

**TRƯỜNG ĐẠI HỌC TÂY ĐÔ**  
**KHOA KỸ THUẬT – CÔNG NGHỆ**



**NIÊN LUẬN 3**  
**TÌM HIỂU THƯ VIỆN ĐỒ HỌA OPENGL**  
**VÀ VIẾT ỨNG DỤNG**

***Giảng viên hướng dẫn***

*Nguyễn Chí Cường*

***Sinh viên thực hiện***

*Nguyễn Văn Minh Đạt* – 13D480201006

*Nguyễn Thị Thảo Nhu* – 13D480201070

*Nguyễn Thị Phương Thảo* – 13D480201092

**Cần Thơ, 2016**

**TRƯỜNG ĐẠI HỌC TÂY ĐÔ**  
**KHOA KỸ THUẬT – CÔNG NGHỆ**



**NIÊN LUẬN 3**  
**TÌM HIỂU THƯ VIỆN ĐỒ HỌA OPENGL**  
**VÀ VIẾT ỨNG DỤNG**

***Giảng viên hướng dẫn***

*Nguyễn Chí Cường*

***Sinh viên thực hiện***

*Nguyễn Văn Minh Đạt* – 13D480201006

*Nguyễn Thị Thảo Nhu* – 13D480201070

*Nguyễn Thị Phương Thảo* – 13D480201092

**Cần Thơ, 2016**

# Nhận xét của Giảng viên hướng dẫn

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

*Cần Thơ, ngày ... tháng ... năm 2016*

# Lời cảm ơn

Chúng em xin chân thành cảm ơn Thầy đã nhiệt tình hướng dẫn nhóm chúng em trong quá trình thực hiện niên luận. Đây là một đề tài mới và khó đối với chúng em, trong quá trình thực hiện niên luận chúng em đã cố gắng nhưng không tránh khỏi sai sót và gặp hạn chế, nhờ thầy nhận xét cho chúng em để bài niên luận hoàn thiện.

Cảm ơn cha mẹ, anh chị và bạn bè đã động viên, giúp đỡ chúng con, chúng em trong quá trình thực hiện niên luận.

Chúng em xin chân thành cảm ơn!

Cần Thơ, ngày 11 tháng 12 năm 2016

Nhóm SVTH

# Mục lục

<b>Chương 1</b>	<b>Giới thiệu</b>	<b>1</b>
1.1	Đặt vấn đề . . . . .	1
1.2	Mục tiêu . . . . .	2
1.3	Phương pháp thực hiện . . . . .	2
<b>Chương 2</b>	<b>Cài đặt môi trường lập trình</b>	<b>3</b>
2.1	Ngôn ngữ lập trình Java . . . . .	3
2.2	Các phần mềm cần thiết . . . . .	3
2.3	Cài đặt JDK . . . . .	4
2.4	Cài đặt Android Studio . . . . .	8
2.5	Cài đặt Genymotion . . . . .	12
<b>Chương 3</b>	<b>Tìm hiểu OpenGL ES 2.0 trên Android và Viết ứng dụng</b>	<b>19</b>
3.1	Kiểu dữ liệu trong OpenGL ES . . . . .	19
3.2	Sử dụng OpenGL ES 2.0 trong Android . . . . .	19
3.2.1	Khai báo sử dụng OpenGL ES 2.0 trong file Manifest . . . . .	20
3.2.2	Khởi tạo file Activity khi sử dụng OpenGL ES 2.0 . . . . .	21
3.2.3	Xây dựng lớp GLSurfaceView . . . . .	21
3.2.4	Xây dựng lớp Renderer . . . . .	22
3.2.5	Nội dung của các file chương trình . . . . .	23
3.3	Viết ứng dụng vẽ một số đối tượng hình học . . . . .	24
3.3.1	Vẽ đối tượng là tam giác . . . . .	24
3.3.2	Vẽ đối tượng là khối lập phương . . . . .	32
3.3.3	Vẽ đối tượng là khối cầu . . . . .	47
<b>Chương 4</b>	<b>Kết quả</b>	<b>55</b>
4.1	Kết quả đạt được . . . . .	55
4.2	Hạn chế . . . . .	55
4.3	Hướng phát triển . . . . .	55
	<b>Tài liệu tham khảo</b>	<b>57</b>

# Danh sách hình ảnh

2.1	Xác nhận đồng ý điều khoản tải JDK về máy . . . . .	4
2.2	Download JDK về máy . . . . .	4
2.3	Mở file jdk-8u111-windows-x64.exe để cài đặt JDK . . . . .	5
2.4	Cài đặt JDK trên hệ điều hành Windows . . . . .	5
2.5	Vị trí của thư mục jdk1.8.0_111 . . . . .	6
2.6	Mở cửa sổ System trên Windows . . . . .	6
2.7	Thiết lập đường dẫn cho biến JAVA_HOME . . . . .	7
2.8	Thêm biến JAVA_HOME vào cửa sổ System variables . . . . .	8
2.9	Download Android Studio từ trang chủ . . . . .	8
2.10	Xác nhận đồng ý các điều khoản khi Download Android Studio . . . . .	9
2.11	Mở file android-studio-bundle-145.3360264-windows.exe để cài đặt . . . . .	9
2.12	Cài đặt Android Studio trên hệ điều hành Windows . . . . .	10
2.13	Cài đặt Android Studio Wizard . . . . .	11
2.14	Trang chủ của Genymotion . . . . .	12
2.15	Đăng nhập tài khoản trên Genymotion . . . . .	13
2.16	Download Genymotion về máy . . . . .	13
2.17	Cài đặt Genymotion trên Windows . . . . .	14
2.18	Cài đặt Oracle VM VirtualBox trên Windows . . . . .	15
2.19	Hoàn thành cài đặt Genymotion và Oracle VM VirtualBox trên Windows . . . . .	16
2.20	Tải máy ảo Genymotion về máy . . . . .	17
2.21	Mở máy ảo Genymotion để kiểm tra . . . . .	18
3.1	Sử dụng OpenGL ES 2.0 trên Android . . . . .	24
3.2	Sử dụng OpenGL ES 2.0 vẽ đối tượng là một tam giác . . . . .	32
3.3	Vẽ và xoay hình lập phương sử dụng OpenGL ES 2.0 trên Android . . . . .	42
3.4	Tự động xoay khối lập phương sử dụng OpenGL trên Android . . . . .	47
3.5	Xoay khối cầu sử dụng OpenGL trên Android . . . . .	54

# Danh sách bảng

1.1	Phương pháp thực hiện niên luận . . . . .	2
3.1	Kiểu dữ liệu trong OpenGL ES . . . . .	19

# Chương 1

## Giới thiệu

### 1.1 Đặt vấn đề

OpenGL – Open Graphics Library: là một trong những thư viện đồ họa phổ biến, được giới thiệu đầu tiên vào năm 1991, dùng làm tiêu chuẩn kỹ thuật đồ họa với mục đích tạo ra giao diện lập trình ứng dụng đồ họa 2D và 3D, cung cấp khoảng 250 hàm để vẽ các đối tượng phức tạp từ những hàm đơn giản (điểm, đoạn thẳng, đa giác,...).

OpenGL được ứng dụng rộng rãi trong các trò chơi điện tử, xây dựng các mô hình trong mô phỏng khoa học và thông tin, được dùng để phát triển trò chơi.

OpenGL không những chạy được trên máy tính mà còn được mở rộng sang di động, được dùng phát triển nhiều game 2D, 3D trên các hệ điều hành Android, iOS,...

OpenGL ES – Open Graphics Library for Embedded System: là một phiên bản của OpenGL dành cho các hệ thống nhúng, tiêu biểu là các thiết bị di động. OpenGL ES gọn nhẹ và nhẹ hơn so với OpenGL do OpenGL ES được lược bớt một số thành phần để phù hợp hơn với các thiết bị di động có cấu hình thấp hơn máy tính.

Các phiên bản của OpenGL ES hỗ trợ cho hệ điều hành Android như sau:

- OpenGL ES 1.0 và 1.1: hỗ trợ Android từ 1.0 trở lên.
- OpenGL ES 2.0: hỗ trợ Android 2.2 (API level 8) trở lên.
- OpenGL ES 3.0: hỗ trợ từ Android 4.3 (API level 18) trở lên.
- OpenGL ES 3.1: hỗ trợ từ Android 5.0 (API level 21) trở lên.

Trong phạm vi đề tài niên luận, nhóm chúng em tìm hiểu về cách sử dụng thư viện đồ họa OpenGL thông qua thư viện OpenGL ES 2.0, một phiên bản dành cho hệ thống nhúng, tiêu biểu là thiết bị di động sử dụng hệ điều hành Android.



## 1.2 Mục tiêu

Trong quá trình thực hiện niên luận về đề tài “*Tìm hiểu về thư viện đồ họa OpenGL*”, mục tiêu chúng em đề ra để đạt được là:

- Biết được một số lĩnh vực cần đến OpenGL: phát triển game, xây dựng các mô hình mô phỏng,...
- Biết cách vẽ một số đối tượng cơ bản với các hàm của OpenGL.
- Biết cách nhúng mã OpenGL (thông qua OpenGL ES 2.0) vào các ứng dụng Android.

## 1.3 Phương pháp thực hiện

- Nội dung và phương pháp thực hiện niên luận của nhóm được trình bày tóm tắt trong bảng 1.1.

<i>STT</i>	<i>Nhiệm vụ</i>	<i>Công cụ</i>
1	Cài đặt môi trường lập trình OpenGL trên Android	JDK, Android Studio, Genymotion
2	Tìm hiểu lý thuyết về OpenGL ES 2.0 trên Android	OpenGL ES 2.0
3	Thực hành một số lệnh cơ bản của OpenGL ES 2.0 trên Android	JDK, Android Studio, Genymotion
4	Quản lý mã nguồn chương trình với Git	Git và GitHub
5	Viết báo cáo hoàn thành niên luận	L <sup>A</sup> T <sub>E</sub> X với T <sub>E</sub> XMaker và T <sub>E</sub> XLive

Bảng 1.1: Phương pháp thực hiện niên luận

- Toàn bộ mã nguồn của bài báo cáo được đưa lên GitHub với địa chỉ:  
<https://github.com/minhdatvnus/nienluan3>

## Chương 2

# Cài đặt môi trường lập trình

### 2.1 Ngôn ngữ lập trình Java

Ngôn ngữ lập trình được sử dụng trong đề tài là ngôn ngữ Java. Một chương trình Java có thể được định nghĩa như là một tập hợp các đối tượng giao tiếp thông qua cách gọi những phương thức. Do đó, yêu cầu cần thiết với lập trình Java là phải biết cách lập trình hướng đối tượng – Object-Oriented Programming.

Khi cài đặt Android Studio, thư viện đồ họa OpenGL ES được tích hợp sẵn trong gói phần mềm, do đó chúng ta cần cài đặt JDK và Android Studio là có thể nhúng mã OpenGL vào các ứng dụng Android. Các ứng dụng trên Android được viết bằng ngôn ngữ lập trình Java.

Do phạm vi đề tài, nhóm chúng em không đi sâu vào trình bày về ngôn ngữ lập trình Java. Cú pháp các lệnh của ngôn ngữ Java sẽ được giải thích trong các ví dụ khi sử dụng OpenGL trên Android.

Chúng ta có thể biên dịch chương trình viết bằng Java trên website: [browxy.com](http://browxy.com).

### 2.2 Các phần mềm cần thiết

Cần thực hiện cài đặt các phần mềm sau:

- Phần mềm JDK: tạo máy ảo Java, dùng biên dịch mã nguồn viết bằng Java.
- Phần mềm Android Studio: công cụ lập trình Android.
- Phần mềm Genymotion: phần mềm giả lập điện thoại trên máy tính.

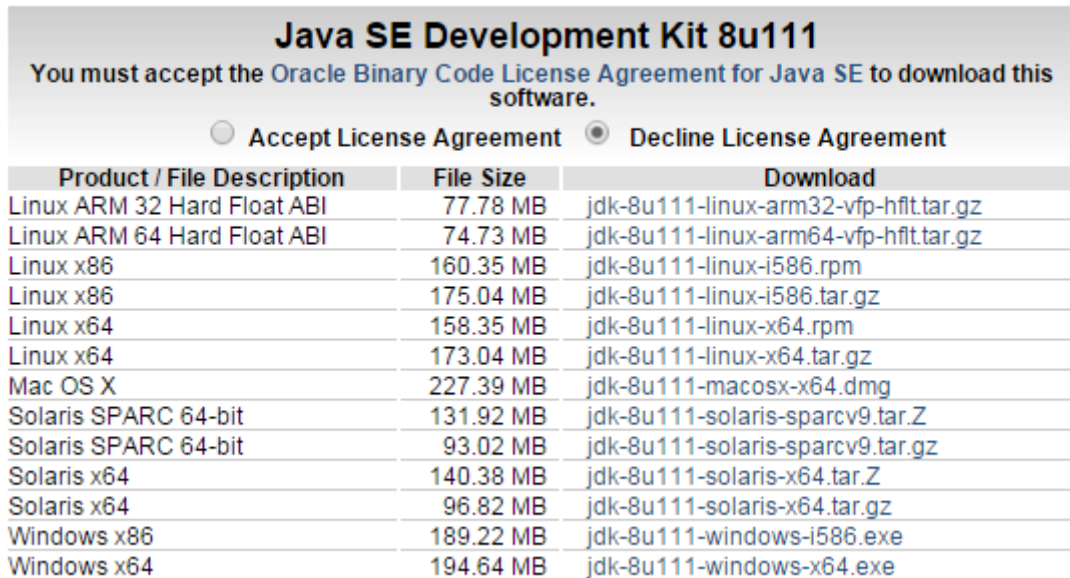
Trong phần mềm Android Studio cũng có máy ảo nhưng chạy chậm hơn so với máy ảo trong phần mềm Genymotion.

Các cài đặt các phần mềm trên được trình bày trong các mục 2.3 – 2.5.

## 2.3 Cài đặt JDK

Thực hiện quá trình cài đặt theo các bước bên dưới:

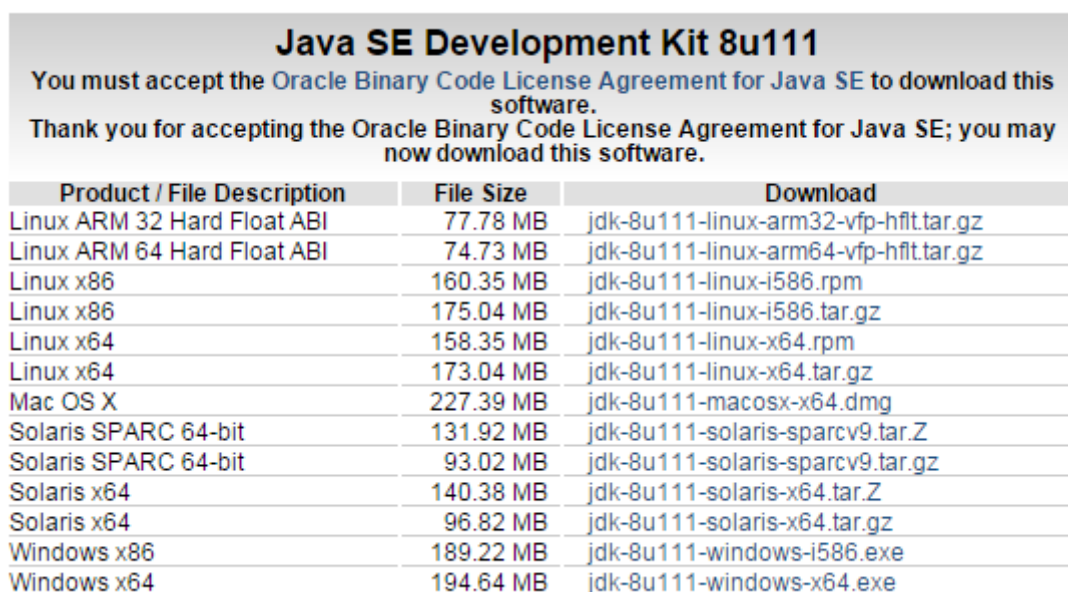
- *Bước 1:* Tải phần mềm JDK về máy.
  - Truy cập địa chỉ: <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>, được giao diện như hình 2.1, click chọn Accept License Agreement.



Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.78 MB	<a href="#">jdk-8u111-linux-arm32-vfp-hflt.tar.gz</a>
Linux ARM 64 Hard Float ABI	74.73 MB	<a href="#">jdk-8u111-linux-arm64-vfp-hflt.tar.gz</a>
Linux x86	160.35 MB	<a href="#">jdk-8u111-linux-i586.rpm</a>
Linux x86	175.04 MB	<a href="#">jdk-8u111-linux-i586.tar.gz</a>
Linux x64	158.35 MB	<a href="#">jdk-8u111-linux-x64.rpm</a>
Linux x64	173.04 MB	<a href="#">jdk-8u111-linux-x64.tar.gz</a>
Mac OS X	227.39 MB	<a href="#">jdk-8u111-macosx-x64.dmg</a>
Solaris SPARC 64-bit	131.92 MB	<a href="#">jdk-8u111-solaris-sparcv9.tar.Z</a>
Solaris SPARC 64-bit	93.02 MB	<a href="#">jdk-8u111-solaris-sparcv9.tar.gz</a>
Solaris x64	140.38 MB	<a href="#">jdk-8u111-solaris-x64.tar.Z</a>
Solaris x64	96.82 MB	<a href="#">jdk-8u111-solaris-x64.tar.gz</a>
Windows x86	189.22 MB	<a href="#">jdk-8u111-windows-i586.exe</a>
Windows x64	194.64 MB	<a href="#">jdk-8u111-windows-x64.exe</a>

Hình 2.1: Xác nhận đồng ý điều khoản tải JDK về máy

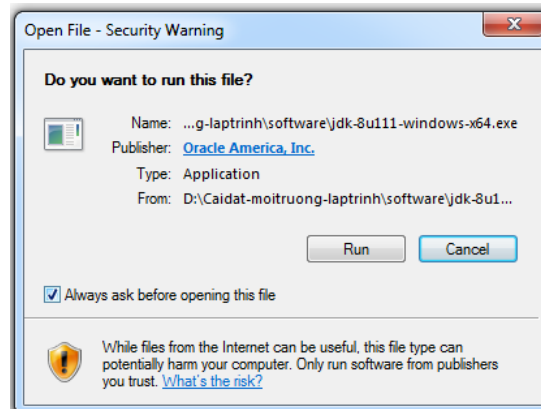
- Được giao diện như hình 2.2: lựa chọn phiên bản cài đặt phù hợp với hệ điều hành và cấu hình máy để tải về máy. Ví dụ: lựa chọn Windows x64 ứng với hệ điều hành Windows 64bit.



Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.78 MB	<a href="#">jdk-8u111-linux-arm32-vfp-hflt.tar.gz</a>
Linux ARM 64 Hard Float ABI	74.73 MB	<a href="#">jdk-8u111-linux-arm64-vfp-hflt.tar.gz</a>
Linux x86	160.35 MB	<a href="#">jdk-8u111-linux-i586.rpm</a>
Linux x86	175.04 MB	<a href="#">jdk-8u111-linux-i586.tar.gz</a>
Linux x64	158.35 MB	<a href="#">jdk-8u111-linux-x64.rpm</a>
Linux x64	173.04 MB	<a href="#">jdk-8u111-linux-x64.tar.gz</a>
Mac OS X	227.39 MB	<a href="#">jdk-8u111-macosx-x64.dmg</a>
Solaris SPARC 64-bit	131.92 MB	<a href="#">jdk-8u111-solaris-sparcv9.tar.Z</a>
Solaris SPARC 64-bit	93.02 MB	<a href="#">jdk-8u111-solaris-sparcv9.tar.gz</a>
Solaris x64	140.38 MB	<a href="#">jdk-8u111-solaris-x64.tar.Z</a>
Solaris x64	96.82 MB	<a href="#">jdk-8u111-solaris-x64.tar.gz</a>
Windows x86	189.22 MB	<a href="#">jdk-8u111-windows-i586.exe</a>
Windows x64	194.64 MB	<a href="#">jdk-8u111-windows-x64.exe</a>

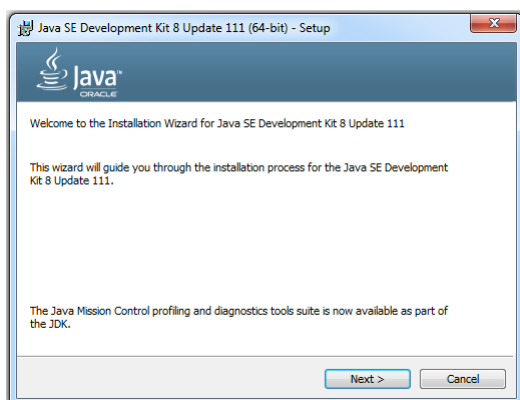
Hình 2.2: Download JDK về máy

- **Bước 2:** Thực hiện cài đặt phần mềm JDK.
  - Mở phần mềm JDK vừa tải ở *bước 1*, (ví dụ file `jdk-8u111-windows-x64.exe`) được giao diện như hình 2.3. Chọn **Run** để bắt đầu cài đặt JDK.

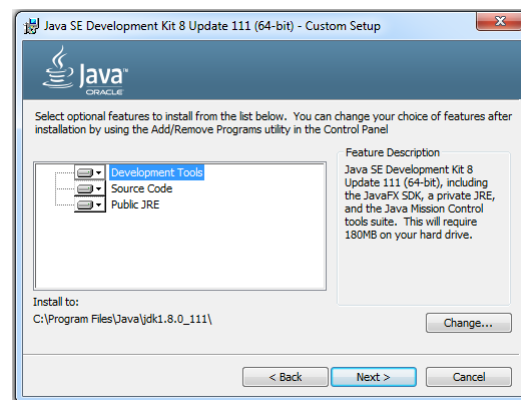


Hình 2.3: Mở file `jdk-8u111-windows-x64.exe` để cài đặt JDK

- Được giao diện như hình 2.4a: chọn **Next**.
- Được giao diện như hình 2.4b: để như mặc định và chọn **Next**.
- Đợi quá trình cài đặt (hình 2.4c) hoàn thành như hình 2.4d: chọn **Close**.



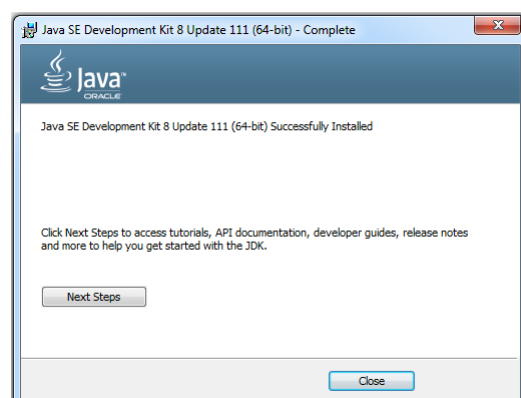
(a) Chọn **Next**



(b) Chọn **Next**



(c) Quá trình cài đặt

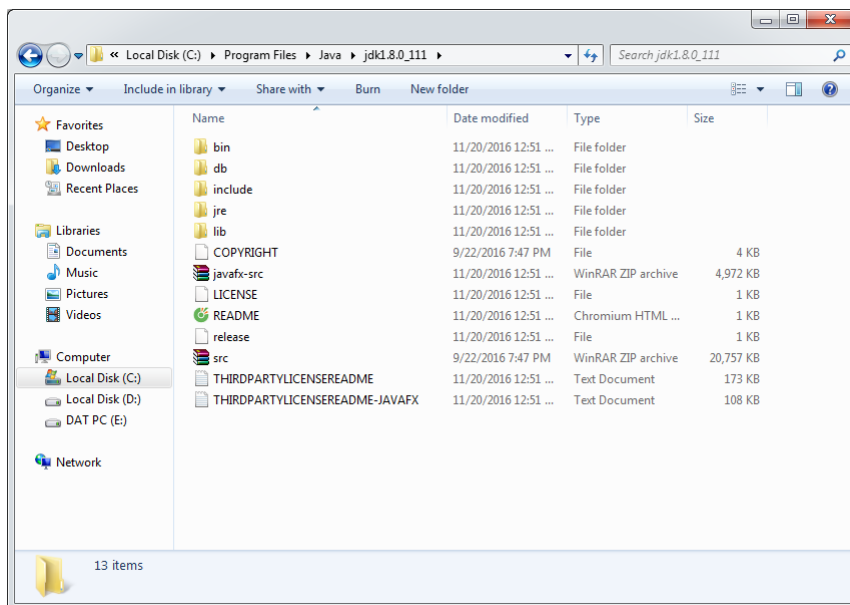


(d) Chọn **Close**

Hình 2.4: Cài đặt JDK trên hệ điều hành Windows

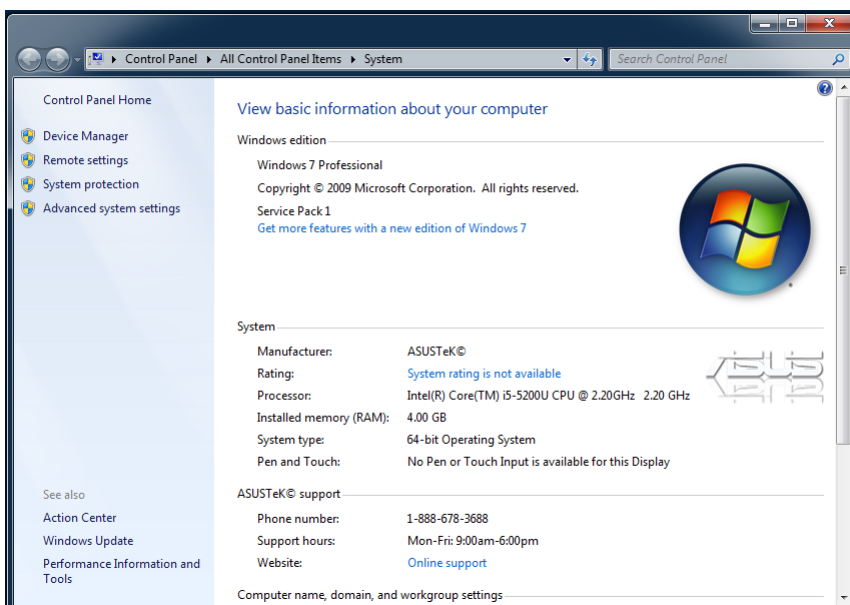
- **Bước 3:** Thiết lập giá trị cho biến `JAVA_HOME`.

- Lấy đường dẫn cài đặt JDK: ví dụ đường dẫn là `C:\Program Files\Java\jdk1.8.0_111` (lưu ý là đi đến thư mục `jdk1.8.0_111` như hình 2.5).



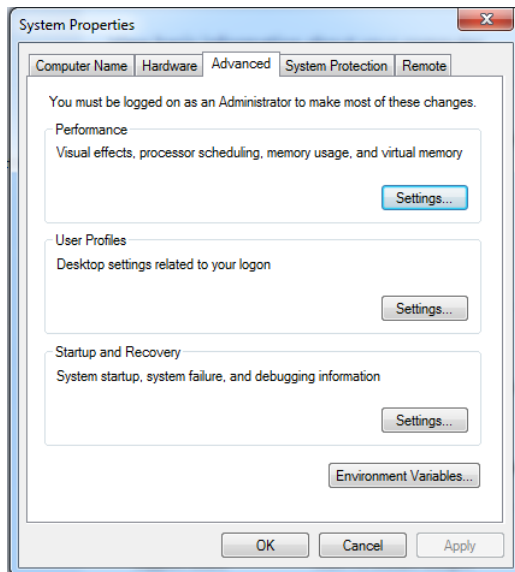
Hình 2.5: Vị trí của thư mục `jdk1.8.0_111`

- Click chuột phải vào biểu tượng Computer, chọn Properties, được giao diện như hình 2.6: chọn Advanced system settings bên menu trái.

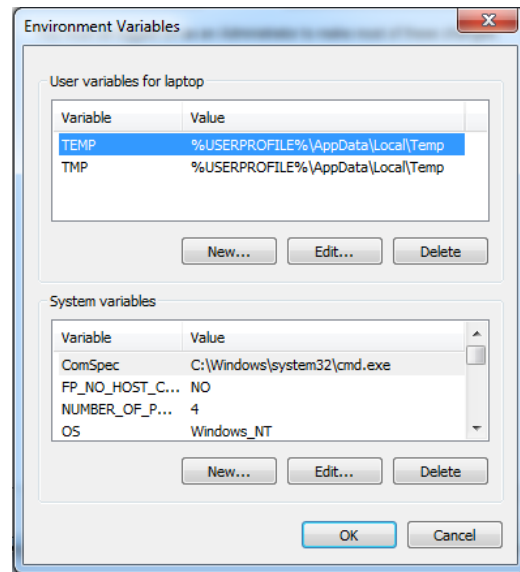


Hình 2.6: Mở cửa sổ System trên Windows

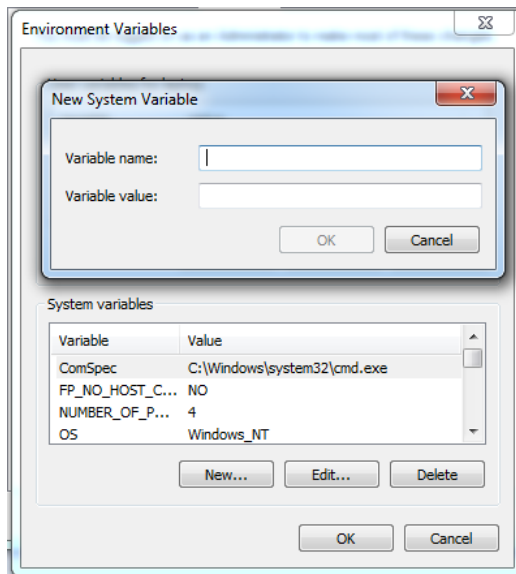
- Được cửa sổ như hình 2.7a: chọn tab Advanced, click chọn Environment Variables... được giao diện như hình 2.7b.



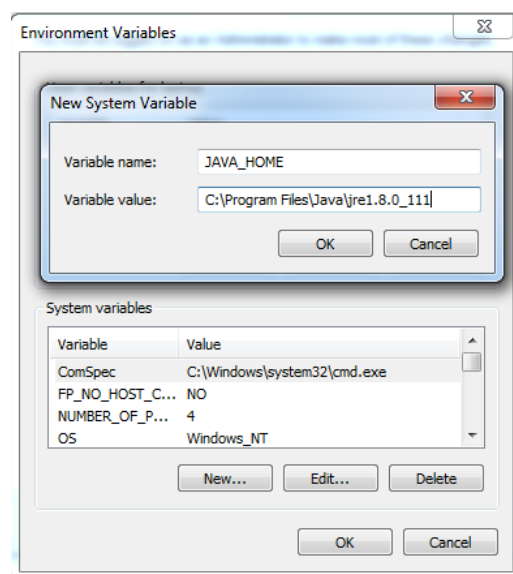
(a) Chọn tab Advanced



(b) Cửa sổ Environment Variables...



(c) Khởi tạo biến JAVA\_HOME

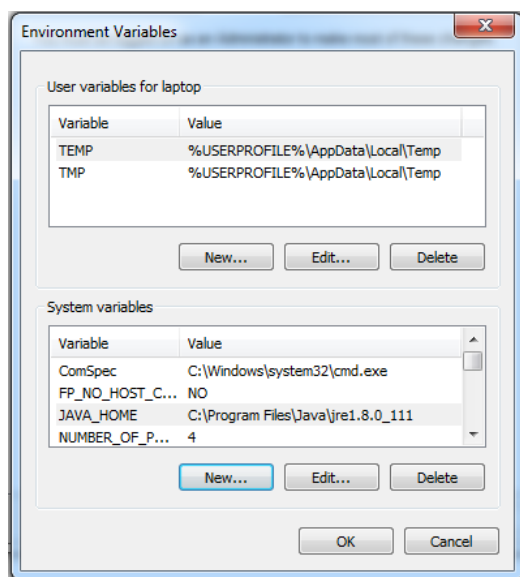


(d) Giá trị của biến JAVA\_HOME

Hình 2.7: Thiết lập đường dẫn cho biến JAVA\_HOME

- Tìm biến JAVA\_HOME trong khung System variables: nếu có biến JAVA\_HOME thì không cần thực hiện tiếp. Nếu chưa có biến JAVA\_HOME thì click chọn New để khởi tạo, được giao diện như hình 2.7c.
- Trên giao diện hình 2.7c:
  - + Ô Variable Name: nhập vào JAVA\_HOME.
  - + Ô Variable value: nhập vào đường dẫn của thư mục jdk1.8.0\_111 (xác định ở trên), ví dụ: C:\Program Files\Java\jdk1.8.0\_111
  - + Kết quả thiết lập trên hình 2.7d.

- Thực hiện xong chọn OK: lúc này biến JAVA\_HOME đã có trong khung System variables (hình 2.8). Chọn OK để đóng cửa sổ Environment Variables..., chọn OK để đóng cửa sổ System Properties.

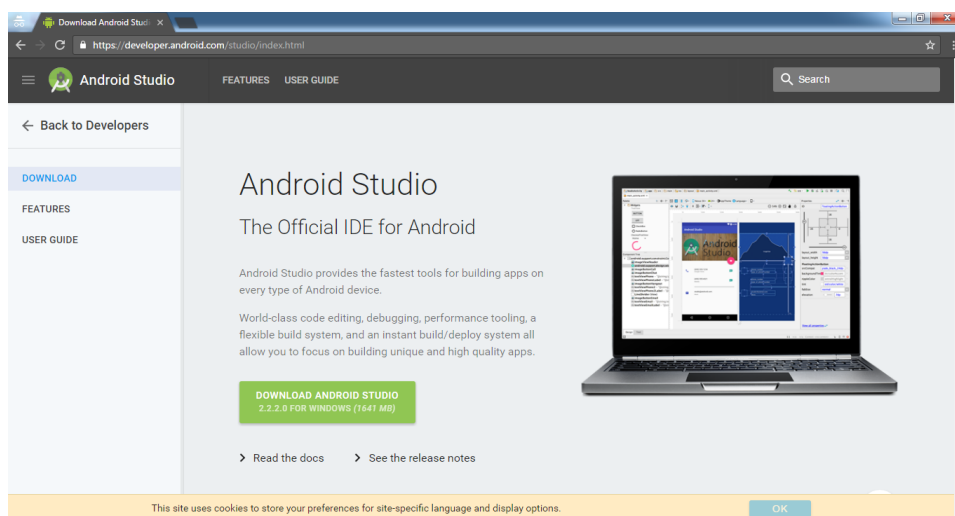


Hình 2.8: Thêm biến JAVA\_HOME vào cửa sổ System variables

## 2.4 Cài đặt Android Studio

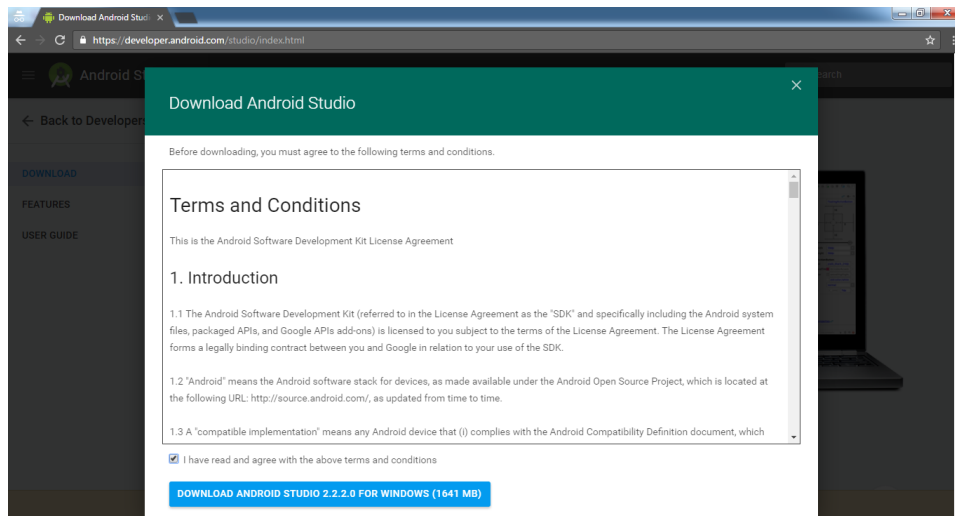
Thực hiện quá trình cài đặt theo các bước bên dưới:

- *Bước 1:* Tải phần mềm Android Studio về máy.
  - Truy cập vào địa chỉ: <https://developer.android.com/studio/index.html>, được giao diện như hình 2.9. Chọn DOWNLOAD ANDROID STUDIO.



