RESEARCH ARTICLE OPEN ACCESS

Android NDK Graphics: Open GL ES Air Hockey Basis Application

Sabyasachi Patra *, Prathamesh Patel**, KarishmaVelisetty***, Mr. Abhay Kolhe****

*,**,***(Department of Computer Science, Mukesh Patel School of Technology Management and Engineering, NMIMS University, Mumbai, India.)

***(Faculty, Department of Computer Science, Mukesh Patel School of Technology Management and Engineering, NMIMS University, Mumbai, India.)

ABSTRACT

When it comes to beautiful visual rendering and games on the Android handsets that we use today, much of that credit has to be given to the various graphic libraries which come along with programming paradigms. Just like anything in programming and technology, there are good and bad ways to implement and get certain things done both at the front end and the backend. What the Android Native Development Kit (NDK) does is that it works alongside the Software Development Kit (SDK) and injects the native powers of any C/C++ application into your Android application which can be packaged as any normal application and run on an emulator/device of choice. The SDK, the NDK, the ADT and Eclipse are primarily what one requires to directly hit on towards Android Native Development. So now how does one enjoy the seamless graphics, multimedia, physics and games on an Android device? It is the Native API's which come as a result of relying on performance critical native code which makes this possible. In this paper we develop a basis game of Air Hockey, which is more of a concept game made using the technological paradigm of the NDK in Android Programming. Right from defining the structure of the table up to defining and implementing the shaders and 3D rendering, almost every important aspect has been researched and put into working through this paper.

Keywords–Air Hockey, Android, Java, OpenGL, Drawing, Shaders, Graphics API, Android 3D Rendering

I. INTRODUCTION

To play a game of air hockey, we need a long rectangular table with two goals (one on each end), a puck, and two mallets to strike the puck with. Each round starts with the puck placed in the middle of the table. Each player then tries to strike the puck into the opponent's goal while preventing the opponent from doing the same. The first player to reach seven goals wins the game.

II. DEFINING THE STRUCTURE OF OUR AIR HOCKEY TABLE

Before drawing the table to the screen, OpenGL needs to know what to draw. The drawing of the structure of the table needs to be done in a form that OpenGL understands. The vertex is the inception of everything in OpenGL. Drawing of any structure starts by defining the vertex for that particular structure.

III. INTRODUCING VERTICES

A vertex is simply a point representing one corner of a geometric object, with various attributes associated with that point. The most important attribute is the position, which represents where this vertex is located in space.

IV. BUILDING THE TABLE WITH VERTICES

When choosing a basic shape it should be kept in mind that the structure should match the objective of the game. So the most suitable shape to represent an air hockey table can be taken as a rectangle. Since a rectangle has four corners, the structure has four vertices.

The rectangle depicted here is taken in the form of a two-dimensional object, so each vertex would need a position, with a coordinate for each dimension i.e. x and y.

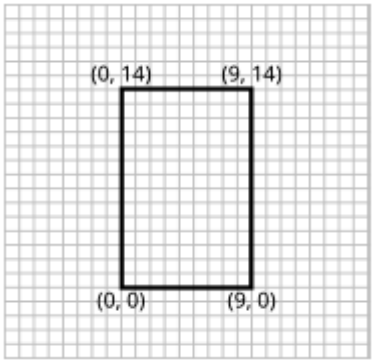

Fig.1: Defining Vertices in Code

```
publicAirHockeyRenderer() {
float[] tableVertices = {
0f, 0f,
0f, 14f,
9f, 14f,
9f, 0f
};
}
```

Points and lines are for certain effects, but only triangles are used to construct an entire scene of complex objects and textures. Triangles in OpenGL are builtby grouping individual vertices together, and then OpenGL can be instructed how to connect the dots.

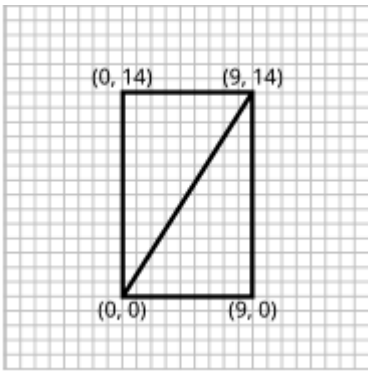

Fig.2: Defining the structure of our air hockey table

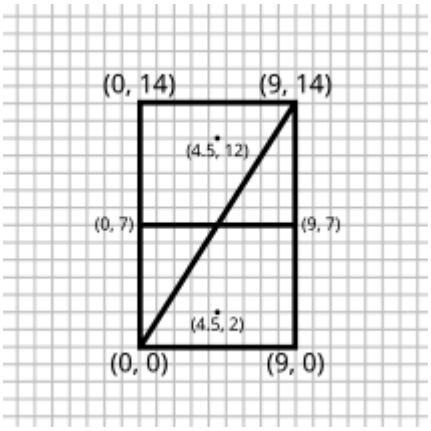

Fig.3: Adding the center line and two mallets

V. MAKING THE DATA ACCESSIBLE TO OPENGL

The OpenGL now needs to access the vertices that are created. There are two main concepts involved here:

