCurrentPrinter

$GetCurrentPrinter()$

GetCurrentPrinter returns the 3DPrinter object storing the details of the printer we are currently set to print to.

3.3.2 Public Method SetCurrentPrinter

$SetCurrentPrinter(3DPrinter\ printer)$

SetCurrentPrinter sets our printer to the printer provided by the given 3DPrinter object.

3.3.3 Public Method GetNumber

$GetNumber()$

GetNumber returns the number of prints we want to do for our object.

3.3.4 Public Method SetNumber

$SetNumber(int\ printNumber)$

SetNumber sets our print number to the provided amount.

Part VI
Traceability Matrix

	Parts			
	II	III	IV	V
OE-1	x		x	x
OE-2		x		
DI-1				
DI-2		x		
DI-3				
DI-4	x			
UD-1				
UD-2	x			
UD-3				
PR-1				
PR-2		x		
PR-3				x
PR-4				x
SC-1		x		
SC-2				
ST-1	x			
ST-2				
ST-3				
ST-4				
ST-5		x		
ST-6			x	
PE-1				
PE-2	x			
PE-3		x		
SU-1			x	
SU-2			x	
SU-3	x			
CO-1				x
CO-2				
CO-3	x			
CO-4	x			
CO-5	x			
SE-1				
SE-2				
SE-3		x		
SE-4		x		
SE-5		x		

Individual Work Breakdown

The following people wrote the sections associated with their names.

Austin Hacker : Part I, VI

Callum Grant : Part V

Josh Tang : Part III, IV

Alex Campbell : Part II