MIT App Inventor

Google Summer of Code

3D Components in App Inventor

An approach to enhance the functionality of MIT App Inventor
by adding support of 3D components using OpenGL

Preet Vadaliya

Table of Contents

Long Answers

Introduction

MIT App Inventor is a web application that allows everyone to build fully functional apps for smartphones and tablets by using a visual programming interface. It was and still is a game-changer in the world of android app development. It allows users to create android apps without having to write a single line of code. You can make a fully functional app by just dragging and dropping some blocks.

Interest in MIT App Inventor

Initially, I didn't know that MIT App Inventor is used to create fully-functional Android apps without coding, I was using it to create an app for home automation with Firebase and NodeMCU, then once I was looking at the component of the components pallet, I realized that I could use it to create an Android app and my great journey began with the MIT App Inventor.

Then day by day my interest grew. First I started with extension development, open-source contributions, and a lot of other stuff related to App Inventor.

I am now a community leader in the MIT App Inventor open source community and an external collaborator with the App Inventor project on GitHub.

Interest In Introductory Programming

I have been programming for the last few years and I have good knowledge of Java, Python, and JavaScript. I also have intermediate-level knowledge of Dart, C++, and C. I have some experience teaching MIT App Inventor. Currently, I am teaching App Inventor to my junior students.

Experience with teams, online developer communities, and large codebases

I have been working with the App Inventor team for the last year, and recently I was promoted as a community leader in the open-source MIT App Inventor global community. Also, I was the most successful external collaborator of the year 2021, with a total of 10 pull requests merged and 15 opened pull requests in the year 2021, I was also part of the Google Summer of Code 2021 and I am very well familiar with the development process of the MIT App Inventor.

Experience with Development Tools

	never used it	some experience	extensive experience
Java			✅
JavaScript			✅
Android development with the Java SDK			✅
Git/Github			✅
Automated testing (JUnit, phantomJS)			✅
JavaDoc			✅
Google Web Toolkit			✅
Google App Engine		✅	
Blockly			✅
Google Closure library and compiler		✅	

Project Proposal

Abstract

As we all know that we have a very good system for rendering and manipulating 2D shapes in the App using the canvas component, but currently, there is no rendering and manipulating system for 3D shapes, this project will add a new rendering and manipulating system for 3D shapes using OpenGL ES.

Use Cases

There are lots of use cases for 3D visualization, some of them are mentioned below...

1. 3D modeling & shape drawing
2. 3D Game development
3. It will open a new door of development in Augmented Reality (Using AR Core)

Impact on App Inventor's Educational Mission

Well, this feature can be of great help to the MIT App Inventor Education Mission...

1. As we all know we have a large number of student users, it can help them to create their school project, for example, the Solar System model can also help teachers in teaching.
2. We currently have some cool 2D games and drawing projects using the canvas component. This project can help us extend the functionality of the projects we already have.

Origin of Idea

The proposal idea is taken from one of the feature requests originally listed on GitHub; the original feature request was created by Evan W. Patton. Here is the link to the feature request: [check it out.]

Deliverables

The project is a bit long, we will not only add the new 3D surface (in which the object will be rendered), but we will also add some basic 3D shapes that will help beginners.

Similar to 2D drawing (Drawing and Animation), it will add a new component category for 3D visualization.

- ☐ New component category 3D Drawings and Animation.
- ☐ 3D shapes holder Space component.
- ☐ AR SurfaceView
- ☐ Realtime plane detection
- ☐ Basic 3D shapes.
    - ☐ Cube
    - ☐ Pyramid
    - ☐ Sprites
    - ☐ Other custom 3D shapes using coordinates.

Note

1. Adding a new Components Category for 3D elements is just an idea, I will finalize it after discussing with my mentors.
2. The name **Space** is just to indicate that it may change in actual implementation.
3. As discussed with `Evan Patton` He suggested that I create separate components for each 3D shape.