5.1 Dalvik VM

When we compile and run our Java code in the emulator or on a device, it runs through the Dalvik virtual machine. Code running in thisvirtual machine has no direct access to the native environment other than via special APIs.

5.2 Garbage Collector

The DalvikVM also uses *garbage collection*. When the VM detects that a variable, object, or a memory address is no longer being used, it will release that memory so that it can be reused.

VI. MAPPING COLORS TO THE DISPLAY:

OpenGL uses the generic additive RGB color model, which works with just the three primary colors - Red, Green, and Blue. Many colors can be created by mixing these primary colors together in various proportions.

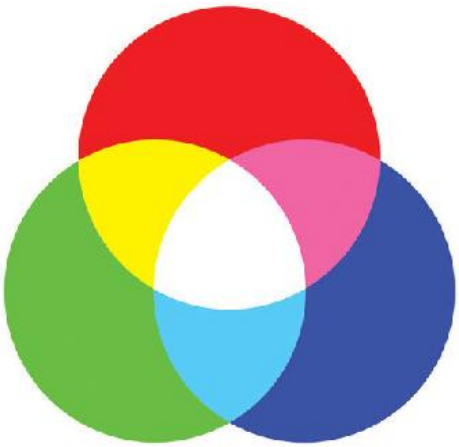

Fig.4: RGB color model

OpenGL assumes that these colors all have a linear relationship with eachother: a red value of 0.5 should be twice as bright as a red value of 0.25, anda red value of 1 should be twice as bright as a red value of 0.5. These primarycolors are clamped to the range [0, 1], with 0 representing the absence of thatparticular primary color and 1 representing the maximum strength for thatcolor.

The important steps that need to be followed are given below:

- Loading Shaders
- Loading Text from a Resource
- Reading in the Shader Code
- Compiling Shaders

```
public static String readTextFileFromResource(Context context, intresourceId) {
StringBuilderbody = new StringBuilder();
try{
InputStreaminputStream = context.getResources().openRawResource(resourceId );
InputStreamReaderinputStreamReader = report erratum • discuss newInputStreamReader(inputStream);
BufferedReaderbufferedReader = new BufferedReader(inputStreamReader);
String nextLine;
while((nextLine = bufferedReader.readLine()) != null) {
body.append(nextLine);
body.append('\n');
}
} catch (IOException e) {
throw new RuntimeException(
"Could not open resource: "+ resourceId, e);
} catch (Resources.NotFoundExceptionnfe) {
throw new RuntimeException("Resource not found: " + resourceId, nfe);
}
returnbody.toString();
}
```

Further a new helper class is createdthat creates a new OpenGL shader object, compile theshader code, and return the shader.

```
finalint[] compileStatus = new int[1];
glGetShaderiv(shaderObjectId, GL_COMPILE_STATUS, compileStatus, 0);
```

Two things are achieved by the above code. Uploading and Compiling the Shader Source Code and retrieving the Compilation Status

Compiling the Shaders from the Renderer class is done using the following three functions.

- compileShader()
- compileVertexShader()
- compileFragmentShader()

The following code implements the process of linking shaders together into an OpenGL program at the same time verifying the link status and returning the program object ID

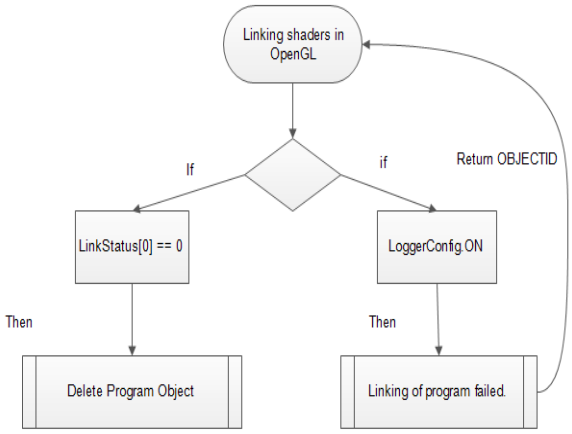

Fig 5. Flowchart which verifies link status and returns object ID

Now the vertices need to be defined in the program with the final connections and Validations. In the following code drawing to the screen and table takes place.

```
float[] tableVerticesWithTriangles = {
// Triangle 1
0f, 0f,
9f, 14f,
0f, 14f,
// Triangle 2
0f, 0f,
```

```
9f, 0f,
9f, 14f,
// Line 1
0f, 7f,
9f, 7f,
// Mallets
4.5f, 2f,
4.5f, 12f
};
```