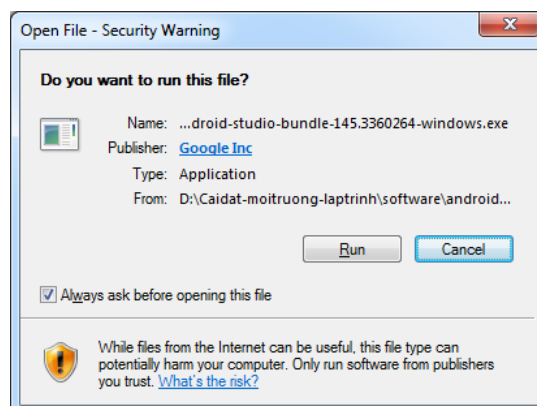
Hình 2.9: Download Android Studio từ trang chủ

- Khi xuất hiện thông báo Download Android Studio (trên hình 2.10), check I have read and agree with the above terms and conditions để xác nhận đồng ý các điều khoản sử dụng phần mềm.



Hình 2.10: Xác nhận đồng ý các điều khoản khi Download Android Studio

- Click chọn DOWNLOAD ANDROID STUDIO 2.2.2.0 FOR WINDOWS (1611MB) , phiên bản hiện tại của Android Studio là 2.2.2.0 và là phiên bản dành cho hệ điều hành Windows.
- Khi sử dụng Android Studio trên các hệ điều hành khác Windows (ví dụ các bản phân phối Linux,...) thì cách download cũng tương tự (trang web sẽ tự nhận dạng hệ điều hành để đưa phiên bản download hợp lý).
- **Bước 2:** Thực hiện cài đặt phần mềm Android Studio.
  - Vào ổ đĩa C (ổ đĩa hệ thống) tạo thư mục Android gồm 2 thư mục con là Android Studio và sdk.
  - Mở phần mềm Android Studio vừa tải về ở *bước 1* (ví dụ tên file là android-studio-bundle-145.3360264-windows.exe) được giao diện như hình 2.11. Chọn Run để bắt đầu cài đặt Android Studio.



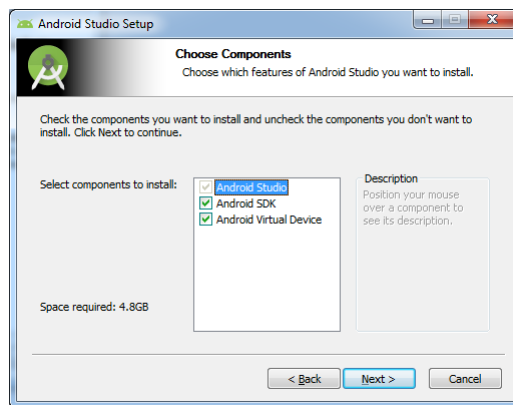
Hình 2.11: Mở file android-studio-bundle-145.3360264-windows.exe để cài đặt



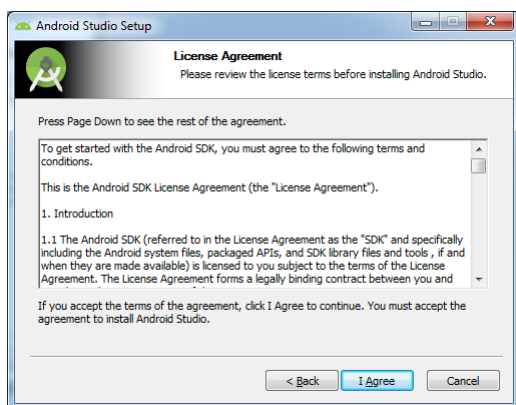
- Giao diện mới xuất hiện như hình 2.12a. Chọn Next.
- Được giao diện như hình 2.12b: để như mặc định, chọn Next.
- Được giao diện như hình 2.12c: để như mặc định, chọn I Agree.



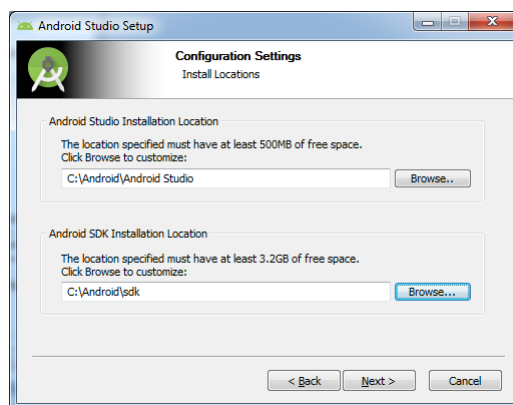
(a) Chọn Next



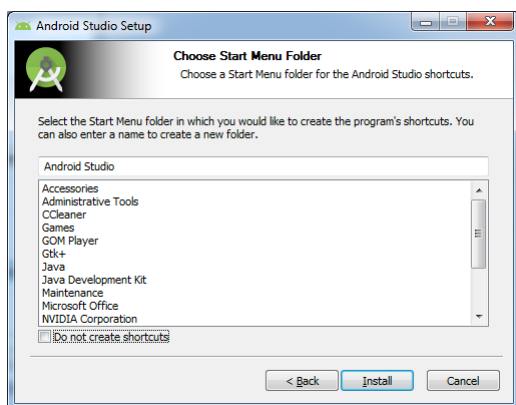
(b) Chọn Next



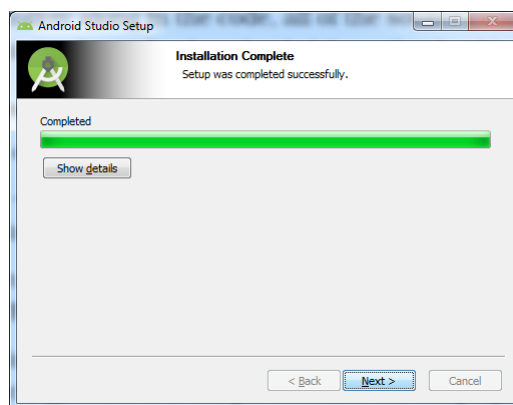
(c) Chọn I Agree



(d) Chọn nơi cài đặt Android Studio và SDK



(e) Chọn Install

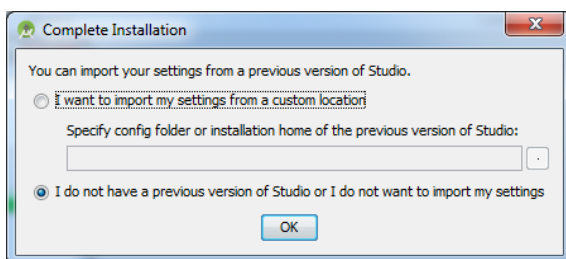


(f) Chọn Next

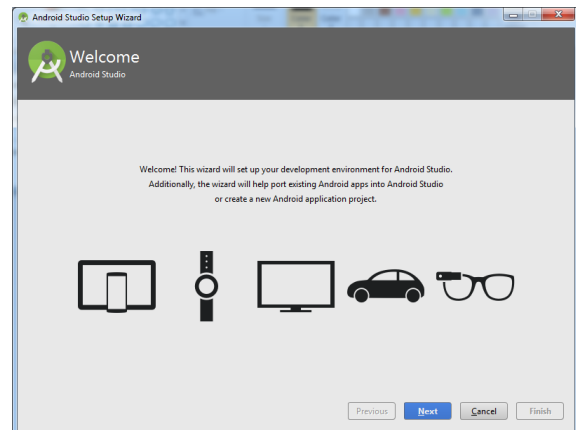
Hình 2.12: Cài đặt Android Studio trên hệ điều hành Windows

- Được giao diện như hình 2.12d: chọn nơi cài đặt vào các thư mục đã được tạo trước đó Android Studio và sdk.
- + Trong khung Android Studio Installation Location: dẫn đến thư mục C : \Android\Android Studio

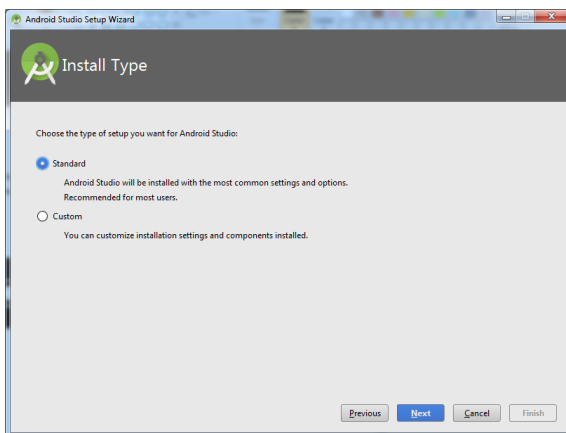
- + Trong khung Android SDK Installation Location: dẫn đến thư mục C:\Android\sdk
- + Chọn Next để sang bước tiếp theo.
- Được giao diện như hình 2.12e: chọn Install để tiến hành cài đặt Android Studio và Android SDK.
- Đợi quá trình cài đặt hoàn thành được giao diện như hình 2.12f: Chọn Next.
- **Bước 3: Cài đặt Android Studio Wizard.**



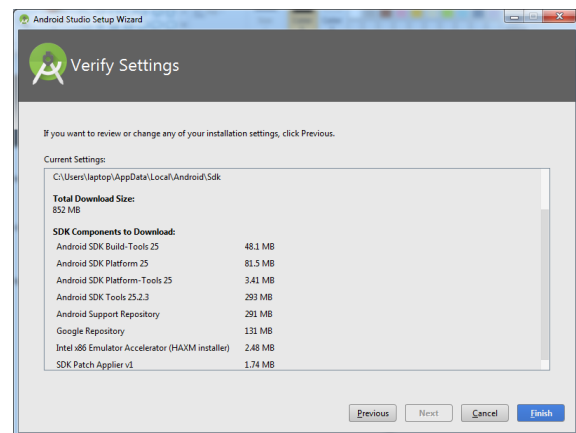
(a) Chọn OK



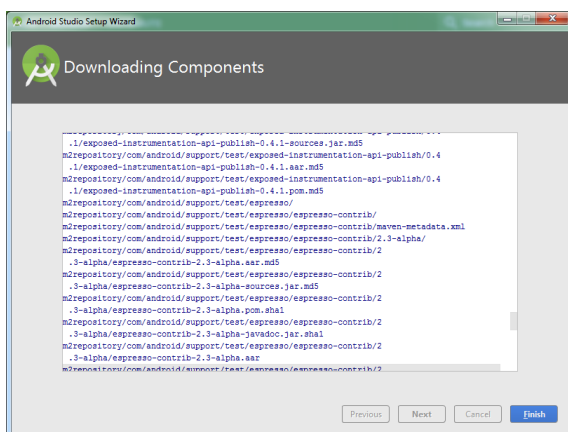
(b) Chọn Next



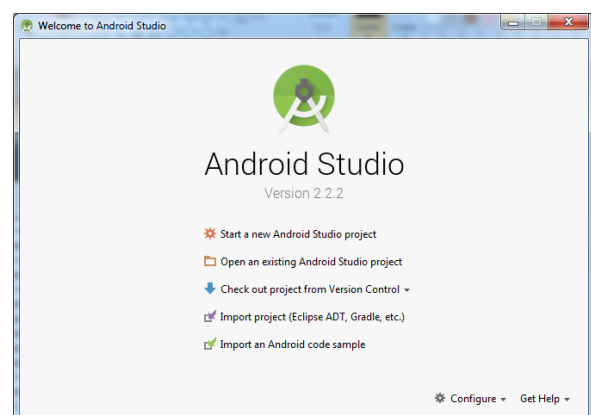
(c) Chọn Next



(d) Chọn Finish



(e) Chọn Finish



(f) Mở Android Studio lần đầu tiên

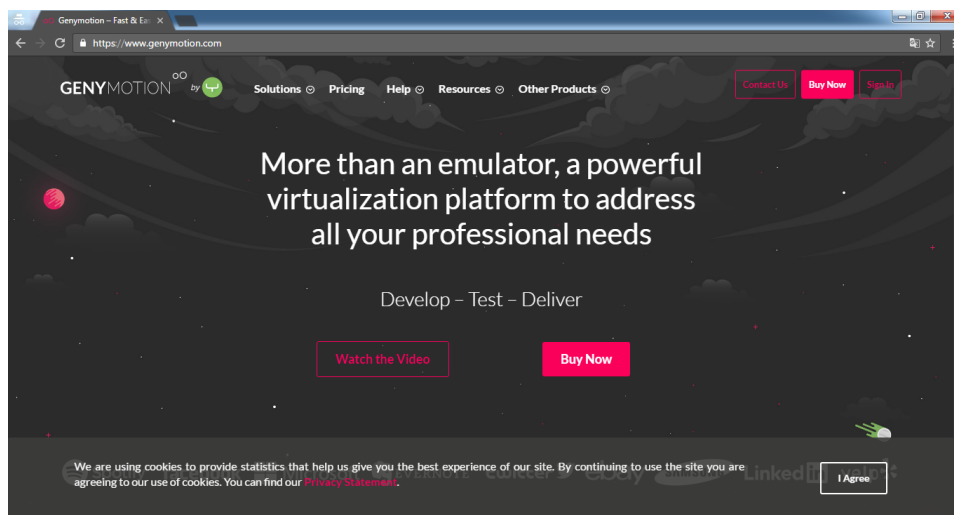
Hình 2.13: Cài đặt Android Studio Wizard

- Sau khi cài đặt Android Studio và Android SDK xong, xuất hiện thông báo như hình 2.13a. Nếu chưa cài đặt phiên bản Android Studio nào trước đó thì chọn I do not have a previous version of Studio or I do not want to import my settings rồi chọn OK.
- Được giao diện như hình 2.13b: chọn Next.
- Được giao diện như hình 2.13c: để mặc định – Standard, chọn Next.
- Được giao diện như hình 2.13d: chọn Finish.
- Được giao diện như hình 2.13e: chọn Finish.
- Khi quá trình cài đặt hoàn thành, mở Android Studio lên có giao diện như hình 2.13f.

## 2.5 Cài đặt Genymotion

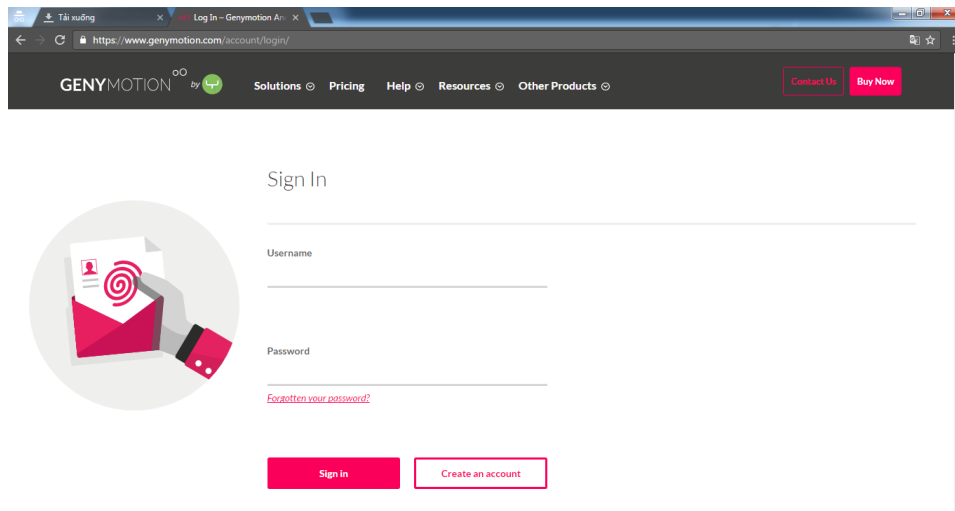
Thực hiện quá trình cài đặt theo các bước bên dưới:

- *Bước 1:* Tải phần mềm Genymotion về máy.
  - Truy cập vào địa chỉ <https://www.genymotion.com>, được giao diện như hình 2.14, để download phần mềm cần thực hiện đăng nhập (nếu chưa có tài khoản thì tạo mới).

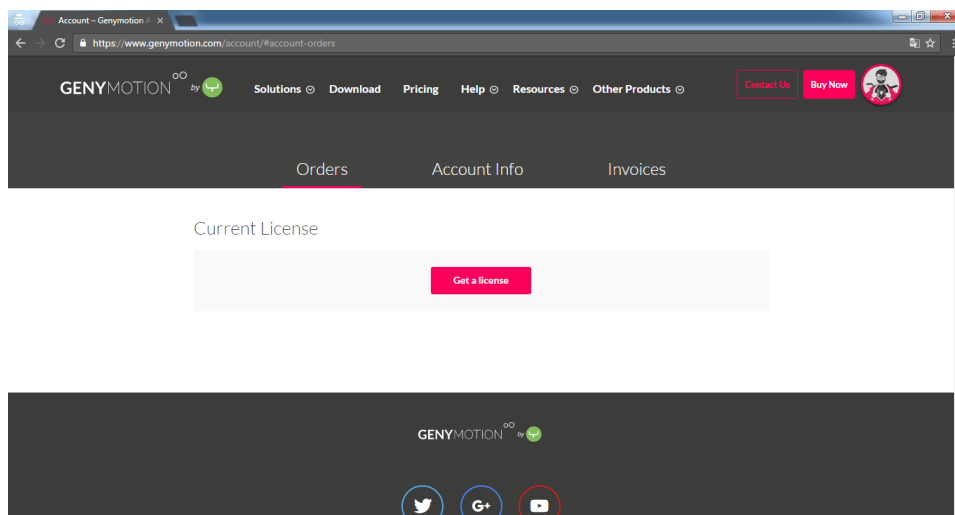


Hình 2.14: Trang chủ của Genymotion

- Click vào nút Sign In được giao diện như hình 2.15a, điền thông tin đăng nhập hoặc đăng ký tài khoản mới.
- Sau khi đăng nhập thành công được giao diện như hình 2.15b: click chọn Get a license.
- Chọn thẻ Download, được giao diện như hình 2.16: chọn phiên bản with VirtualBox để tải về máy. Ví dụ như trên hình 2.16 tải gói có dung lượng 154MB.