Implementation

Basic of OpenGL ES

Android supports OpenGL both through its framework API and the Native Development Kit(NDK); this proposal focuses on the Android framework API interfaces. There are two fundamental classes in the Android framework that let you help in rendering and manipulating graphics with the OpenGL ES API:

OpenGL requirements:

- Since we are using OpenGL we need to make some changes to the `AndroidManifest.xml` file.

1. Tell the system this app requires OpenGL ES.
2. Select the OpenGL Version.

Changes to AndroidManifest.xml

```
<!— Tell the system this app requires OpenGL ES 2.0. —>
<uses-feature android:glEsVersion="0x00020000" android:required="true" />
```

```
ActivityManager activityManager = (ActivityManager)
getSystemService(Context.ACTIVITY_SERVICE);
ConfigurationInfo configurationInfo =
activityManager.getDeviceConfigurationInfo();

configurationInfo.getGlEsVersion();
configurationInfo.reqGlEsVersion ⩾ 0x30000;
configurationInfo.reqGlEsVersion;
```

GLSurfaceView

This class is a View where we can draw and manipulate objects using OpenGL API calls, we can use this class by creating an instance of GLSurfaceView and adding

your Renderer to it, However, if we want to capture touch screen events, we should extend the GLSurfaceView class to implement the touch listeners.

`GLSurfaceView.Renderer`

This interface defines the methods required for drawing graphics in a GLSurfaceView, we must provide an implementation of this interface as a separate class and attach it to our GLSurfaceView.

Implementation in App Inventor

To implement OpenGL ES in App Inventor, we need to do two things ...

1. Implementation of GLSurfaceView.
2. Create our own renderer using GLSurfaceView.Renderer.

Space

Space will be a simple visible component that will capture all 3D shapes and 2D surfaces at runtime just like canvas in a 2D drawing.

`Execution Flow`

1. Space also extends to the AndroidViewComponent as another visible component.
2. When a new instance of Space is created, it will internally create an instance of AIGLSurfaceView and wrap it around a LinearLayout.
3. It will then set up some configurations like the OpenGL ES version, Renderer.

How about User Input?

- If we want an interactive application (like a game), we will typically subclass GLSurfaceView, because that's an easy way of obtaining input events.

AIGLSurfaceView.java

```
package com.google.appinventor.components.runtime;

import android.content.Context;
import android.opengl.GLSurfaceView;

public class AIGLSurfaceView extends GLSurfaceView {

  public AIGLSurfaceView(Context context) {
    super(context);
  }

}
```

AIGLRenderer.java

AIGLRenderer will be the main renderer for all 3D objects and AR surfaces, it will contain rendering properties like background color, line width, and background image (if we want to render an object on a real plane)

```
package com.google.appinventor.components.runtime;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.content.Context;
import android.opengl.GLSurfaceView;
import android.opengl.GLU;

public class MyGLRenderer implements GLSurfaceView.Renderer {
  Context context;

  public MyGLRenderer(Context context) {
    this.context = context;
  }

  @Override
  public void onSurfaceCreated(GL10 gl, EGLConfig config) {
  }

  @Override
```

```
  public void onSurfaceChanged(GL10 gl, int width, int height) {
  }

  @Override
  public void onDrawFrame(GL10 gl) {
  }
}
```