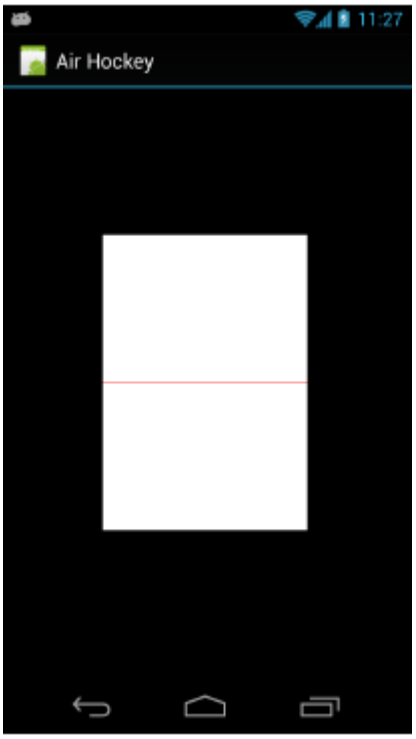

Fig.6: Intermediate Result

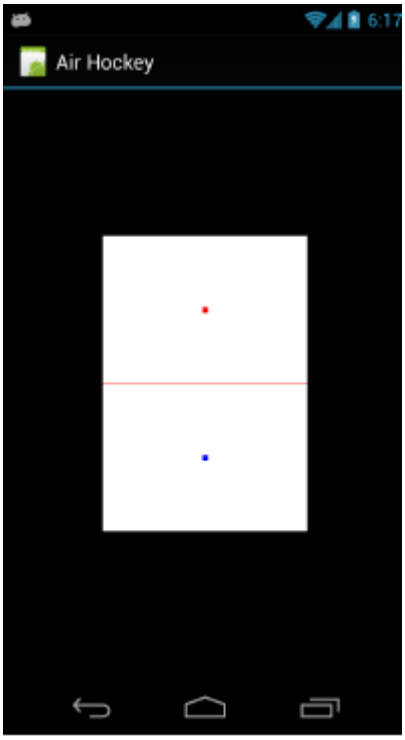

Fig.7: Table including the mallets

VII. SMOOTH SHADING

Smooth Shading Is Done Between Vertices. OpenGL provides an option to smoothly blend the colors at each vertex across a line or across the surface of a triangle. This type of shading can be used to make the table appear brighter in the middle and dimmer toward the edges, just to give an effect of a light hovering over the middle of the table.

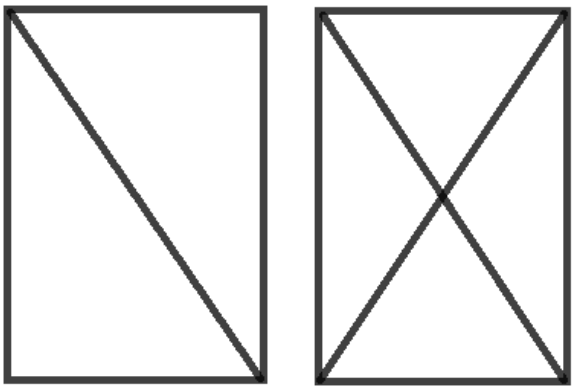

Fig.8: Smooth shading

VIII. INTRODUCING TRIANGLE FANS

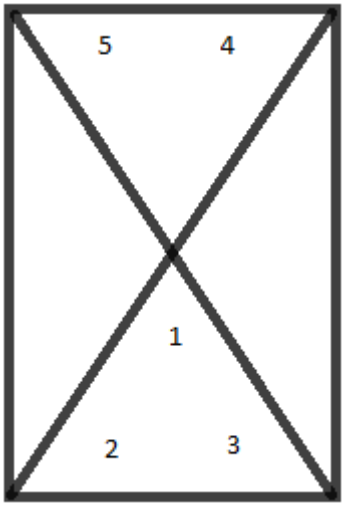

Fig.9: Triangle Fans

This involves the following steps:

- Color is added to each vertex taking into account interpolation and gradients.
- We add a new attribute to the vertex data and vertex shaderand OpenGL reads this data by using a stride.
- Interpolate this data across the surface of a triangle.

