COMP4971C – Independent Studies Report

A Smart Display System – Meta Display Implementation

Student:

Wicaksana, Jeffry

Advisor:

Prof. David Rossiter

Table of Contents

List of Figures:

1. Introduction

1.1 Overview

A substantial percentage of people in the world have a modern smart phone and/or a tablet. Many people have 2 or 3 such devices. However, there is currently a huge wasted opportunity for these devices to work together with each other. Why can't a group of students put their devices close to each other in a 2*2 arrangement and watch a YouTube video together, with a quarter of the video being shown on each device? On a larger scale, a large 80-inch monitor may cost in the region of HK$80,000 [1]. Yet by using smart devices which are already cheap and in abundance in conjunction with each other the cost of an equivalent system would be a fraction of the price and can be used in multiple innovative ways which a flat 2D display cannot.

1.2 Project Outline

We envisage the following modes for the proposed Smart Display System. It should be noted that when we refer to 'display' both image and audio are included. Essentially, whatever is being displayed on the source device is transmitted. For example, this might be a video currently being played by the user, a game, a browser being used to read email or a web site such as Facebook, and anything else an individual device is typically used to display.

1. **Simple display broadcast**. In this mode the display of 1 individual device is shown on *n* devices. This are multiple applications where this can be of great benefit. For example, in a classroom environment a teacher may project the display of his device directly to the display of all students' devices, to help with tutoring. This could have direct cost benefits. For example, a school or University would not need to purchase a large display device for each classroom such as a large monitor or projector, as is typically the case now.
2. **Multiple device monitoring.** In this mode the displays of *n* devices are shown on 1 device. To continue the illustration of a training environment, a teacher

could use this feature to see the activity of any or all students while they work on a particular project using their own devices, and offer advice accordingly. Similarly, members of a group project can see exactly what other members are doing, which is be of great benefit as they work together.

3. **Multiple device display in aggregate of one display, in 2D.** In this mode the image from 1 device is displayed on *n* devices, treating the *n* devices as 'virtual windows' into part of the display. The result is that the whole display is visible when all devices are viewed together. A simple illustrative figure is given below, using a 2D treatment. In our proposed Smart Display System the devices can be physically organized in any creative and interesting way, making their usefulness far greater than would be the case if a fixed row of identical devices was used. This applies to businesses as well as individuals using the system. For example, an advertising business can use a creative array of varying size devices in a varying set of angles and distances from each other, but when viewed together give the appearance of small windows into a single display. This would be highly innovative, attractive and attention grabbing, not to mention highly affordable compared to the very high cost of comparable large screen displays.

Image 1.2. 1 Example of one usage in our project

4. **Multiple device display in aggregate of one display, in 3D.** This mode extends the previously described 2D display mode by dynamically altering the image shown on any of the output devices being used in aggregate such that, simply speaking, the angle of the device in the third dimension is used to appropriately transform the image displayed on the device. The result is that a viewer at a specific position perceives each individual device as correctly contributing to the display in aggregate, regardless of the 3D orientation of any device. This means that the Smart Display System can be used for even more creative output displays than those provided by the 2D mode alone. For example, some or all of the display devices being used in aggregate can be rotated in 3D in a shop display system for an even more remarkable result.

In this report, writer will mainly focus on implementing the third capability of Smart Display and work on the smart display called by meta display, to provide users with similar experience from multiple angles and point of views without having to worry about the location of their tablets and devices. Mostly implementation of Meta-Display can be shown for commercial use.

A database will need to be compiled containing the relevant details for a range of different electronic gadgets. Noted attributes will include the speed of the devices and the screen size. This data is needed for accurate display by each individual device as part of the set. Devices will communicate via any appropriate channel, such as Wi-Fi or high speed Internet.

1.3 Goals and Impact

Our project aims to transform the way people use their gadgets from individual smart device usage to multiple units working together, supporting a huge range of applications, and with no additional hardware cost. This idea will be of benefit to many millions of individuals and companies around the world which use display devices.

Our Smart Display System will also potentially give a new lease of life for older smart devices which would otherwise lie unused or be dumped, as these devices can be used as part of the multiple display system. In this way our system significantly benefits the environment by extending the life of smart devices.

There are multiple opportunities for collaboration. For example, a target customer base is large advertising companies in need of creative, novel and affordable advertising mechanisms.

2. Implementation and Result

2.1 Meta Display

Meta display is one of the feature of Smart Display system where object shown or displayed from a smart device will always be seen in the same size and same orientation no matter where the viewer's eyes are located. For instance, the whatever the orientation of the tablet below is, it should still be showing image of globes of picture facing toward the same way. Instead of being rotated accordingly to the tablet.