3D Objects

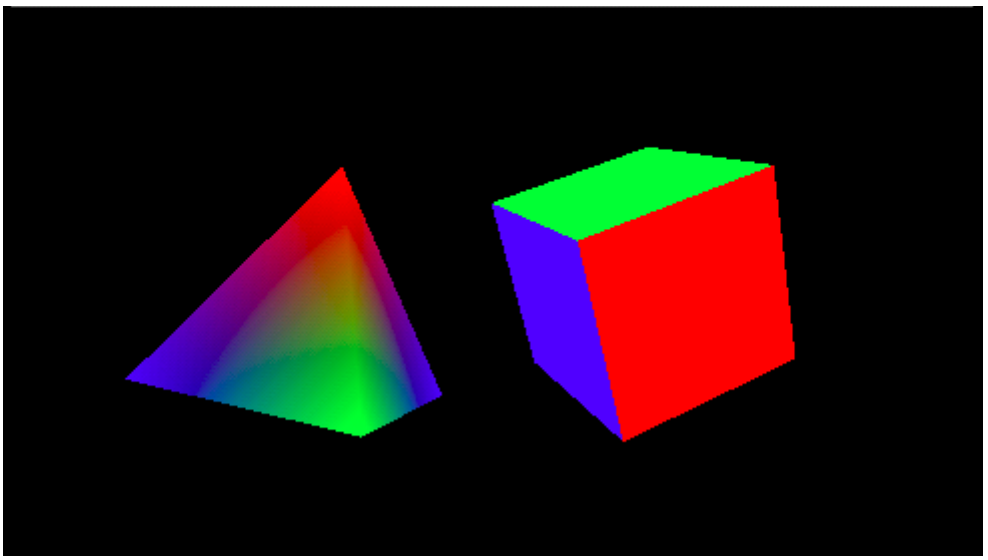
Similar to sprites in 2D canvas we will follow the same approach for 3D objects, every 3D object should extend the same class which has some common methods like rendering etc.

Cube Example

```
package com.google.appinventor.components.runtime;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import javax.microedition.khronos.opengles.GL10;

public class Cube {
   private FloatBuffer vertexBuffer;  // Buffer for vertex-array
   private int numFaces = 6;

   private float[][] colors = {  // Colors of the 6 faces
      {1.0f, 0.5f, 0.0f, 1.0f},  // 0. orange
      {1.0f, 0.0f, 1.0f, 1.0f},  // 1. violet
      {0.0f, 1.0f, 0.0f, 1.0f},  // 2. green
      {0.0f, 0.0f, 1.0f, 1.0f},  // 3. blue
      {1.0f, 0.0f, 0.0f, 1.0f},  // 4. red
      {1.0f, 1.0f, 0.0f, 1.0f}   // 5. yellow
   };

   private float[] vertices = {  // Vertices of the 6 faces
      // FRONT
      -1.0f, -1.0f,  1.0f,  // 0. left-bottom-front
       1.0f, -1.0f,  1.0f,  // 1. right-bottom-front
      -1.0f,  1.0f,  1.0f,  // 2. left-top-front
       1.0f,  1.0f,  1.0f,  // 3. right-top-front
      // BACK
       1.0f, -1.0f, -1.0f,  // 6. right-bottom-back
      -1.0f, -1.0f, -1.0f,  // 4. left-bottom-back
       1.0f,  1.0f, -1.0f,  // 7. right-top-back
      -1.0f,  1.0f, -1.0f,  // 5. left-top-back
      // LEFT
      -1.0f, -1.0f, -1.0f,  // 4. left-bottom-back
```

```
      -1.0f, -1.0f,  1.0f,  // 0. left-bottom-front
      -1.0f,  1.0f, -1.0f,  // 5. left-top-back
      -1.0f,  1.0f,  1.0f,  // 2. left-top-front
      // RIGHT
       1.0f, -1.0f,  1.0f,  // 1. right-bottom-front
       1.0f, -1.0f, -1.0f,  // 6. right-bottom-back
       1.0f,  1.0f,  1.0f,  // 3. right-top-front
       1.0f,  1.0f, -1.0f,  // 7. right-top-back
      // TOP
      -1.0f,  1.0f,  1.0f,  // 2. left-top-front
       1.0f,  1.0f,  1.0f,  // 3. right-top-front
      -1.0f,  1.0f, -1.0f,  // 5. left-top-back
       1.0f,  1.0f, -1.0f,  // 7. right-top-back
      // BOTTOM
      -1.0f, -1.0f, -1.0f,  // 4. left-bottom-back
       1.0f, -1.0f, -1.0f,  // 6. right-bottom-back
      -1.0f, -1.0f,  1.0f,  // 0. left-bottom-front
       1.0f, -1.0f,  1.0f   // 1. right-bottom-front
   };

   // Constructor - Set up the buffers
   public Cube() {
      // Setup vertex-array buffer. Vertices in float. An float has 4 bytes
      ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length * 4);
      vbb.order(ByteOrder.nativeOrder()); // Use native byte order
      vertexBuffer = vbb.asFloatBuffer(); // Convert from byte to float
      vertexBuffer.put(vertices);         // Copy data into buffer
      vertexBuffer.position(0);           // Rewind
   }

   // Draw the shape
   public void draw(GL10 gl) {
      gl.glFrontFace(GL10.GL_CCW);    // Front face in counter-clockwise
orientation
      gl.glEnable(GL10.GL_CULL_FACE); // Enable cull face
      gl.glCullFace(GL10.GL_BACK);    // Cull the back face (don't display)

      gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
      gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);

      // Render all the faces
```

```
      for (int face = 0; face < numFaces; face++) {
         // Set the color for each of the faces
         gl.glColor4f(colors[face][0], colors[face][1], colors[face][2],
colors[face][3]);
         // Draw the primitive from the vertex-array directly
         gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, face*4, 4);
      }
      gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
      gl.glDisable(GL10.GL_CULL_FACE);
   }
}
```