We need to adjust the coordinate space so that it takes the screen shape into account, and one way to do this is to keep the smaller range fixed to [-1, 1] and adjust the larger range in proportion to the screen dimensions.

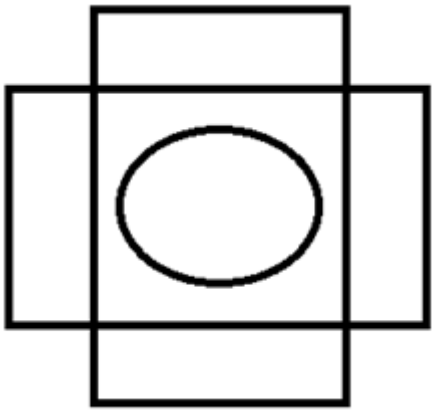

Fig.10: Adjusting the screen's aspect ratio

IX. ENTERING THE THIRD DIMENSION

The following are the two most vital components:

- OpenGL's *perspective division* and the components to create the illusion of 3D on a 2D screen.
- Setting up a perspective projection so that we can see the table in 3D.

Homogenous coordinates are used because of the perspective division and coordinates in clip space are often referred to as homogenous coordinates.

(1, 1, 1, 1), (2, 2, 2, 2), (3, 3, 3, 3), (4, 4, 4, 4), (5, 5, 5, 5)

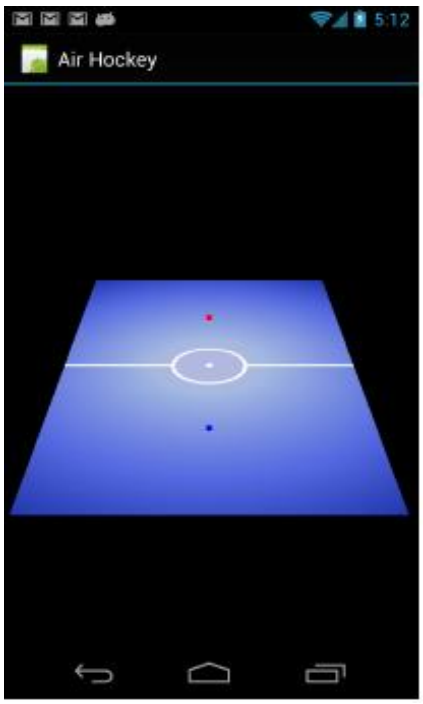

Fig 11. Air Hockey table with filtered texture

Fig 12. Transformation steps and different co-ordinate spaces

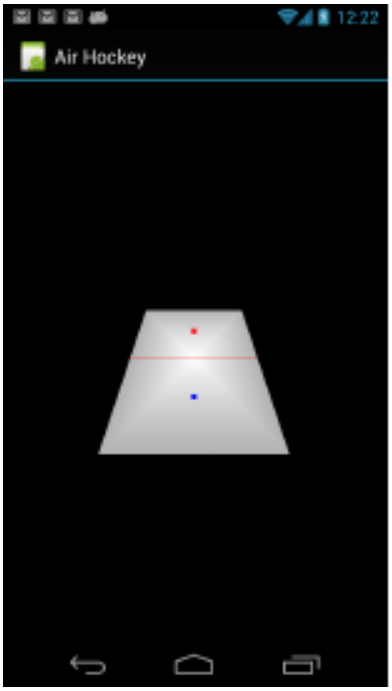

Fig 13. The first look 3D view of the table

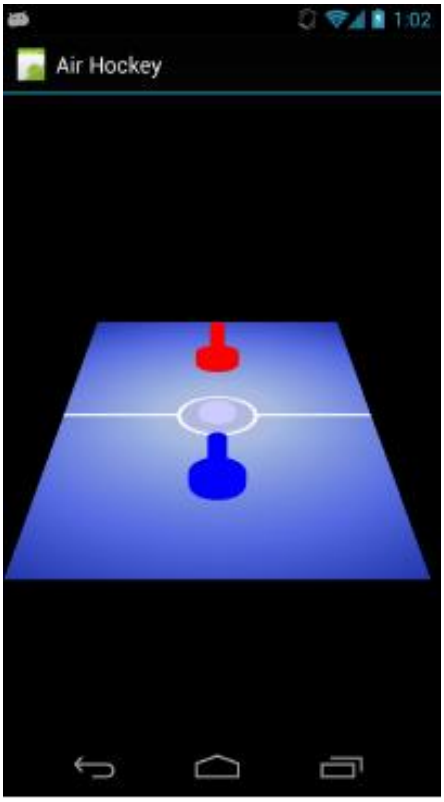

Fig 14. The final layout of the NDK Concept game

X. ADDING TOUCH FEEDBACK AND INTERACTION

```
finalAirHockeyRendererairHockeyRenderer = new AirHockeyRenderer(this);
if(supportsEs2) {
// ...
  glSurfaceView.setResnderer(airHockeyRenderer);
```