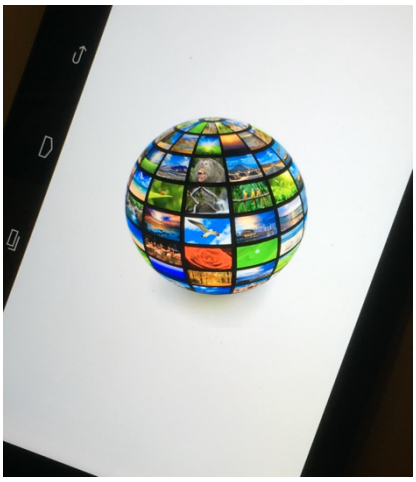


Image 2.1 1 Globe example

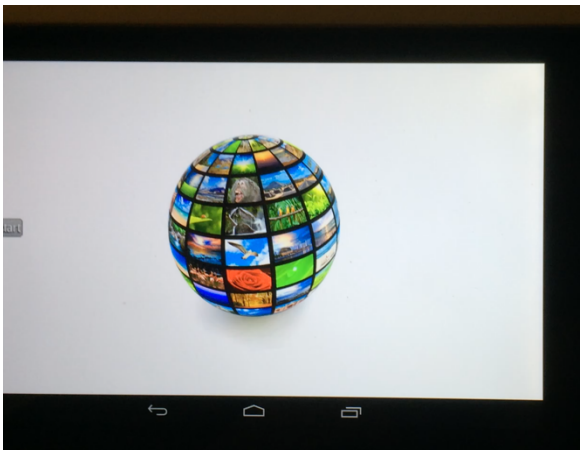


Image 2.1 2 Different orientation of tablet for the Globe example

Besides that, multiple devices can synchronize altogether to show something, for example pair of eye that can always be seen 'the same' no matter how the smart devices in being oriented. For this case, size of the image will change accordingly to compensate humans' eye weaknesses while analyzing and seeing object. In the example below, I tried to keep the size of the eyeball to be the same from humans' perspective of viewing the display.

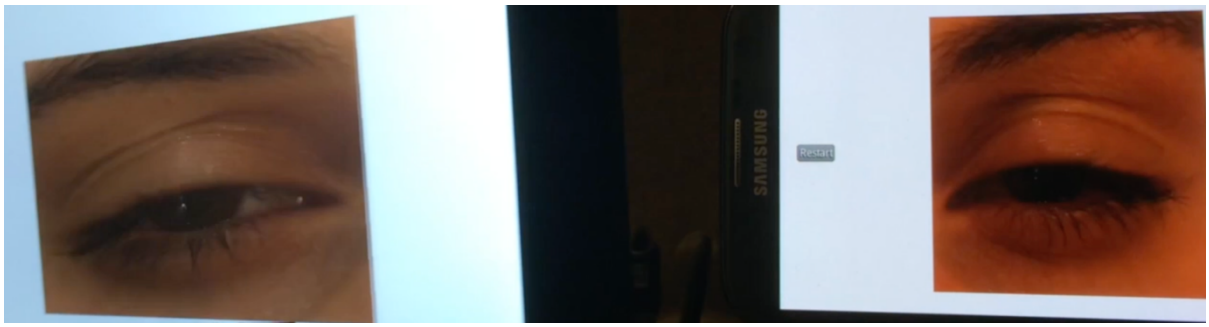


Image 2.1 3 Eye examples with different smart devices orientation – Starting Position