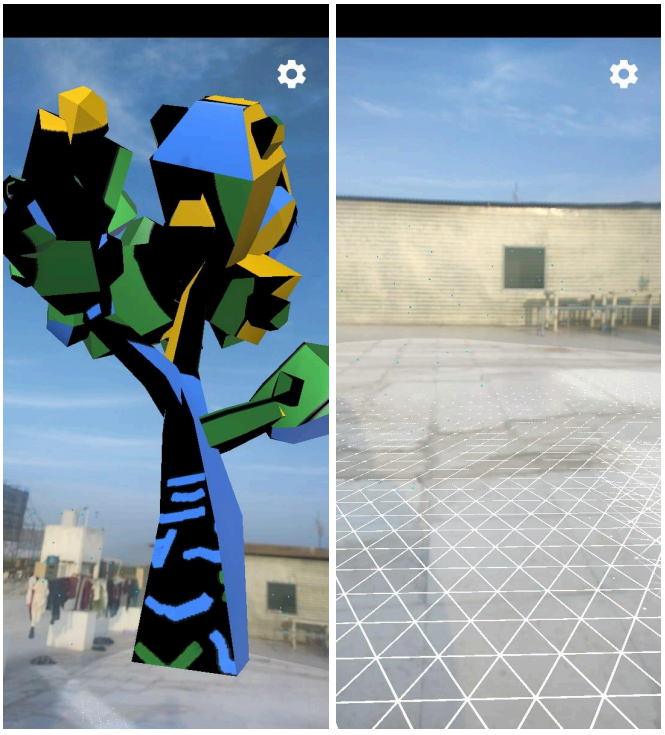
Output(This demo is implemented in App Inventor)[1]

[1] Here I am only mentioning the code for the Cube example but Pyramid is also working fine with App Inventor.

Discussion with Evan Patton

First of all I would like to thank Evan Patton Sir for guiding me to make this proposal. I discussed it with him almost everyday. Here is the summary of the talk...

1. We will use ARCore as much as possible to render 3D objects on a real plane.
2. We will try to follow Nichole Clarke's thesis in which she applied it for iOS.
3. Each 3D component will be a first order component.
4. First we will try to implement an example in Android Studio and then try to replicate it in App Inventor.
5. Work done before GSOC proposal
    a. Simple shape rendering on GLSurfaceView (Android & App Inventor).
    b. Custom 3D object on GLSurfaceView (App Inventor).
    c. Plane detection with ARCore (Android Studio).



Object placing and surface detection on Android Studio.