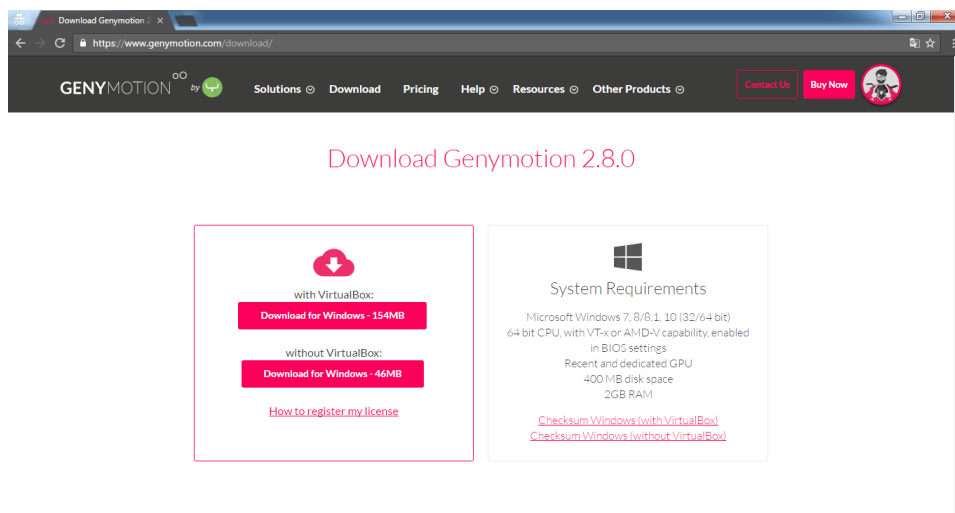


(a) Điền thông tin đăng nhập hoặc đăng ký tài khoản



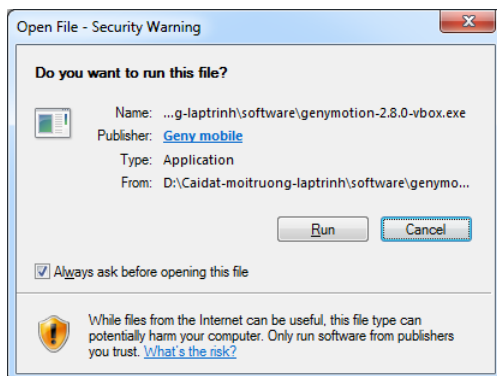
(b) Chọn Get a license

Hình 2.15: Đăng nhập tài khoản trên Genymotion

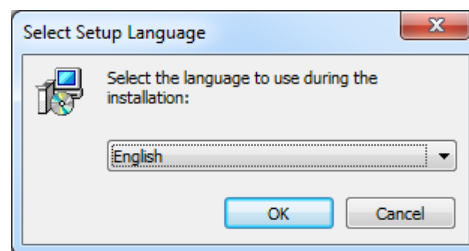


Hình 2.16: Download Genymotion về máy

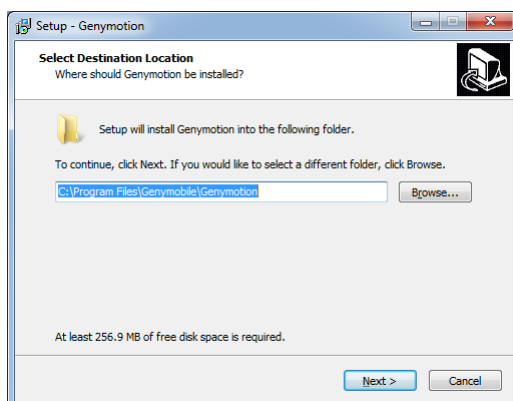
- **Bước 2:** Cài đặt phần mềm Genymotion và Oracle VM VirtualBox.
  - Mở phần mềm Genymotion vừa tải về, được giao diện như hình 2.17a: chọn Run để bắt đầu cài đặt.
  - Được giao diện như hình 2.17b: chọn ngôn ngữ là English và chọn OK.
  - Trên giao diện hình 2.17c: nơi cài đặt chọn như mặc định, chọn Next.
  - Được giao diện hình 2.17d: tiếp tục chọn Next.



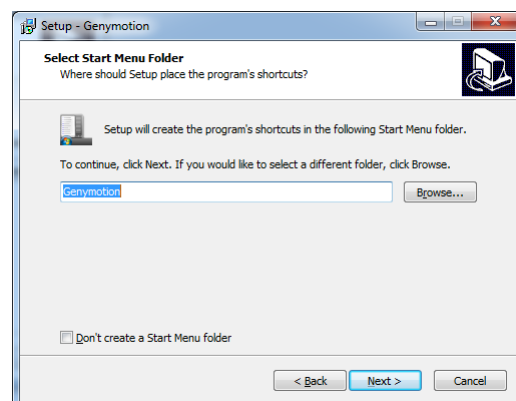
(a) Chọn Run



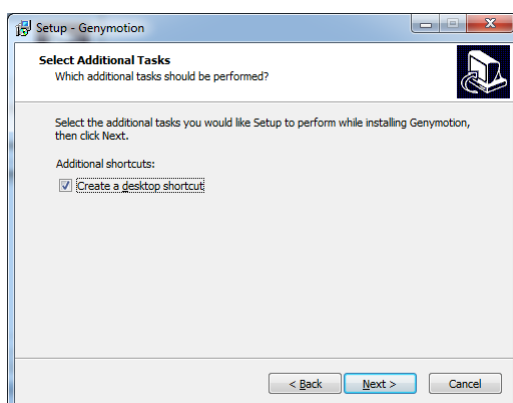
(b) Chọn ngôn ngữ English và chọn OK



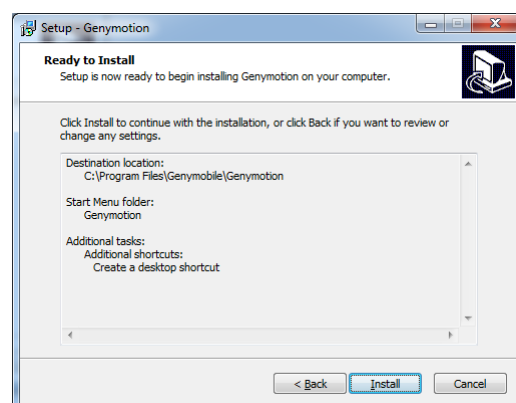
(c) Chọn Next



(d) Chọn Next



(e) Chọn Next



(f) Chọn Install

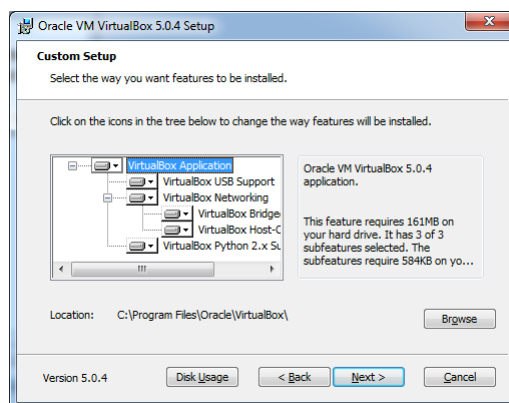
Hình 2.17: Cài đặt Genymotion trên Windows

- Đến phần cài đặt Oracle VM VirtualBox (được tích hợp sẵn trong gói Genymotion), được giao diện như hình 2.18a: chọn Next.
- Được giao diện như hình 2.18b: để nơi cài đặt như mặc định, chọn Next.

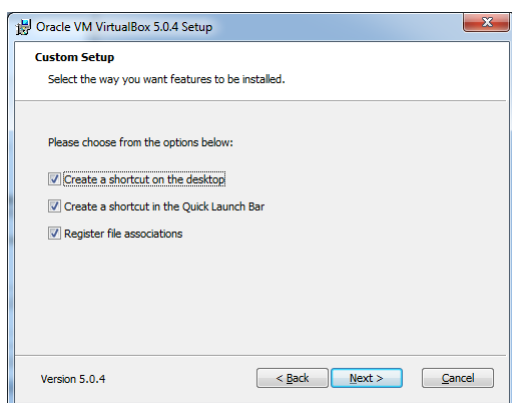
- Được giao diện như hình 2.18c: chọn Next. Được giao diện như hình 2.18d: chọn Yes.
- Được giao diện như hình 2.18e: chọn Install để cài đặt.
- Trong quá trình cài đặt, xuất hiện thông báo như hình 2.18f: chọn Always trust software from "Oracle Corporation" và chọn Install để cài đặt.



(a) Chọn Next



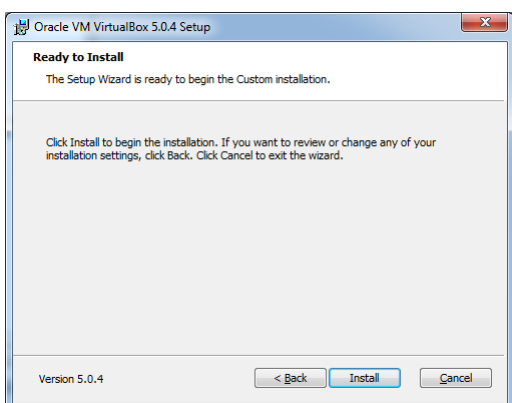
(b) Chọn Next



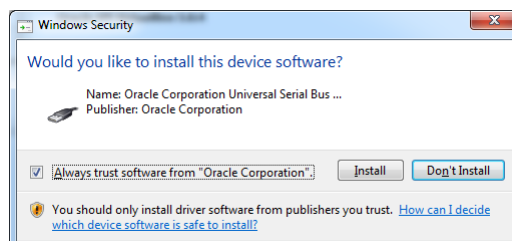
(c) Chọn Next



(d) Chọn Yes



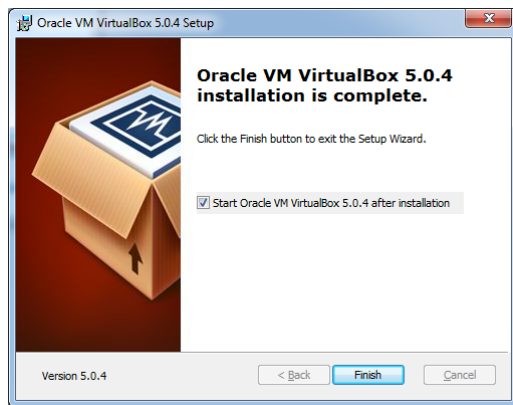
(e) Chọn Install



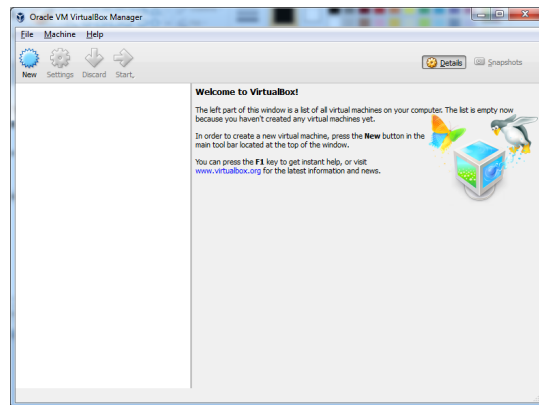
(f) Check chọn Always trust. . . và chọn Install

Hình 2.18: Cài đặt Oracle VM VirtualBox trên Windows

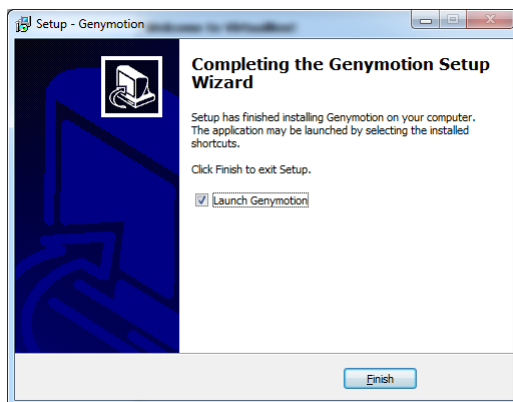
- Quá trình cài đặt Genymotion và Oracle VM VirtualBox hoàn thành được giao diện như hình 2.19: chọn Finish để mở các phần mềm lần đầu tiên để thiết lập.



(a) Chọn Finish



(b) Giao diện của Oracle VM VirtualBox Manager



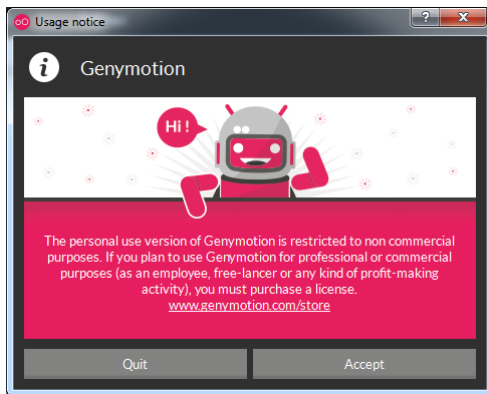
(c) Chọn Finish



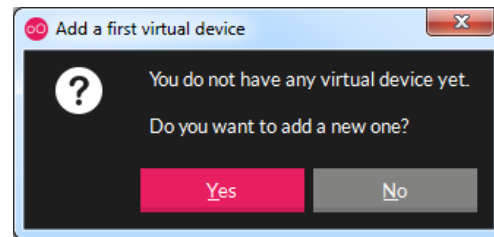
(d) Mở Genymotion

Hình 2.19: Hoàn thành cài đặt Genymotion và Oracle VM VirtualBox trên Windows

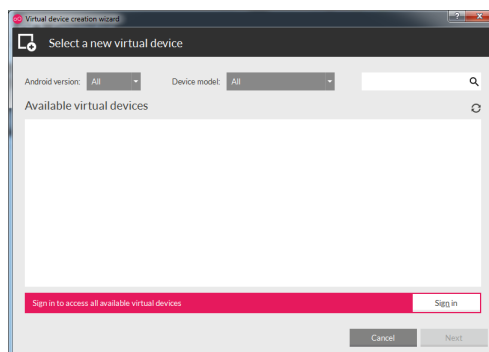
- **Bước 3:** Tải thiết bị ảo trên Genymotion về máy để thực hiện mô phỏng.
  - Khi mở Genymotion lần đầu tiên, xuất hiện thông báo như hình 2.20a: chọn Accept.
  - Do mới cài đặt nên chưa có thiết bị máy ảo nào được tải về, được giao diện như hình 2.20b: chọn Yes để tải thiết bị mới về máy.
  - Được giao diện như hình 2.20c: chọn Sign in để đăng nhập xem tất cả các máy ảo.
  - Xuất hiện thông báo như hình 2.20d: nhập vào Username và Password đã đăng ký lúc tải Genymotion từ trang chủ về máy.
  - Sau khi đăng nhập thành công, chúng ta thấy danh sách các thiết bị máy ảo như hình 2.20e. Chọn bất kỳ một máy ảo nào đó, ví dụ chọn máy ảo Google Nexus 4 - 4.3 - API - 768x1280 (hình 2.20f), có thông tin máy cho trên hình 2.20g, chọn Next để tải máy ảo về máy. Đợi quá trình tải hoàn thành, được giao diện như hình 2.20h: chọn Finish để hoàn tất.
  - Khi tải thiết bị máy ảo thành công, danh sách Your Virtual Devices có tên thiết bị vừa tải về như hình 2.21a: chọn máy ảo và chọn Start để mở máy ảo lên kiểm tra (như hình 2.21b), nếu khởi động máy ảo thành công thì được kết quả như hình 2.21c và 2.21d (tùy thuộc vào cấu hình máy mà quá trình khởi động nhanh hay chậm).