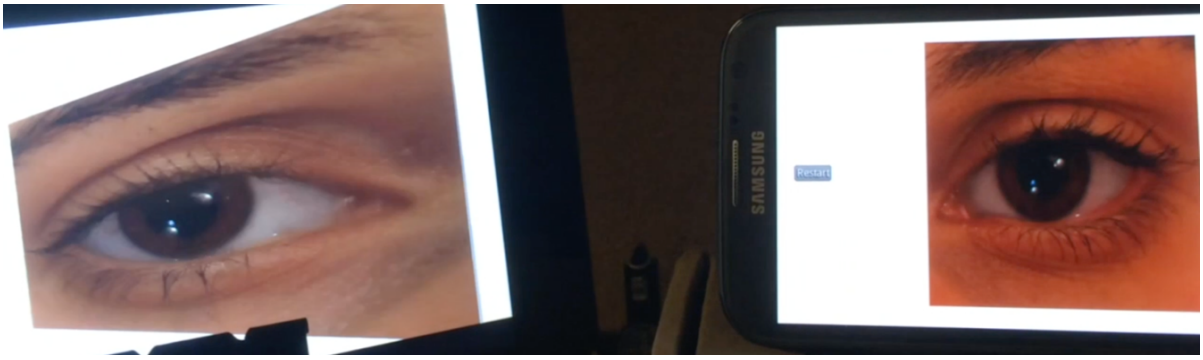
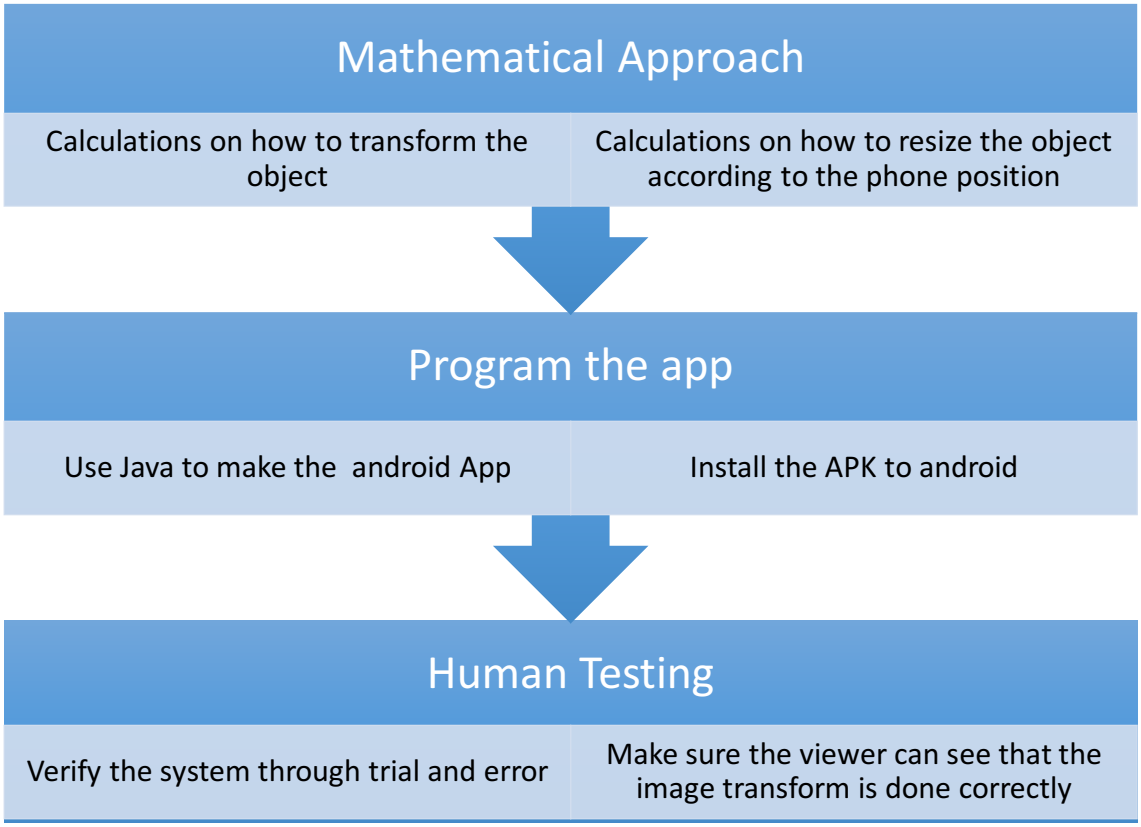


Image 2.1 4 Eye examples with different smart devices orientation – When doing rotation

2.2 Implementation

In order to implement this system, android app programming need to be done and in this project I am using well-known library for game developing called as LibGDX since with this library I do not have to change the code completely to implement the same thing in the IOS. Flow-chart for creating the code for creating the meta-display system can be seen below:



2.2.1 Mathematical Approach

First of all, I will explain about the mathematical Approach. In order to keep the image transformation, correct I have to process the pitch, yaw, and roll [2] data of the phone in three dimensional aspects. Besides that, current position of the phone needs to be considered as well since the rotation in either x, y, z axis will be different if the phone lies on different plane, for instance XY and YZ plane. In order to determine the transformation matrix and value, trial and error method is used for this project. Circular interpolation also has to be done to compensate the noise generated by the sensor inside the smart devices. Moreover, the image has to be resized according with the orientation of the phone as well in order to ensure that it is perceptually correct. In order to do so, I label the starting position with scaling factor 1 and the scaling factor on that axis value will increase to X value linearly, in this app I use 5, when the phone is rotated, with 180 degrees as the max value increment.