Application Prerequisites

Design Challenge

Problem

App Inventor has a Camera component to take pictures with your device's camera. The Paint Pot Extended with Camera tutorial is a good example of that. There used to be a property called 'useFront' that unfortunately had stopped working with newer releases of the Android OS. As explained in the reference doc, the property would allow an App Inventor developer to use the front camera as a default when opening up the Camera (note that the user would still be able to switch back to the back camera manually if wanted).

Solution

UseFront Property

The property useFront() uses `android.intent.extras.LENS_FACING_FRONT` intent extras **See this**, However, Google removed it in the release of Lollipop MR1 (Android SDK 22, Android 5.1), so it is not working with the new Android OS release.
This intent extra was added when the TakePicture()method was called (as seen in **Camara.java**)
Therefore, when trying to implement the camera's intent, the new phones did not recognize the extras as it was removed.

I RESULT

- No front camera is working by default.

SOLUTION

- Using a conditional check to know the SDK version and the new android.intent.extras.LENS_FACING_FRONT should get the job done.

```
if (useFront) {
  if (Build.VERSION.SDK_INT ⩾ Build.VERSION_CODES.LOLLIPOP_MR1)
    intent.putExtra("android.intent.extras.LENS_FACING_FRONT", 1);
  else
    intent.putExtra("android.intent.extras.CAMERA_FACING", 1);
}
```

Taking photos automatically

The current camera component invokes an Android intent to call an Android camera. This is not a self-baked camera, it just requires the Android OS to take a photo using the native camera.

Therefore, to avoid confusing users, I will create a new component. Users connect the camera to the normal phone's camera. However, having two separate components makes this easier, as they do not conflict with each other.

Add SurfaceView Component

Provides a dedicated drawing surface embedded inside of a view hierarchy, In order to create a new component in MIT App Inventor, the following files need to be altered...

Images.java

path : **appinventor/appengine/src/com/google/appinventor/client/**

This file is the one that links the component images to the image path usable in the GWT, which is presented in the Google App Engine.

```
@Source("com/google/appinventor/images/surfaceView.png")
ImageResource surfaceView();
```

OdeMessages.java

path : **appinventor/appengine/src/com/google/appinventor/client/**

OdeMessages.java is the file that contains all localizable strings in App Inventor.

surfaceView.png

path : **appinventor/appengine/src/com/google/appinventor/images/**

This is the image file for the new component.

YaVersion.java

path : **appinventor/components/src/com/google/appinventor/components/common/**

```
...

//For SURFACEVIEW_COMPONENT_VERSION
public static final int SURFACEVIEW_COMPONENT_VERSION = 1;

...
```

SurfaceView.java

path : **appinventor/components/src/com/google/appinventor/components/common/**

Source code of the component.

SimpleComponentDescriptor.java

path : **appinventor/appengine/src/com/google/appinventor/client/editor/simple/palette**

```
...

bundledImages.put("images/surfaceView.png", images.surfaceView());

...
```

Application Challenge

To complete the application challenge I have created this simple demo application using ML Kit which detects text from images.

The demo app has been successfully built through an online server as well as a local instance of App Inventor.

Here in the folder you will find a demo .aia and .apk file. **Folder Link**

Timeline

May 20 to June 12

COMMUNITY BONDING

- [ ] Discussion with mentors, power users, teachers, and students on what exactly they want from this project.

June 13 to July 24

PHASE - 1 CODING

- [ ] Basic stuff with OpenGL and AR-Core with android. In the first phase of the development I will focus more on detecting surfaces for AR-View.
- [ ] After that, The basic objects like cube and pyramid will be implemented on android studio and App Inventor.

July 25 to July 29

PHASE - 1 EVALUATIONS

- [ ] Presenting the work progress to mentors.
- [ ] Documenting the work progress.

July 30 to Sep 4

PHASE - 2 CODING

- [ ] This will be a main coding phase of the summer in which I will replicate all the features which will be completed during the first phase of the program.

Sep 5 to Sep 12

FINAL EVALUATIONS

- [ ] Presenting the work progress to mentors.
- [ ] Documenting the work progress.