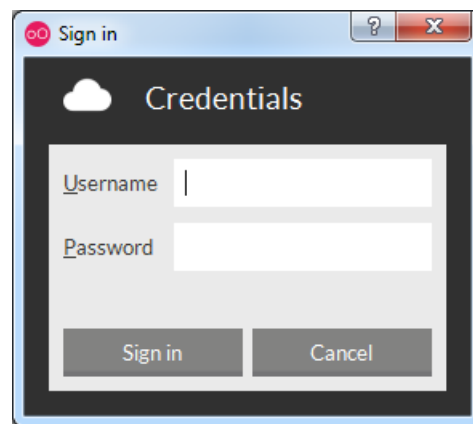
(a) Chọn Accept



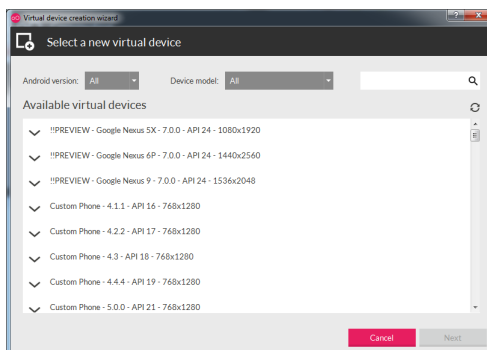
(b) Chọn Yes



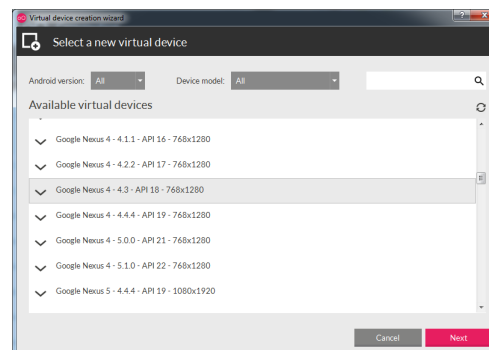
(c) Chọn Sign in



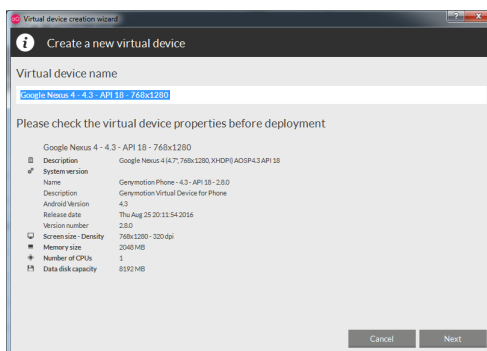
(d) Nhập thông tin đăng nhập



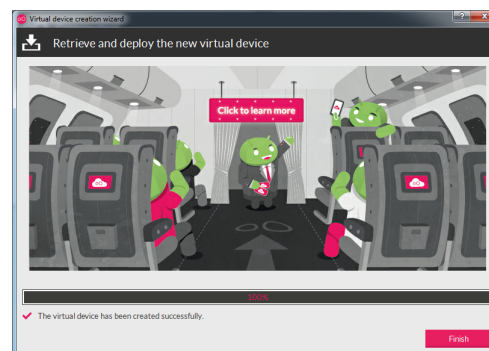
(e) Danh sách các máy ảo



(f) Chọn bất kỳ một máy ảo nào và chọn Next



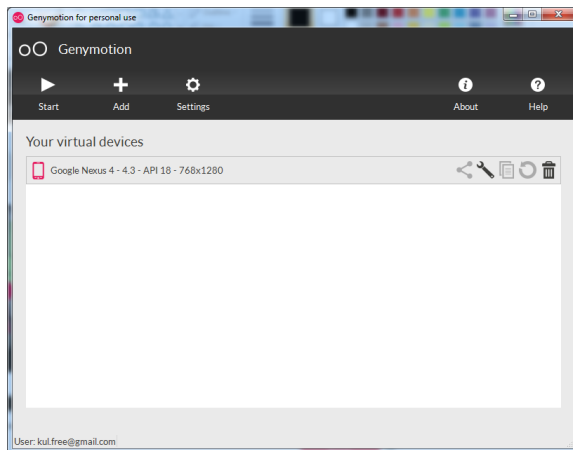
(g) Xem thông tin máy ảo và chọn Next



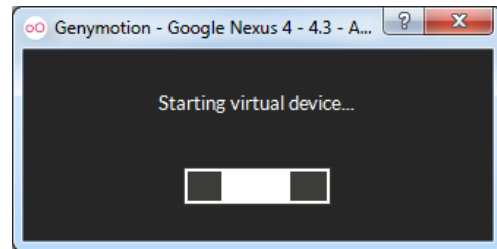
(h) Quá trình tải máy ảo về máy

Hình 2.20: Tải máy ảo Genymotion về máy

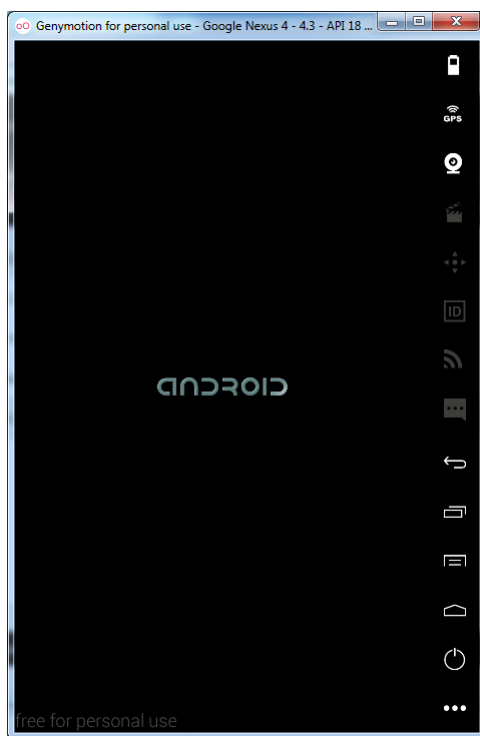




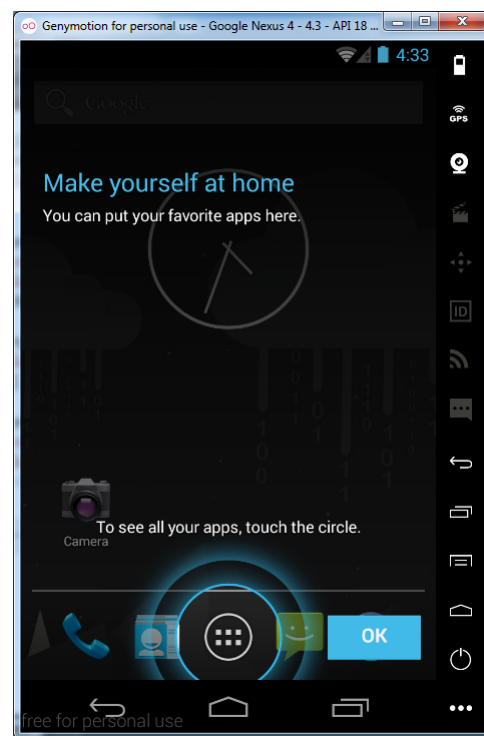
(a) Chọn máy ảo và chọn Start



(b) Quá trình kiểm tra mở máy ảo



(c) Quá trình khởi động máy ảo



(d) Khởi động máy ảo thành công

Hình 2.21: Mở máy ảo Genymotion để kiểm tra

- Sau này muốn mở máy ảo lên sử dụng thì chọn shortcut có tên là Genymotion (không mở shortcut có tên Genymotion Shell do chưa cần dùng đến), sẽ có giao diện như hình 2.21a:
  - Muốn khởi động thiết bị thì chọn thiết bị và nhấn vào nút Start.
  - Muốn thêm thiết bị mới thì nhấn vào nút Add và chọn thiết bị mới.

## Chương 3

# Tìm hiểu OpenGL ES 2.0 trên Android và Viết ứng dụng

### 3.1 Kiểu dữ liệu trong OpenGL ES

Với một số lệnh tham số truyền vào là các kiểu dữ liệu được cho trong bảng 3.1.

Hậu tố	Kiểu dữ liệu	Ứng với kiểu trong C	Ứng với kiểu trong OpenGL
b	8-bit signed integer	signed char	GLbyte
ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean
s	16-bit signed integer	short	GLshort
us	16-bit unsigned integer	unsigned short	GLushort
i	32-bit signed integer	int	GLint
ui	32-bit unsigned integer	unsigned int	GLuint, GLbitfield, GLenum
x	16.16 fixed point	int	GLfixed
f	32-bit floating point	float	GLfloat, GLclampf

Bảng 3.1: Kiểu dữ liệu trong OpenGL ES

Phần mô tả các lệnh của OpenGL ES 2.0 cùng với kiểu dữ liệu của tham số được cho trong tài liệu có địa chỉ <https://www.khronos.org/opengles/sdk/docs/man/>

### 3.2 Sử dụng OpenGL ES 2.0 trong Android

Có hai lớp chính để tạo và thực thi đồ họa với OpenGL ES API: `GLSurfaceView` và `GLSurfaceView.Renderer`

- `GLSurfaceView`: là View mà chúng ta vẽ, thao tác các đối tượng trên đó.

- `GLSurfaceView.Renderer`: là một Interface định nghĩa các Methods được yêu cầu để vẽ đồ họa trên `GLSurfaceView`. Chúng ta cần cài tiến các Methods sau:
  - `onSurfaceCreated()`: hệ thống gọi phương thức này một lần khi tạo `GLSurfaceView`. Dùng phương thức này để thực hiện những hành động chỉ cần tạo ra một lần như cài đặt tham số môi trường OpenGL hoặc khởi tạo các đối tượng đồ họa OpenGL.
  - `onDrawFrame()`: hệ thống gọi phương thức này khi vẽ lại `GLSurfaceView`.
  - `onSurfaceChanged()`: hệ thống gọi phương thức này khi `GLSurfaceView` thay đổi kích thước.

Để sử dụng OpenGL ES 2.0 yêu cầu Android 2.2 (API Level 8) hoặc cao hơn.

### 3.2.1 Khai báo sử dụng OpenGL ES 2.0 trong file Manifest

Thêm vào file `AndroidManifest.xml` khai báo sau:

---

```
1 <uses-feature android:glEsVersion="0x00020000"
  android:required="true" />
```

---

- Nếu xuất hiện lỗi `Default Activity not found`, khắc phục bằng cách thêm đoạn code bên dưới vào file `AndroidManifest.xml` (nằm trong thẻ `application`):

---

```
1 <activity
2   android:name="name_package.MainActivity"
3   android:label="@string/app_name">
4   <intent-filter>
5     <action android:name="android.intent.action.MAIN" />
6
7     <category android:name="android.intent.category.LAUNCHER"
8       />
9   </intent-filter>
10 </activity>
```

---

với `name_package` được thay bằng tên của package trong manifest. Ví dụ: cần khai báo như sau:

---

```
1 <activity
2   android:name="com.opengles.android.opengles20.MainActivity"
3   android:label="@string/app_name">
4   <intent-filter>
5     <action android:name="android.intent.action.MAIN" />
6
7     <category android:name="android.intent.category.LAUNCHER"
8       />
9   </intent-filter>
10 </activity>
```

---

### 3.2.2 Khởi tạo file Activity khi sử dụng OpenGL ES 2.0

- Nội dung của file Activity như bên dưới (tên file MainActivity.java):

---

```
1 package com.opengles.android.opengles20;
2
3 import android.app.Activity;
4 import android.opengl.GLSurfaceView;
5 import android.os.Bundle;
6
7 public class MainActivity extends Activity {
8     private GLSurfaceView mGLView;
9
10    @Override
11    public void onCreate(Bundle savedInstanceState) {
12        super.onCreate(savedInstanceState);
13
14        mGLView = new MyGLSurfaceView(this);
15        setContentView(mGLView);
16    }
17 }
```

---

- Với đoạn code trên ta thấy cần tạo lớp GLSurfaceView để sử dụng trong lớp Activity.

### 3.2.3 Xây dựng lớp GLSurfaceView

- Xây dựng lớp GLSurfaceView như bên dưới (tên file MyGLSurfaceView.java):

---

```
1 package com.opengles.android.opengles20;
2
3 import android.content.Context;
4 import android.opengl.GLSurfaceView;
5
6 public class MyGLSurfaceView extends GLSurfaceView {
7     private final MyGLRenderer mRenderer;
8
9     public MyGLSurfaceView(Context context){
10        super(context);
11
12        // Khai bao su dung OpenGL ES 2.0
13        setEGLContextClientVersion(2);
14
15        mRenderer = new MyGLRenderer();
16
17        // Cai dat Renderer dung de ve GLSurfaceView
18        setRenderer(mRenderer);
19    }
20 }
```

---

### 3.2.4 Xây dựng lớp Renderer

- Xây dựng lớp Renderer như bên dưới (tên file MyGLRendererer.java).

---

```
1 package com.opengles.android.opengles20;
2
3 import android.opengl.GLES20;
4 import android.opengl.GLSurfaceView.Renderer;
5
6 import javax.microedition.khronos.egl.EGLConfig;
7 import javax.microedition.khronos.opengles.GL10;
8
9 public class MyGLRenderer implements GLSurfaceView.Renderer {
10
11     public void onSurfaceCreated(GL10 unused, EGLConfig
12         config) {
13         // Dat mau nen cho frame: tham so RGBA
14         GLES20.glClearColor(0.8f, 0.0f, 0.0f, 1.0f);
15     }
16
17     public void onDrawFrame(GL10 unused) {
18         // Xac dinh lai mau nen
19         GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT);
20     }
21
22     public void onSurfaceChanged(GL10 unused, int width, int
23         height) {
24         GLES20.glViewport(0, 0, width, height);
25     }
26 }
```

---

- Phương thức `onSurfaceCreated`, `onDrawFrame` và `onSurfaceChanged` được giải thích trước đó.
- Trong lớp `Renderer`, định nghĩa các phương thức để vẽ lên lớp `GLSurfaceView`.
- Còn vẽ các đối tượng cụ thể thì tạo ra các lớp riêng lớp riêng biệt cho từng đối tượng rồi khai báo vào lớp `Renderer`.
- Các thao tác sau này chủ yếu là trên lớp `Renderer` và tạo các lớp định nghĩa cho từng đối tượng cụ thể.

### 3.2.5 Nội dung của các file chương trình

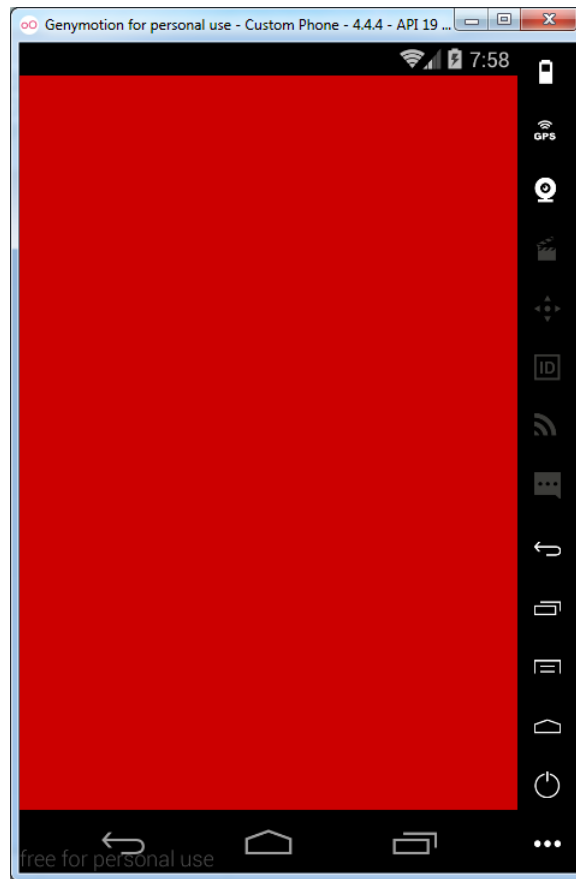
- Source code của project được lưu ở địa chỉ: <https://github.com/minhdatvnus/nienluan3/blob/master/CreateOpenGLES20.zip>
- Nội dung của file `AndroidManifest.xml`:

---

```
1 <manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
2   package="com.opengles.android.opengles20">
3
4   <uses-feature android:glEsVersion="0x00020000"
     android:required="true" />
5
6   <application
7     android:allowBackup="true"
8     android:icon="@mipmap/ic_launcher"
9     android:label="@string/app_name"
10    android:supportsRtl="true"
11    android:theme="@style/AppTheme">
12
13     <activity
14       android:name=
15       "com.opengles.android.opengles20.MainActivity"
16       android:label="@string/app_name">
17       <intent-filter>
18         <action
19           android:name="android.intent.action.MAIN" />
20
21         <category
22           android:name="android.intent.category.LAUNCHER"
23         />
24       </intent-filter>
25     </activity>
26   </application>
27 </manifest>
```

---

- Nội dung của file `MainActivity.java`: mục 3.2.2 (trang 21).
- Nội dung của file `MyGLSurfaceView.java`: mục 3.2.3 (trang 21).
- Nội dung của file `MyGLRenderer.java`: mục 3.2.4 (trang 22).
- Sau khi biên dịch được kết quả như hình 3.1.



Hình 3.1: Sử dụng OpenGL ES 2.0 trên Android

### 3.3 Viết ứng dụng vẽ một số đối tượng hình học

Để vẽ các đối tượng hình học cơ bản thực hiện theo 2 bước:

- *Bước 1*: Định nghĩa các shapes.
- *Bước 2*: Thực hiện vẽ các shapes đã định nghĩa trước đó.
- \* Các đối tượng này được định nghĩa trên một lớp riêng.