2.2.2 Programming the App

After analyzing the whole mathematical transformation, the app for devices are made and installed. Java is the main compiler for the app and android is the smart devices

being used. However, the main problem is different phone will subject the app to work differently so in that sense, human testing will be the major factor in determining the scale, value and rotation matrix.

2.2.3 Human Testing

Throughout each iteration and installation process, human testing needs to be done so that I can see whether the transform is not only mathematically correct but also perceptually correct. After doing the human testing, revision for the mathematical approach will be done again if it is not perceptually correct. This process keeps on looping until it finally seems alright and acceptable.

3. Limitations and Improvements

3.1 Limitations

Several limitations I faced in these project can be described as following:

1. Every smart device used has low tolerance against noise so the data reading of yaw, pitch, and roll may not be accurate.
2. Transformation matrix required a bit more complex mathematical approach but it is still not implemented yet.
3. For the human testing, constant rotation needs to be tested while rotating the smart devices in order to actually judge whether or not the view is perceptually correct but I do not have that in house.

3.2 Improvements

For the future development of this project, several things can be done:

1. Either buying new smart devices that has higher resistance toward noise to make sure the reading is correct or by implementing some kind of data filtering, such as Kalman Filter to keep the reading stable.
2. Implementing higher level mathematical transformation calculation to make sure that the image will be perceptually correct seen from different point and angles.

4. Reference

[1]"Electronics - Televisions | HK Free Classifieds | AsiaXPAT.com", *Hongkong.asiaxpat.com*, 2016. [Online]. Available: http://hongkong.asiaxpat.com/classifieds/televisions/. [Accessed: 13- May- 2016].

[2]"Inertial measurement unit", *Wikipedia*, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Inertial_measurement_unit. [Accessed: 13- May- 2016].