In Android, we can listen in on a view's touch events by calling setOnTouchListener().When a user touches that view, we'll receive a call to onTouch().There is also a need to extend a Two-Dimensional Point into a Three-Dimensional Line

XI. ADDING COLLISION DETECTION

This involves keeping the player's mallet within bounds and also adding velocity and direction to the mallet. Now we can add some code to smack the puck with the mallet. To get an idea of how the puck should react, information is needed about the speed of the mallet and the direction on which it is moving.

The puck doesn't slow down because it still has a uniform velocity.That doesn't look realistic, so friction needs to be added to the code to slow the puck over time. The following lines of code deal with this problem.

puckVector = puckVector.scale(0.99f);

puckVector = puckVector.scale(0.9f);

XII. FUTURE SCOPE

Some basic extensions to the above implementation could bea bowling game where a ball gets flung by the player and you watch that ball head down the lane to knock out the pins at the far side. Touch interaction is what really sets mobile games apart once the technology is in place, and there are many ways to be imaginative.Usage of Physics Libraries can make the tasks a little more seamless and basic sounds can be included using OpenSL. Artificial Intelligence can also be implemented to make the game more interactive with the user. A menu can be included to provide the user with more options.

ACKNOWLEDGEMENT

This research paper is made possible through the help and support of many people both inside and outside the domain of Engineering and Science.
Especially, please allow me to dedicate my acknowledgment of gratitude towards our college Librarian Mr.Pradip Das and his team. I would also like to thank Mr.AnandGawadekar of the NMIMS IEEE Committee due to which we could get all requested references seamlessly without any trouble and on time.

A sincere thanks to our college and Computer Science department H.O.D. Professor Dhirendra Mishra for allowing us to enter the invaluable field of research in our so important final year of B.Tech. Our mentor, Mr.AbhayKolhe also guided us on a weekly basis at periodic meets with him. Finally, we would like to thank our parents who always encouraged us to do as much research we could do in our capacity in the final year and extend an outside support whenever and wherever required.

CONCLUSION

In this paper we started with the basics of how an Android OpenGL based Air Hockey application should be structured and what it is expected to do. The basics which enable us to grab and move a mallet around our fingers and bounce the puck on the table were covered. A research on the semantics of a game with such features and technology would enable us to go further ahead and deeper into the physics and more complicated mathematics involved.

REFERENCES

Web Resources:

[1] Ed Burnette. *Hello, Android: Introducing Google's Mobile Development Platform, Third Edition*. The Pragmatic Bookshelf, Raleigh, NC and Dallas,TX, 2010.
[2] Android NDK group (http://groups.google.com/group/android-ndk)
[3] The Android Developer BlogSpot (http://android-developers.blogspot.com/)
[4] Google Code (http://code.google.com/hosting/) for lots of NDK exampleapplications.
[5] Stack Overflow (http://stackoverflow.com/)

Journal papers:

[6] Walter Binder, JarleHulaas and Philippe Moret "A Quantitative Evaluation of the Contribution of Native Code to Java Workloads" Workload Characterization, 2006 IEEE International Symposium pages 201-209.
[7] M. B. Dillecourt, H. Samet and M. Tamminen, "A general approach to connected component labeling for arbitrary image representations", Journal of the ACM, vol. 39, no. 2, (1992), pp. 253-280.
[8] Sangchul Lee and Jae WookJeon "Evaluating Performance of Android Platform Using Native C for Embedded Systems" Control Automation and Systems (ICCAS), 2010 International Conference pages 1160 - 1163.

Books:

[9] Ken Arnold, James Gosling, and David Holmes. The Java Programming
[10] Language. Prentice Hall, Englewood Cliffs, NJ, 4th, 2005.
[11] Joshua Bloch. Effective Java. Addison-Wesley, Reading, MA, 2008.
[12] Ed Burnette. Hello, Android: Introducing Google's Mobile Development
[13] Platform, Third Edition. The Pragmatic Bookshelf, Raleigh, NC and Dallas, Bruce Eckel. Thinking in Java. Prentice Hall, Englewood Cliffs, NJ, Fourth, 2006.Mario Zechner. Beginning Android Games. Apress, New York City, NY, 2012.