#### 3.3.1 Vẽ đối tượng là tam giác

##### Định nghĩa tam giác

- Mỗi đỉnh của tam giác được xác định bởi tọa độ:  $(x, y, z)$ .
- Đối tượng được định nghĩa như bên dưới (file `Triangle.java`):

---

```

1 public class Triangle {
2
3     private FloatBuffer vertexBuffer;
4
5     // number of coordinates per vertex in this array

```

```

6     static final int COORDS_PER_VERTEX = 3;
7     static float triangleCoords[] = { // in counterclockwise
        order:
8         0.0f, 0.5f, 0.0f, // top
9         -0.5f, -0.5f, 0.0f, // bottom left
10        0.5f, -0.5f, 0.0f // bottom right
11    };
12
13    // Set color with red, green, blue and alpha (opacity)
    values
14    float color[] = {0.0f, 0.6f, 1.0f, 1.0f};
15
16    public Triangle() {
17        // initialize vertex byte buffer for shape coordinates
18
19        // (number of coordinate values * 4 bytes per float)
20        ByteBuffer bb =
21            ByteBuffer.allocateDirect(triangleCoords.length*4);
22        // use the device hardware's native byte order
23        bb.order(ByteOrder.nativeOrder());
24
25        // create a floating point buffer from the ByteBuffer
26        vertexBuffer = bb.asFloatBuffer();
27        // add the coordinates to the FloatBuffer
28        vertexBuffer.put(triangleCoords);
29        // set the buffer to read the first coordinate
30        vertexBuffer.position(0);
31    }

```

---

## Vẽ đối tượng là tam giác

- Trong lớp `Renderer`, khai báo và định nghĩa thêm các phần sau (bước khởi tạo các shapes trong phương thức `onSurfaceCreated`).

```

1 public class MyGLRenderer implements GLSurfaceView.Renderer {
2
3     ...
4     private Triangle mTriangle;
5
6     public void onSurfaceCreated(GL10 unused, EGLConfig
        config) {
7         ...
8
9         // initialize a triangle
10        mTriangle = new Triangle();
11    }
12    ...
13 }

```

---



- Thực hiện vẽ các shapes đã được định nghĩa trước đó (trong file `Triangle.java`):

– Khai báo các dòng sau trong lớp `Triangle`:

---

```
1 public class Triangle {
2
3     private final String vertexShaderCode =
4         "attribute vec4 vPosition;" +
5         "void main() {" +
6         "    gl_Position = vPosition;" +
7         "}";
8
9     private final String fragmentShaderCode =
10        "precision mediump float;" +
11        "uniform vec4 vColor;" +
12        "void main() {" +
13        "    gl_FragColor = vColor;" +
14        "}";
15    ...
16 }
```

---

– Tạo phương thức `loadShader` trong lớp `Renderer`:

---

```
1 public static int loadShader(int type, String
   shaderCode) {
2
3     // create a vertex shader type
4     (GL_ES20.GL_VERTEX_SHADER)
5     // or a fragment shader type
6     (GL_ES20.GL_FRAGMENT_SHADER)
7     int shader = GL_ES20.glCreateShader(type);
8
9     // add the source code to the shader and compile it
10    GL_ES20.glShaderSource(shader, shaderCode);
11    GL_ES20.glCompileShader(shader);
12
13    return shader;
14 }
```

---

– Khai báo thêm các dòng sau trong lớp `Triangle`:

---

```
1 public class Triangle() {
2     ...
3
4     private final int mProgram;
5
6     public Triangle() {
7         ...
8
9         int vertexShader =
10            MyGLRenderer.loadShader(GL_ES20.GL_VERTEX_SHADER,
                                   vertexShaderCode);
```

```

11
12         int fragmentShader =
13             MyGLRenderer.loadShader(GLES20.GL_FRAGMENT_SHADER,
14                                     fragmentShaderCode);
15
16         // create empty OpenGL ES Program
17         mProgram = GLES20.glCreateProgram();
18
19         // add the vertex shader to program
20         GLES20.glAttachShader(mProgram, vertexShader);
21
22         // add the fragment shader to program
23         GLES20.glAttachShader(mProgram, fragmentShader);
24
25         // creates OpenGL ES program executables
26         GLES20.glLinkProgram(mProgram);
27     }

```

---

- Tạo phương thức draw trong lớp Triangle để vẽ các shapes đã định nghĩa:

```

1 private int mPositionHandle;
2 private int mColorHandle;
3
4 private final int vertexCount = triangleCoords.length /
5     COORDS_PER_VERTEX;
6 private final int vertexStride = COORDS_PER_VERTEX * 4;
7     // 4 bytes per vertex
8
9 public void draw() {
10     // Add program to OpenGL ES environment
11     GLES20.glUseProgram(mProgram);
12
13     // get handle to vertex shader's vPosition member
14     mPositionHandle =
15         GLES20.glGetAttribLocation(mProgram, "vPosition");
16
17     // Enable a handle to the triangle vertices
18     GLES20.glEnableVertexAttribArray(mPositionHandle);
19
20     // Prepare the triangle coordinate data
21     GLES20.glVertexAttribPointer(mPositionHandle,
22                                 COORDS_PER_VERTEX,
23                                 GLES20.GL_FLOAT, false,
24                                 vertexStride, vertexBuffer);
25
26     // get handle to fragment shader's vColor member
27     mColorHandle = GLES20.glGetUniformLocation(mProgram,
28         "vColor");
29
30 }

```

```

26
27 // Set color for drawing the triangle
28 GLES20.glUniform4fv(mColorHandle, 1, color, 0);
29
30 // Draw the triangle
31 GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0,
    vertexCount);
32
33 // Disable vertex array
34 GLES20.glDisableVertexAttribArray(mPositionHandle);
35 }

```

---

- Nội dung của các file chương trình:

- Source code của project được lưu ở địa chỉ:

<https://github.com/minhdatvnus/nienluan3/blob/master/TriangleOpenGL20.zip>

- Nội dung của các file `AndroidManifest.xml`, `MainActivity.java` và `MyGLSurfaceView.java` không thay đổi.
- Nội dung của file `MyGLRenderrer.java`:

---

```

1 package com.opengles.android.opengles20;
2
3 import android.opengl.GLES20;
4 import android.opengl.GLSurfaceView.Renderer;
5
6 import javax.microedition.khronos.egl.EGLConfig;
7 import javax.microedition.khronos.opengles.GL10;
8
9 public class MyGLRenderrer implements Renderer {
10
11     private Triangle mTriangle;
12
13     public void onSurfaceCreated(GL10 unused, EGLConfig
        config) {
14         // Cài đặt màu sắc của frame
15         GLES20.glClearColor(0.8f, 0.0f, 0.0f, 1.0f);
16         mTriangle = new Triangle();
17     }
18
19
20     public void onDrawFrame(GL10 unused) {
21         // Đặt lại màu sắc của frame
22         GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT);
23         mTriangle.draw();
24     }
25
26
27     public void onSurfaceChanged(GL10 unused, int width,
        int height) {

```

```

28         GLES20.glViewport(0, 0, width, height);
29     }
30
31
32     public static int loadShader(int type, String
        shaderCode) {
33
34         // Create a vertex shader type
35         (GLES20.GL_VERTEX_SHADER)
36         // or a fragment shader type
37         (GLES20.GL_FRAGMENT_SHADER)
38         int shader = GLES20.glCreateShader(type);
39
40         // add the source code to the shader and compile it
41         GLES20.glShaderSource(shader, shaderCode);
42         GLES20.glCompileShader(shader);
43
44         return shader;
45     }
46 }

```

---

– Nội dung của file Triangle.java:

---

```

1 package com.opengles.android.opengles20;
2
3 import android.opengl.GLES20;
4
5 import java.nio.ByteBuffer;
6 import java.nio.ByteOrder;
7 import java.nio.FloatBuffer;
8
9 public class Triangle {
10
11     private final String vertexShaderCode =
12         "attribute vec4 vPosition;" +
13         "void main() {" +
14         "    gl_Position = vPosition;" +
15         "}";
16
17     private final String fragmentShaderCode =
18         "precision mediump float;" +
19         "uniform vec4 vColor;" +
20         "void main() {" +
21         "    gl_FragColor = vColor;" +
22         "}";
23
24     private FloatBuffer vertexBuffer;
25     private final int mProgram;
26
27     private int mPositionHandle;
28     private int mColorHandle;

```

```

29
30     private final int vertexCount = triangleCoords.length
        / COORDS_PER_VERTEX;
31
32     private final int vertexStride = COORDS_PER_VERTEX *
        4; // 4 bytes per vertex
33
34     // number of coordinates per vertex in this array
35     static final int COORDS_PER_VERTEX = 3;
36
37     static float triangleCoords[] = {
38         // in counterclockwise order:
39         0.0f, 0.5f, 0.0f, // top
40         -0.5f, -0.5f, 0.0f, // bottom left
41         0.5f, -0.5f, 0.0f // bottom right
42     };
43
44     // Set color with red, green, blue and alpha
        (opacity) values
45     float color[] = {0.0f, 0.6f, 1.0f, 1.0f };
46
47     public Triangle() {
48
49         // initialize vertex byte buffer for shape
            coordinates
50
51         // (number of coordinate values * 4 bytes per
            float)
52         ByteBuffer bb =
            ByteBuffer.allocateDirect(triangleCoords.length
                * 4);
53         // use the device hardware's native byte order
54         bb.order(ByteOrder.nativeOrder());
55
56         // create a floating point buffer from the
            ByteBuffer
57         vertexBuffer = bb.asFloatBuffer();
58         // add the coordinates to the FloatBuffer
59         vertexBuffer.put(triangleCoords);
60         // set the buffer to read the first coordinate
61         vertexBuffer.position(0);
62
63         int vertexShader =
            MyGLRenderer.loadShader(GLES20.GL_VERTEX_SHADER,
                vertexShaderCode);
64
65         int fragmentShader =
            MyGLRenderer.loadShader(GLES20.GL_FRAGMENT_SHADER,
                fragmentShaderCode);
66
67
68         // create empty OpenGL ES Program
69         mProgram = GLES20.glCreateProgram();

```

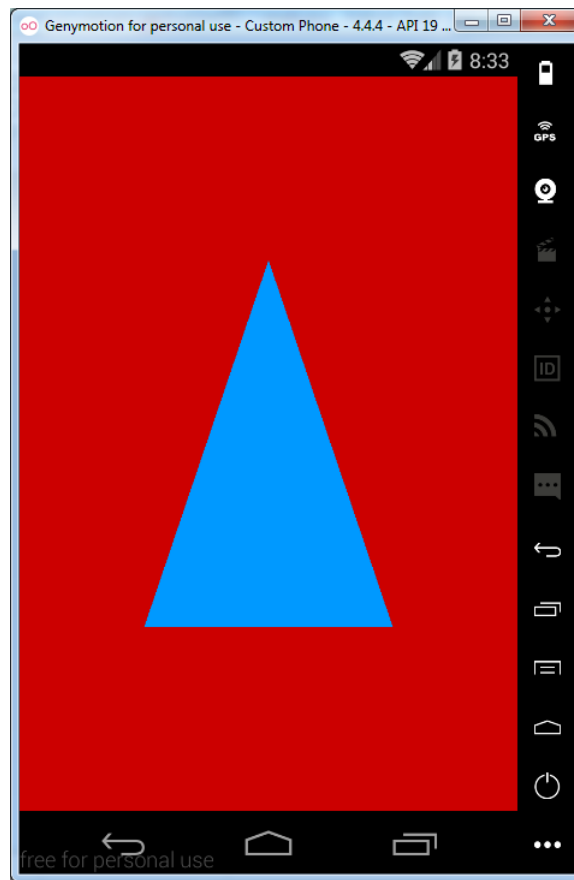
```

70
71 // add the vertex shader to program
72 GLES20.glAttachShader(mProgram, vertexShader);
73
74 // add the fragment shader to program
75 GLES20.glAttachShader(mProgram, fragmentShader);
76
77 // creates OpenGL ES program executables
78 GLES20.glLinkProgram(mProgram);
79 }
80
81 public void draw() {
82     // Add program to OpenGL ES environment
83     GLES20.glUseProgram(mProgram);
84
85     // get handle to vertex shader's vPosition member
86     mPositionHandle =
87         GLES20.glGetAttribLocation(mProgram,
88             "vPosition");
89
90     // Enable a handle to the triangle vertices
91     GLES20.glEnableVertexAttribArray(mPositionHandle);
92
93     // Prepare the triangle coordinate data
94     GLES20.glVertexAttribPointer(mPositionHandle,
95         COORDS_PER_VERTEX,
96         GLES20.GL_FLOAT, false,
97         vertexStride, vertexBuffer);
98
99     // get handle to fragment shader's vColor member
100     mColorHandle =
101         GLES20.glGetUniformLocation(mProgram, "vColor");
102
103     // Set color for drawing the triangle
104     GLES20.glUniform4fv(mColorHandle, 1, color, 0);
105
106     // Draw the triangle
107     GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0,
108         vertexCount);
109
110     // Disable vertex array
111     GLES20.glDisableVertexAttribArray(mPositionHandle);
112 }
113 }

```

---

- Sau khi biên dịch được kết quả như hình 3.2.



Hình 3.2: Sử dụng OpenGL ES 2.0 vẽ đối tượng là một tam giác

### 3.3.2 Vẽ đối tượng là khối lập phương

a. Xoay khối lập phương khi chạm vào:

- Source code của project được lưu ở địa chỉ:  
<https://github.com/minhdatvnus/nienluan3/blob/master/CubeOpenGLS20.zip>
- Nội dung của các file chương trình:
  - Nội dung của các file `AndroidManifest.xml`:

---

```

1  <manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
2  package="com.opengles.android.cubeopengles20">
3
4  <uses-feature android:glEsVersion="0x00020000"
    android:required="true" />
5
6  <application
7  android:allowBackup="true"

```

```

8         android:icon="@mipmap/ic_launcher"
9         android:label="@string/app_name"
10        android:supportsRtl="true"
11        android:theme="@style/AppTheme">
12
13        <activity
14            android:name=
15            "com.opengles.android.cubeopengles20.MainActivity"
16            android:label="@string/app_name">
17            <intent-filter>
18                <action
19                    android:name="android.intent.action.MAIN"
20                />
21
22                <category
23                    android:name="android.intent.category.LAUNCHER"
24                />
25            </intent-filter>
26        </activity>
27    </application>
28</manifest>

```

---

#### – Nội dung của file MainActivity.java:

---

```

1 package com.opengles.android.cubeopengles20;
2
3 import android.app.Activity;
4 import android.opengl.GLSurfaceView;
5 import android.os.Bundle;
6
7 public class MainActivity extends Activity {
8     private GLSurfaceView mGLView;
9
10    @Override
11    public void onCreate(Bundle savedInstanceState) {
12        super.onCreate(savedInstanceState);
13
14        mGLView = new CubeGLSurfaceView(this);
15        setContentView(mGLView);
16    }
17 }

```

---

#### – Nội dung của file CubeGLSurfaceView.java:

---

```

1 package com.opengles.android.cubeopengles20;
2 import android.content.Context;
3 import android.opengl.GLES20;
4 import android.opengl.GLSurfaceView;
5 import android.opengl.Matrix;
6 import android.util.Log;
7 import android.view.MotionEvent;

```



```

8  import java.nio.ByteBuffer;
9  import java.nio.ByteOrder;
10 import java.nio.FloatBuffer;
11 import java.nio.ShortBuffer;
12 import javax.microedition.khronos.egl.EGLConfig;
13 import javax.microedition.khronos.opengles.GL10;
14
15 public class CubeGLSurfaceView extends GLSurfaceView {
16     CubeRenderer mRenderer;
17     private float mPreviousX;
18     private float mPreviousY;
19     private float mPreviousDeg;
20
21     public CubeGLSurfaceView(Context context) {
22         super(context);
23         setEGLContextClientVersion(2);
24         setRenderer(mRenderer = new CubeRenderer());
25         setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
26     }
27
28     @Override
29     public void onPause() {
30     }
31
32     @Override
33     public void onResume() {
34     }
35
36     @Override
37     public boolean onTouchEvent(MotionEvent event) {
38         if (event != null) {
39             System.out.println();
40             if (event.getPointerCount() == 1) {
41                 float x = event.getX();
42                 float y = event.getY();
43
44                 if (event.getAction() ==
45                     MotionEvent.ACTION_MOVE) {
46                     if (mRenderer != null) {
47                         float deltaX = (x - mPreviousX) /
48                             this.getWidth() * 360;
49                         float deltaY = (y - mPreviousY) /
50                             this.getHeight() * 360;
51                         mRenderer.mDeltaX += deltaX;
52                         mRenderer.mDeltaY += deltaY;
53                     }
54                 }
55                 mPreviousX = x;
56                 mPreviousY = y;
57             }
58             else if (event.getPointerCount() == 2) {

```

```

56         float dx = event.getX(1)-event.getX(0);
57         float dy = event.getY(1)-event.getY(0);
58         float deg =
            (float) Math.toDegrees(Math.atan2(dy,
            dx));
59         if(event.getAction() !=
            MotionEvent.ACTION_MOVE) {
60             mPreviousDeg = deg;
61             mPreviousX = event.getX();
62             mPreviousY = event.getY();
63             return true;
64         }
65         float ddeg = deg-mPreviousDeg;
66         mRenderer.mDeltaZ -= ddeg;
67         mPreviousDeg = deg;
68     }
69     requestRender();
70 }
71 return true;
72 }
73
74 public void spinCube(float dx, float dy, float dz){
75     mRenderer.mDeltaX += dx;
76     mRenderer.mDeltaY += dy;
77     mRenderer.mDeltaZ += dz;
78     requestRender();
79 }
80
81 private class CubeRenderer implements Renderer {
82     volatile public float mDeltaX, mDeltaY, mDeltaZ;
83
84     int iProgId;
85     int iPosition;
86     int iVPMatrix;
87     int iTexId;
88     int iTexLoc;
89     int iTexCoords;
90
91     float[] m_fProjMatrix = new float[16];
92     float[] m_fViewMatrix = new float[16];
93     float[] m_fIdentity = new float[16];
94     float[] m_fVPMatrix = new float[16];
95
96     private float[] mAccumulatedRotation = new
        float[16];
97     private float[] mCurrentRotation = new float[16];
98     private float[] mTemporaryMatrix = new float[16];
99
100    float[] cube = {
101        2,2,2, -2,2,2, -2,-2,2, 2,-2,2, //0-1-2-3
        front
    }

```

```

102         2,2,2, 2,-2,2, 2,-2,-2, 2,2,-2, //0-3-4-5
           right
103         2,-2,-2, -2,-2,-2, -2,2,-2,
           2,2,-2, //4-7-6-5 back
104         -2,2,2, -2,2,-2, -2,-2,-2,
           -2,-2,2, //1-6-7-2 left
105         2,2,2, 2,2,-2, -2,2,-2, -2,2,2, //top
106         2,-2,2, -2,-2,2, -2,-2,-2, 2,-2,-2, //bottom
107     };
108
109
110     short[] indices = {
111         0,1,2, 0,2,3,
112         4,5,6, 4,6,7,
113         8,9,10, 8,10,11,
114         12,13,14, 12,14,15,
115         16,17,18, 16,18,19,
116         20,21,22, 20,22,23,
117     };
118
119     float[] tex = {
120         1,1,1, -1,1,1, -1,-1,1, 1,-1,1, //0-1-2-3
           front
121         1,1,1, 1,-1,1, 1,-1,-1, 1,1,-1, //0-3-4-5
           right
122         1,-1,-1, -1,-1,-1, -1,1,-1,
           1,1,-1, //4-7-6-5 back
123         -1,1,1, -1,1,-1, -1,-1,-1,
           -1,-1,1, //1-6-7-2 left
124         1,1,1, 1,1,-1, -1,1,-1, -1,1,1, //top
125         1,-1,1, -1,-1,1, -1,-1,-1, 1,-1,-1, //bottom
126     };
127
128     final String strVShader =
129         "attribute vec4 a_position;" +
130         "attribute vec4 a_color;" +
131         "attribute vec3 a_normal;" +
132         "uniform mat4 u_VPMatrix;" +
133         "uniform vec3 u_LightPos;" +
134         "varying vec3 v_texCoords;" +
135         "attribute vec3 a_texCoords;" +
136         "void main() " +
137         "{" +
138         "v_texCoords = a_texCoords;" +
139         "gl_Position = u_VPMatrix *
           a_position;" +
140         "}";
141
142     final String strFShader =
143         "precision mediump float;" +
144         "uniform samplerCube u_texId;" +