5. Appendix

```
public class MyGdxGame extends ApplicationAdapter {
	DecalBatch batch;
	SpriteBatch spriteBatch;
	Decal img;
	Decal img2;
	Decal img3;
	Decal img_temp;
	private BitmapFont font;
	public PerspectiveCamera cam;
	// New
	private Stage stage;
	private Table table;
	// New

	Vector3 target = Vector3.Zero;

	float deltaAngleX = 1.1f;
	float deltaAngleY = 1.9f;

	float current_x = 0f;
	float current_y = 0f;
	float z_callibration = 0f;
	float y_callibration = 0f;
	float x_callibration = 0f;
	float current_z = 0f;
	private OrthographicCamera chumbucket;
	int count =0;
	int count_img =0;
	Button resetbutton;

	@Override
	public void create() {
		Gdx.gl.glEnable(GL20.GL_DEPTH_TEST);
		Gdx.gl.glDepthFunc(GL20.GL_LESS);

		cam = new PerspectiveCamera(45, Gdx.graphics.getWidth(),Gdx.graphics.getHeight());
		cam.position.set(0, 0, 5);
		cam.near = 1;
		cam.far = 300f;

		chumbucket = new OrthographicCamera();

		img = Decal.newDecal(3, 3, new TextureRegion(new Texture("bird1.png")));
		img.setPosition(0, 0, 0);
		img.transformationOffset = new Vector2(0, 0);

		img2 = Decal.newDecal(3, 3, new TextureRegion(new Texture("bird2.png")));
		img2.setPosition(0, 0, 0);
		img2.transformationOffset = new Vector2(0, 0);

		img3 = Decal.newDecal(3, 3, new TextureRegion(new Texture("bird3.png")));
		img3.setPosition(0, 0, 0);
		img3.transformationOffset = new Vector2(0, 0);

		batch = new DecalBatch(new CameraGroupStrategy(cam));
		spriteBatch = new SpriteBatch();
```

```
		img_temp = img;
		z_callibration = Gdx.input.getAzimuth();
		y_callibration = Gdx.input.getRoll();
		x_callibration = Gdx.input.getPitch();
		// New Stuff
		stage = new Stage(new FitViewport(800, 480));
		Gdx.input.setInputProcessor(stage);

		Skin skin = new Skin(Gdx.files.internal("uiskin.json"));

		TextButton btn = new TextButton("Restart", skin);
		btn.addListener(new ClickListener() {

			@Override
			public void clicked(InputEvent event, float x, float y) {
				current_x = Gdx.input.getPitch();
				current_y = Gdx.input.getRoll();
				current_z = Gdx.input.getAzimuth();
				//cam.rotate(45, current_x, current_y, current_z);
			}


		});

		table = new Table();
		table.left().add(btn).fillX();

		table.setFillParent(true);
		stage.addActor(table);

		//end
		// Definition of Camera
}

// New
public void resize (int width, int height) {
   stage.getViewport().update(width, height, true);
}


public void dispose() {
   stage.dispose();
}
// new
Vector2 lastRotXY = new Vector2(0, 0);
Vector2 lastRotYZ = new Vector2(0, 0);
Vector2 lastRotZX = new Vector2(0, 0);
Vector3 lastRotXYZ = new Vector3(0,0,0);
float a = 0;

float lastRotZ = 0;
float totalRotZ = 0;

boolean currentDir = false;

@Override
public void render() {
		Gdx.gl.glClearColor(1, 1, 1, 1);
```

```
                Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT | GL20.GL_DEPTH_BUFFER_BIT);
                Gdx.gl.glEnable(GL20.GL_DEPTH_TEST);
                if(count <5){
                        z_callibration = Gdx.input.getAzimuth();
                        y_callibration = Gdx.input.getRoll();
                        x_callibration = Gdx.input.getPitch();
                        count++;
                }
                cam.update();
                if(count_img == 48){
                        count_img = 0;
                }
                if((count_img %48)/16 == 0) img_temp = img;
                if((count_img %48)/16 == 1) img_temp = img2;
                if((count_img %48)/16 == 2) img_temp = img3;
                batch.add(img_temp);
                count_img++;
                Vector3 currentRotXYZ= new Vector3(Gdx.input.getPitch(),Gdx.input.getRoll(),Gdx.input.getAzimuth());
                currentRotXYZ = Math.abs(lastRotXYZ.x - currentRotXYZ.x) >= 320 || Math.abs(lastRotXYZ.y - currentRotXYZ.y) >=320 ||
Math.abs(lastRotXYZ.z- currentRotXYZ.z) >=320 ? currentRotXYZ: currentRotXYZ.interpolate(lastRotXYZ, 0.7f, Interpolation.circle);
                if((y_callibration-10f < current_y) && (current_y < y_callibration+10f)){
                img.setRotation(currentRotXYZ.x-current_x, currentRotXYZ.y-current_y, currentRotXYZ.z-current_z);
                img2.setRotation(currentRotXYZ.x-current_x, currentRotXYZ.y-current_y, currentRotXYZ.z-current_z);
                img3.setRotation(currentRotXYZ.x-current_x, currentRotXYZ.y-current_y, currentRotXYZ.z-current_z);
                }
                else if((x_callibration-10f < current_x) && (current_x < x_callibration+10f)){
                        //float rot = currentRotXYZ.z - current_z;
                        //rot = MathUtils.atan2(MathUtils.sin(rot * MathUtils.degRad), MathUtils.cos(rot * MathUtils.degRad
                                        //)) / MathUtils.PI * 70f / 4f;
                        //System.out.println(rot);

                img.setRotation( currentRotXYZ.z-current_z,currentRotXYZ.y-current_y,-currentRotXYZ.x+current_x );
                img2.setRotation( currentRotXYZ.z-current_z,currentRotXYZ.y-current_y,-currentRotXYZ.x+current_x );
                img3.setRotation( currentRotXYZ.z-current_z,currentRotXYZ.y-current_y,-currentRotXYZ.x+current_x );
                lastRotZ = currentRotXYZ.z - current_z;

                //float scale = MathUtils.clamp(Math.abs(rot/MathUtils.PI*4)+1, 1f, 2.2f);
                float scale = Math.abs((currentRotXYZ.z-current_z)/180*4)+1;
                img.setScaleX(scale);
                img2.setScaleX(scale);
                img3.setScaleX(scale);
                }
                else{
                img.setRotation(currentRotXYZ.x-current_x, currentRotXYZ.y-current_y, currentRotXYZ.z-current_z);
                img2.setRotation(currentRotXYZ.x-current_x, currentRotXYZ.y-current_y, currentRotXYZ.z-current_z);
                img3.setRotation(currentRotXYZ.x-current_x, currentRotXYZ.y-current_y, currentRotXYZ.z-current_z);
                }
                lastRotXYZ.set(currentRotXYZ.x, currentRotXYZ.y, currentRotXYZ.z);
                batch.flush();

                //New
            stage.act(Gdx.graphics.getDeltaTime());
            stage.draw();
        }
}
```