```

```

145         "varying vec3 v_texCoords;" +
146         "void main()" +
147         "{" +
148         "gl_FragColor = textureCube(u_texId,"
149         "    v_texCoords);" +
150         "}";
151
152 FloatBuffer cubeBuffer = null;
153 FloatBuffer colorBuffer = null;
154 ShortBuffer indexBuffer = null;
155 FloatBuffer texBuffer = null;
156 FloatBuffer normBuffer = null;
157 public CubeRenderer()
158 {
159     cubeBuffer =
160         ByteBuffer.allocateDirect(cube.length *
161             4).order(ByteOrder.nativeOrder()).asFloatBuffer();
162     cubeBuffer.put(cube).position(0);
163
164     indexBuffer =
165         ByteBuffer.allocateDirect(indeces.length *
166             4).order(ByteOrder.nativeOrder()).asShortBuffer();
167     indexBuffer.put(indeces).position(0);
168
169     texBuffer =
170         ByteBuffer.allocateDirect(tex.length *
171             4).order(ByteOrder.nativeOrder()).asFloatBuffer();
172     texBuffer.put(tex).position(0);
173 }
174
175 public void onDrawFrame(GL10 arg0) {
176     GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT |
177         GLES20.GL_DEPTH_BUFFER_BIT);
178     GLES20.glUseProgram(iProgId);
179
180     cubeBuffer.position(0);
181     GLES20.glVertexAttribPointer(iPosition, 3,
182         GLES20.GL_FLOAT, false, 0, cubeBuffer);
183     GLES20.glEnableVertexAttribArray(iPosition);
184
185     texBuffer.position(0);
186     GLES20.glVertexAttribPointer(iTexCoords, 3,
187         GLES20.GL_FLOAT, false, 0, texBuffer);
188     GLES20.glEnableVertexAttribArray(iTexCoords);
189
190     GLES20.glActiveTexture(GLES20.GL_TEXTURE0);
191     GLES20.glBindTexture(GLES20.GL_TEXTURE_CUBE_MAP,
192         iTexId);
193     GLES20.glUniform1i(iTexLoc, 0);
194
195     Matrix.setIdentityM(m_fIdentity, 0);

```

```

185
186     Matrix.setIdentityM(mCurrentRotation, 0);
187     Matrix.rotateM(mCurrentRotation, 0, mDeltaX,
188         1.0f, 0.0f, 0.0f);
189     Matrix.rotateM(mCurrentRotation, 0, mDeltaY,
190         0.0f, 1.0f, 0.0f);
191     Matrix.rotateM(mCurrentRotation, 0, mDeltaZ,
192         0.0f, 0.0f, 1.0f);
193     mDeltaX = 0.0f;
194     mDeltaY = 0.0f;
195     mDeltaZ = 0.0f;
196
197     Matrix.multiplyMM(mTemporaryMatrix, 0,
198         mCurrentRotation, 0, mAccumulatedRotation,
199         0);
200     System.arraycopy(mTemporaryMatrix, 0,
201         mAccumulatedRotation, 0, 16);
202
203     Matrix.multiplyMM(mTemporaryMatrix, 0,
204         m_fIdentity, 0, mAccumulatedRotation, 0);
205     System.arraycopy(mTemporaryMatrix, 0,
206         m_fIdentity, 0, 16);
207
208     Matrix.multiplyMM(m_fVPMatrix, 0,
209         m_fViewMatrix, 0, m_fIdentity, 0);
210     Matrix.multiplyMM(m_fVPMatrix, 0,
211         m_fProjMatrix, 0, m_fVPMatrix, 0);
212
213     GLES20.glUniformMatrix4fv(iVPMatrix, 1, false,
214         m_fVPMatrix, 0);
215
216     GLES20.glDrawElements(GLES20.GL_TRIANGLES,
217         36, GLES20.GL_UNSIGNED_SHORT, indexBuffer);
218 }
219
220 public void onSurfaceChanged(GL10 arg0, int
221     width, int height) {
222     GLES20.glViewport(0, 0, width, height);
223     Matrix.frustumM(m_fProjMatrix, 0,
224         -(float)width/height, (float)width/height,
225         -1, 1, 1, 10);
226 }
227
228 public void onSurfaceCreated(GL10 arg0, EGLConfig
229     arg1) {
230     GLES20.glClearColor(0, 0, 0, 0);
231     GLES20.glEnable(GLES20.GL_DEPTH_TEST);
232     GLES20.glDepthFunc(GLES20.GL_LEQUAL);
233     GLES20.glFrontFace(GLES20.GL_CCW);
234     GLES20.glEnable(GLES20.GL_CULL_FACE);
235     GLES20.glCullFace(GLES20.GL_BACK);

```

```

220     GLES20.glEnable(GLES20.GL_BLEND);
221     GLES20.glBlendFunc(GLES20.GL_SRC_ALPHA,
        GLES20.GL_ONE_MINUS_SRC_ALPHA);
222
223     Matrix.setLookAtM(m_fViewMatrix, 0, 0, 0, 6,
        0, 0, 0, 0, 1, 0);
224     Matrix.setIdentityM(mAccumulatedRotation, 0);
225
226     iProgId = loadProgram(strVShader, strFShader);
227     iPosition =
        GLES20.glGetAttribLocation(iProgId,
            "a_position");
228     iVPMatrix =
        GLES20.glGetUniformLocation(iProgId,
            "u_VPMatrix");
229     iTexLoc = GLES20.glGetUniformLocation(iProgId,
        "u_texId");
230     iTexCoords =
        GLES20.glGetAttribLocation(iProgId,
            "a_texCoords");
231     iTexId = CreateCubeTexture();
232 }
233
234 public int CreateCubeTexture() {
235     int[] textureId = new int[1];
236
237     byte[] cubePixels0 = { 127, 0, 0 }; // Face 0
        - Red
238     byte[] cubePixels1 = { 0, 127, 0 }; // Face 1
        - Green
239     byte[] cubePixels2 = { 0, 0, 127 }; // Face 2
        - Blue
240     byte[] cubePixels3 = { 127, 127, 0 }; // Face
        3 - Yellow
241     byte[] cubePixels4 = { 127, 0, 127 }; // Face
        4 - Purple
242     byte[] cubePixels5 = { 127, 127, 127 }; //
        Face 5 - White
243
244     ByteBuffer cubePixels =
        ByteBuffer.allocateDirect(3);
245     GLES20.glGenTextures(1, textureId, 0 );
246     GLES20.glBindTexture(GLES20.GL_TEXTURE_CUBE_MAP,
        textureId[0] );
247
248     // Load the cube face - Positive X
249     cubePixels.put(cubePixels0).position(0);
250     GLES20.glTexImage2D (
        GLES20.GL_TEXTURE_CUBE_MAP_POSITIVE_X, 0,
251     GLES20.GL_RGB, 1, 1, 0,
252         GLES20.GL_RGB,

```

```

253                                     GLES20.GL_UNSIGNED_BYTE,
254                                     cubePixels );
255
256 // Load the cube face - Negative X
257 cubePixels.put(cubePixels1).position(0);
258 GLES20.glTexImage2D (
259     GLES20.GL_TEXTURE_CUBE_MAP_NEGATIVE_X, 0,
260     GLES20.GL_RGB, 1, 1, 0,
261     GLES20.GL_RGB,
262     GLES20.GL_UNSIGNED_BYTE,
263     cubePixels );
264
265 // Load the cube face - Positive Y
266 cubePixels.put(cubePixels2).position(0);
267 GLES20.glTexImage2D (
268     GLES20.GL_TEXTURE_CUBE_MAP_POSITIVE_Y, 0,
269     GLES20.GL_RGB, 1, 1, 0,
270     GLES20.GL_RGB,
271     GLES20.GL_UNSIGNED_BYTE,
272     cubePixels );
273
274 // Load the cube face - Negative Y
275 cubePixels.put(cubePixels3).position(0);
276 GLES20.glTexImage2D (
277     GLES20.GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, 0,
278     GLES20.GL_RGB, 1, 1, 0,
279     GLES20.GL_RGB,
280     GLES20.GL_UNSIGNED_BYTE,
281     cubePixels );
282
283 // Load the cube face - Positive Z
284 cubePixels.put(cubePixels4).position(0);
285 GLES20.glTexImage2D (
286     GLES20.GL_TEXTURE_CUBE_MAP_POSITIVE_Z, 0,
287     GLES20.GL_RGB, 1, 1, 0,
288     GLES20.GL_RGB,
289     GLES20.GL_UNSIGNED_BYTE,
290     cubePixels );
291
292 // Load the cube face - Negative Z
293 cubePixels.put(cubePixels5).position(0);
294 GLES20.glTexImage2D (
295     GLES20.GL_TEXTURE_CUBE_MAP_NEGATIVE_Z, 0,
296     GLES20.GL_RGB, 1, 1, 0,
297     GLES20.GL_RGB,
298     GLES20.GL_UNSIGNED_BYTE,
299     cubePixels );
300
301 // Set the filtering mode
302 GLES20.glTexParameterf (
303     GLES20.GL_TEXTURE_CUBE_MAP,

```

```

286         GLES20.GL_TEXTURE_MIN_FILTER,
287         GLES20.GL_NEAREST );
288     GLES20.glTexParameterf (
289         GLES20.GL_TEXTURE_CUBE_MAP,
290         GLES20.GL_TEXTURE_MAG_FILTER,
291         GLES20.GL_NEAREST );
292
293     return textureId[0];
294 }
295 }
296
297 public static int loadShader(String strSource, int
298     iType) {
299     int[] compiled = new int[1];
300     int iShader = GLES20.glCreateShader(iType);
301     GLES20.glShaderSource(iShader, strSource);
302     GLES20.glCompileShader(iShader);
303     GLES20.glGetShaderiv(iShader,
304         GLES20.GL_COMPILE_STATUS, compiled, 0);
305     if (compiled[0] == 0) {
306         Log.d("Load Shader Failed",
307             "Compilation\n"+GLES20.glGetShaderInfoLog(iShader));
308         return 0;
309     }
310     return iShader;
311 }
312
313 public static int loadProgram(String strVSource,
314     String strFSource) {
315     int iVShader;
316     int iFShader;
317     int iProgId;
318     int[] link = new int[1];
319     iVShader = loadShader(strVSource,
320         GLES20.GL_VERTEX_SHADER);
321     if (iVShader == 0) {
322         Log.d("Load Program", "Vertex Shader Failed");
323         return 0;
324     }
325     iFShader = loadShader(strFSource,
326         GLES20.GL_FRAGMENT_SHADER);
327     if (iFShader == 0) {
328         Log.d("Load Program", "Fragment Shader
329             Failed");
330         return 0;
331     }
332
333     iProgId = GLES20.glCreateProgram();
334
335     GLES20.glAttachShader(iProgId, iVShader);
336     GLES20.glAttachShader(iProgId, iFShader);

```

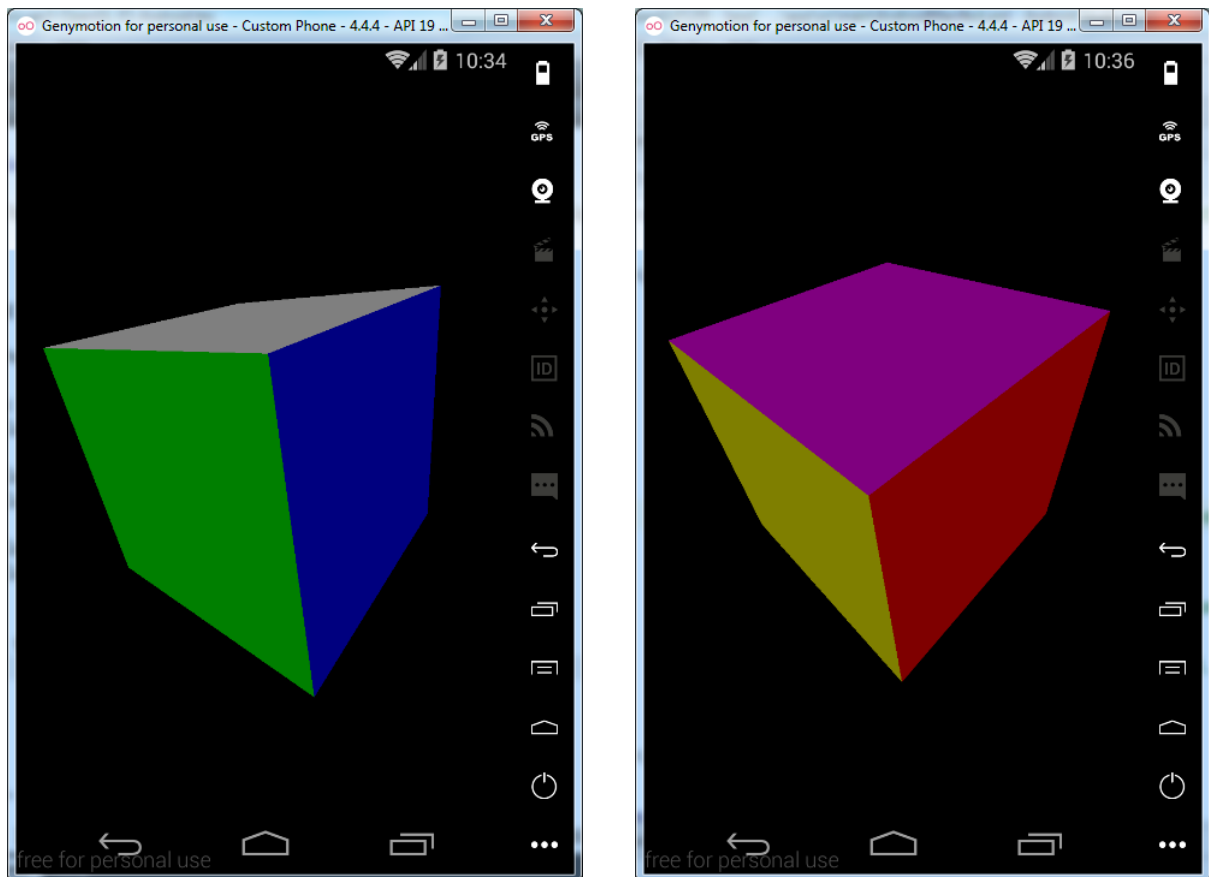


```

325
326         GLES20.glLinkProgram(iProgId);
327
328         GLES20.glGetProgramiv(iProgId,
329                               GLES20.GL_LINK_STATUS, link, 0);
329         if (link[0] <= 0) {
330             Log.d("Load Program", "Linking Failed");
331             return 0;
332         }
333         GLES20.glDeleteShader(iVShader);
334         GLES20.glDeleteShader(iFShader);
335         return iProgId;
336     }
337 }

```

- Kết quả khi chạy chương trình: các mặt của khối lập phương được thể hiện trên hình 3.3.



Hình 3.3: Vẽ và xoay hình lập phương sử dụng OpenGL ES 2.0 trên Android

*b. Tự động xoay khối lập phương:* Nội dung của các file chương trình:

- Source code của project được lưu ở địa chỉ:  
<https://github.com/minhdatvnus/nienluan3/blob/master/CubeRotateOpenGLES20.zip>

- Nội dung của các file AndroidManifest.xml:

---

```
1 <manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
2   package="com.opengles.android.cuberotateopengles20">
3
4   <uses-feature android:glEsVersion="0x00020000"
     android:required="true" />
5
6   <application
7     android:allowBackup="true"
8     android:icon="@mipmap/ic_launcher"
9     android:label="@string/app_name"
10    android:supportsRtl="true"
11    android:theme="@style/AppTheme">
12
13     <activity
14       android:name=
15       "com.opengles.android.cuberotateopengles20.MainActivity"
16       android:label="@string/app_name">
17       <intent-filter>
18         <action
19           android:name="android.intent.action.MAIN" />
20
21         <category
22           android:name="android.intent.category.LAUNCHER"
23           />
24       </intent-filter>
25     </activity>
26   </application>
27 </manifest>
```

---

- Nội dung của file MainActivity.java:

---

```
1 package com.opengles.android.cuberotateopengles20;
2 import android.app.Activity;
3 import android.opengl.GLSurfaceView;
4 import android.os.Bundle;
5 import android.view.Window;
6 import android.view.WindowManager;
7
8 public class MainActivity extends Activity {
9     @Override
10    public void onCreate(Bundle savedInstanceState) {
11        super.onCreate(savedInstanceState);
12
13        requestWindowFeature(Window.FEATURE_NO_TITLE);
14        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
15                             WindowManager.LayoutParams.FLAG_FULLSCREEN);
16    }
```

```

17         GLSurfaceView view = new GLSurfaceView(this);
18         view.setRenderer(new CubeRotateGLSurfaceView());
19         setContentView(view);
20     }
21 }

```

---

- Nội dung của file CubeRotateGLSurfaceView.java:

---

```

1  package com.opengles.android.cuberotateopengles20;
2  import android.opengl.GLSurfaceView;
3  import android.opengl.GLU;
4  import javax.microedition.khronos.egl.EGLConfig;
5  import javax.microedition.khronos.opengles.GL10;
6
7  class CubeRotateGLSurfaceView implements
8      GLSurfaceView.Renderer {
9      private Cube mCube = new Cube();
10     private float mCubeRotation;
11
12     @Override
13     public void onSurfaceCreated(GL10 gl, EGLConfig config)
14     {
15         gl.glClearColor(0.0f, 0.0f, 0.0f, 0.5f);
16         gl.glClearDepthf(1.0f);
17         gl.glEnable(GL10.GL_DEPTH_TEST);
18         gl.glDepthFunc(GL10.GL_LEQUAL);
19         gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,
20                 GL10.GL_NICEST);
21     }
22
23     @Override
24     public void onDrawFrame(GL10 gl) {
25         gl.glClear(GL10.GL_COLOR_BUFFER_BIT |
26                 GL10.GL_DEPTH_BUFFER_BIT);
27         gl.glLoadIdentity();
28         gl.glTranslatef(0.0f, 0.0f, -10.0f);
29         gl.glRotatef(mCubeRotation, 1.0f, 1.0f, 1.0f);
30         mCube.draw(gl);
31         gl.glLoadIdentity();
32         mCubeRotation -= 0.5f; // Toc do xoay
33     }
34
35     @Override
36     public void onSurfaceChanged(GL10 gl, int width, int
37         height) {
38         gl.glViewport(0, 0, width, height);
39         gl.glMatrixMode(GL10.GL_PROJECTION);
40         gl.glLoadIdentity();
41         GLU.gluPerspective(gl, 45.0f, (float)width /
42             (float)height, 0.1f, 100.0f);
43         gl.glViewport(0, 0, width, height);

```

```

39         gl.glMatrixMode(GL10.GL_MODELVIEW);
40         gl.glLoadIdentity();
41     }
42 }

```

---

## • Nội dung của file Cube.java

---

```

1  package com.opengles.android.cuberotateopengles20;
2  import android.opengl.GLES20;
3  import java.nio.ByteBuffer;
4  import java.nio.ByteOrder;
5  import java.nio.FloatBuffer;
6  import javax.microedition.khronos.opengles.GL10;
7
8  class Cube {
9      private FloatBuffer mVertexBuffer;
10     private FloatBuffer mColorBuffer;
11     private ByteBuffer mIndexBuffer;
12
13     private float vertices[] = {
14         -1.0f, -1.0f, -1.0f,
15         1.0f, -1.0f, -1.0f,
16         1.0f, 1.0f, -1.0f,
17         -1.0f, 1.0f, -1.0f,
18         -1.0f, -1.0f, 1.0f,
19         1.0f, -1.0f, 1.0f,
20         1.0f, 1.0f, 1.0f,
21         -1.0f, 1.0f, 1.0f
22     };
23     private float colors[] = {
24         0.0f, 1.0f, 0.0f, 1.0f,
25         0.0f, 1.0f, 0.0f, 1.0f,
26         1.0f, 0.5f, 0.0f, 1.0f,
27         1.0f, 0.5f, 0.0f, 1.0f,
28         1.0f, 0.0f, 0.0f, 1.0f,
29         1.0f, 0.0f, 0.0f, 1.0f,
30         0.0f, 0.0f, 1.0f, 1.0f,
31         1.0f, 0.0f, 1.0f, 1.0f
32     };
33
34     private byte indices[] = {
35         0, 4, 5, 0, 5, 1,
36         1, 5, 6, 1, 6, 2,
37         2, 6, 7, 2, 7, 3,
38         3, 7, 4, 3, 4, 0,
39         4, 7, 6, 4, 6, 5,
40         3, 0, 1, 3, 1, 2
41     };
42
43     public Cube() {
44         ByteBuffer byteBuf =

```

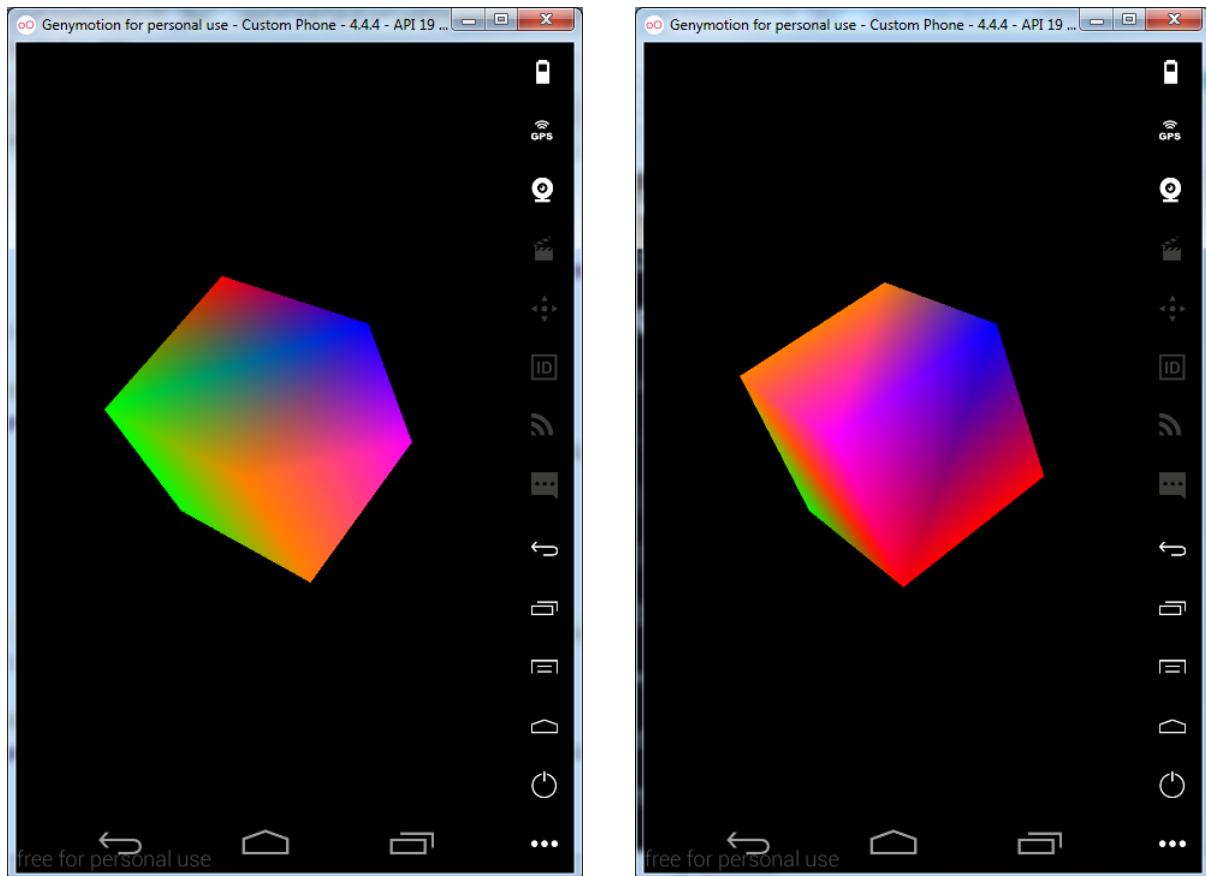
```

45         ByteBuffer.allocateDirect(vertices.length * 4);
46         byteBuf.order(ByteOrder.nativeOrder());
47         mVertexBuffer = byteBuf.asFloatBuffer();
48         mVertexBuffer.put(vertices);
49         mVertexBuffer.position(0);
50
51         byteBuf = ByteBuffer.allocateDirect(colors.length *
52             4);
53         byteBuf.order(ByteOrder.nativeOrder());
54         mColorBuffer = byteBuf.asFloatBuffer();
55         mColorBuffer.put(colors);
56         mColorBuffer.position(0);
57
58         mIndexBuffer =
59             ByteBuffer.allocateDirect(indices.length);
60         mIndexBuffer.put(indices);
61         mIndexBuffer.position(0);
62     }
63
64     public void draw(GL10 gl) {
65         gl.glFrontFace(GLES20.GL_CW);
66         gl.glVertexPointer(3, GLES20.GL_FLOAT, 0,
67             mVertexBuffer);
68         gl.glColorPointer(4, GLES20.GL_FLOAT, 0,
69             mColorBuffer);
70
71         gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
72         gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
73
74         gl.glDrawElements(GLES20.GL_TRIANGLES, 36,
75             GLES20.GL_UNSIGNED_BYTE,
76             mIndexBuffer);
77
78         gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
79         gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
80     }
81 }

```

---

- Kết quả khi chạy chương trình: các mặt của khối lập phương tự động xoay, được thể hiện trên hình 3.4.



Hình 3.4: Tự động xoay khối lập phương sử dụng OpenGL trên Android

### 3.3.3 Vẽ đối tượng là khối cầu

Nội dung của các file chương trình:

- Source code của project được lưu ở địa chỉ: <https://github.com/minhdatvnus/nienluan3/blob/master/CircleOpenGLS20.zip>
- Nội dung của các file `AndroidManifest.xml`:

```

1 <manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
2   package="com.opengles.android.circleopengles20">
3
4   <uses-feature android:glEsVersion="0x00020000"
     android:required="true" />
5
6   <application
7     android:allowBackup="true"
8     android:icon="@mipmap/ic_launcher"
9     android:label="@string/app_name"

```

```

10     android:supportsRtl="true"
11     android:theme="@style/AppTheme">
12
13     <activity
14         android:name=
15         "com.opengles.android.circleopengles20.MainActivity"
16         android:label="@string/app_name">
17         <intent-filter>
18             <action
19                 android:name="android.intent.action.MAIN" />
20
21             <category
22                 android:name="android.intent.category.LAUNCHER"
23                 />
24             </intent-filter>
25         </activity>
26     </application>
27 </manifest>

```

---

- Nội dung của file MainActivity.java:

```

1 package com.opengles.android.circleopengles20;
2
3 import android.app.Activity;
4 import android.os.Bundle;
5 import android.view.Window;
6 import android.view.WindowManager;
7
8 public class MainActivity extends Activity {
9
10     private CircleOpenGLSurfaceView mOpenGLView;
11
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         requestWindowFeature(Window.FEATURE_NO_TITLE);
16         getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
17         WindowManager.LayoutParams.FLAG_FULLSCREEN);
18         mOpenGLView = new CircleOpenGLSurfaceView(this);
19         setContentView(mOpenGLView);
20     }
21 }

```

---

- Nội dung của file CircleOpenGLSurfaceView.java:

```

1 package com.opengles.android.circleopengles20;
2 import android.content.Context;
3 import android.opengl.GLSurfaceView;
4 import android.view.MotionEvent;
5

```

```

6 public class CircleOpenGLSurfaceView extends GLSurfaceView {
7     private CircleRenderer mRenderer;
8
9     private float mDownX = 0.0f;
10    private float mDownY = 0.0f;
11
12    public CircleOpenGLSurfaceView(Context context) {
13        super(context);
14        mRenderer = new CircleRenderer();
15        this.setRenderer(mRenderer);
16    }
17
18    @Override
19    public boolean onTouchEvent(MotionEvent event) {
20        int action = event.getActionMasked();
21        switch (action) {
22            case MotionEvent.ACTION_DOWN:
23                mDownX = event.getX();
24                mDownY = event.getY();
25                return true;
26            case MotionEvent.ACTION_UP:
27                return true;
28            case MotionEvent.ACTION_MOVE:
29                float mX = event.getX();
30                float mY = event.getY();
31                mRenderer.mLightX += (mX-mDownX)/10;
32                mRenderer.mLightY -= (mY-mDownY)/10;
33                mDownX = mX;
34                mDownY = mY;
35                return true;
36            default:
37                return super.onTouchEvent(event);
38        }
39    }
40 }

```

---

- Nội dung của file CircleRendererer.java:

---

```

1 package com.opengles.android.circleopengles20;
2
3 import android.opengl.GLES20;
4 import android.opengl.GLSurfaceView;
5 import android.opengl.GLU;
6
7 import java.nio.ByteBuffer;
8 import java.nio.ByteOrder;
9 import java.nio.FloatBuffer;
10
11 import javax.microedition.khronos.egl.EGLConfig;
12 import javax.microedition.khronos.opengles.GL10;
13

```



```

14 public class CircleRenderer implements
    GLSurfaceView.Renderer {
15
16     private final float[] mat_ambient = { 0.2f, 0.3f, 0.4f,
        1.0f };
17     private FloatBuffer mat_ambient_buf;
18
19     private final float[] mat_diffuse = { 0.4f, 0.6f, 0.8f,
        1.0f };
20     private FloatBuffer mat_diffuse_buf;
21
22     private final float[] mat_specular = { 0.2f * 0.4f, 0.2f
        * 0.6f, 0.2f * 0.8f, 1.0f };
23     private FloatBuffer mat_specular_buf;
24
25     private Sphere mSphere = new Sphere();
26
27     public volatile float mLightX = 10f;
28     public volatile float mLightY = 10f;
29     public volatile float mLightZ = 10f;
30
31     @Override
32     public void onDrawFrame(GL10 gl) {
33         gl.glClear(GLES20.GL_COLOR_BUFFER_BIT |
            GLES20.GL_DEPTH_BUFFER_BIT);
34         gl.glLoadIdentity();
35
36         gl.glEnable(GL10.GL_LIGHTING);
37         gl.glEnable(GL10.GL_LIGHT0);
38
39         gl.glMaterialfv(GLES20.GL_FRONT_AND_BACK,
            GL10.GL_AMBIENT, mat_ambient_buf);
40         gl.glMaterialfv(GLES20.GL_FRONT_AND_BACK,
            GL10.GL_DIFFUSE, mat_diffuse_buf);
41         gl.glMaterialfv(GLES20.GL_FRONT_AND_BACK,
            GL10.GL_SPECULAR, mat_specular_buf);
42         gl.glMaterialf(GLES20.GL_FRONT_AND_BACK,
            GL10.GL_SHININESS, 96.0f);
43
44         float[] light_position = {mLightX, mLightY, mLightZ,
            0.0f};
45         ByteBuffer mpbb =
            ByteBuffer.allocateDirect(light_position.length*4);
46         mpbb.order(ByteOrder.nativeOrder());
47         FloatBuffer mat_posiBuf = mpbb.asFloatBuffer();
48         mat_posiBuf.put(light_position);
49         mat_posiBuf.position(0);
50         gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_POSITION,
            mat_posiBuf);
51
52         gl.glTranslatef(0.0f, 0.0f, -3.0f);

```

```

53     mSphere.draw(gl);
54 }
55
56 @Override
57 public void onSurfaceChanged(GL10 gl, int width, int
    height) {
58     gl.glViewport(0, 0, width, height);
59     gl.glMatrixMode(GL10.GL_PROJECTION);
60     gl.glLoadIdentity();
61
62     GLU.gluPerspective(gl, 90.0f, (float) width / height,
        0.1f, 50.0f);
63
64     gl.glMatrixMode(GL10.GL_MODELVIEW);
65     gl.glLoadIdentity();
66 }
67
68 @Override
69 public void onSurfaceCreated(GL10 gl, EGLConfig arg1) {
70     gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,
        GLES20.GL_FASTEST);
71     gl.glClearColor(0, 0.0f, 0.0f, 0.0f);
72     gl.glShadeModel(GL10.GL_SMOOTH);
73
74     gl.glClearDepthf(1.0f);
75     gl.glEnable(GLES20.GL_DEPTH_TEST);
76     gl.glDepthFunc(GLES20.GL_LEQUAL);
77
78     initBuffers();
79 }
80
81 private void initBuffers() {
82     ByteBuffer bufTemp =
        ByteBuffer.allocateDirect(mat_ambient.length * 4);
83     bufTemp.order(ByteOrder.nativeOrder());
84     mat_ambient_buf = bufTemp.asFloatBuffer();
85     mat_ambient_buf.put(mat_ambient);
86     mat_ambient_buf.position(0);
87
88     bufTemp = ByteBuffer.allocateDirect(mat_diffuse.length
        * 4);
89     bufTemp.order(ByteOrder.nativeOrder());
90     mat_diffuse_buf = bufTemp.asFloatBuffer();
91     mat_diffuse_buf.put(mat_diffuse);
92     mat_diffuse_buf.position(0);
93
94     bufTemp =
        ByteBuffer.allocateDirect(mat_specular.length * 4);
95     bufTemp.order(ByteOrder.nativeOrder());
96     mat_specular_buf = bufTemp.asFloatBuffer();
97     mat_specular_buf.put(mat_specular);

```

```

98         mat_specular_buf.position(0);
99     }
100 }

```

---

- Nội dung của file Sphere.java:

---

```

1  package com.opengles.android.circleopengles20;
2  import android.opengl.GLES20;
3  import java.nio.ByteBuffer;
4  import java.nio.ByteOrder;
5  import java.nio.FloatBuffer;
6
7  import javax.microedition.khronos.opengles.GL10;
8
9  public class Sphere {
10     public void draw(GL10 gl) {
11         float angleA, angleB;
12         float cos, sin;
13         float r1, r2;
14         float h1, h2;
15         float step = 1.0f;
16         float[][] v = new float[32][3];
17         ByteBuffer vbb;
18         FloatBuffer vBuf;
19
20         vbb = ByteBuffer.allocateDirect(v.length * v[0].length
21             * 4);
22         vbb.order(ByteOrder.nativeOrder());
23         vBuf = vbb.asFloatBuffer();
24
25         gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
26         gl.glEnableClientState(GL10.GL_NORMAL_ARRAY);
27
28         for (angleA = -90.0f; angleA < 90.0f; angleA += step) {
29             int n = 0;
30
31             r1 = (float) Math.cos(angleA * Math.PI / 180.0);
32             r2 = (float) Math.cos((angleA + step) * Math.PI /
33                 180.0);
34             h1 = (float) Math.sin(angleA * Math.PI / 180.0);
35             h2 = (float) Math.sin((angleA + step) * Math.PI /
36                 180.0);
37
38             for (angleB = 0.0f; angleB <= 360.0f; angleB +=
39                 step) {
40
41                 cos = (float) Math.cos(angleB * Math.PI / 180.0);
42                 sin = -(float) Math.sin(angleB * Math.PI / 180.0);
43
44                 v[n][0] = (r2 * cos);
45                 v[n][1] = (h2);

```

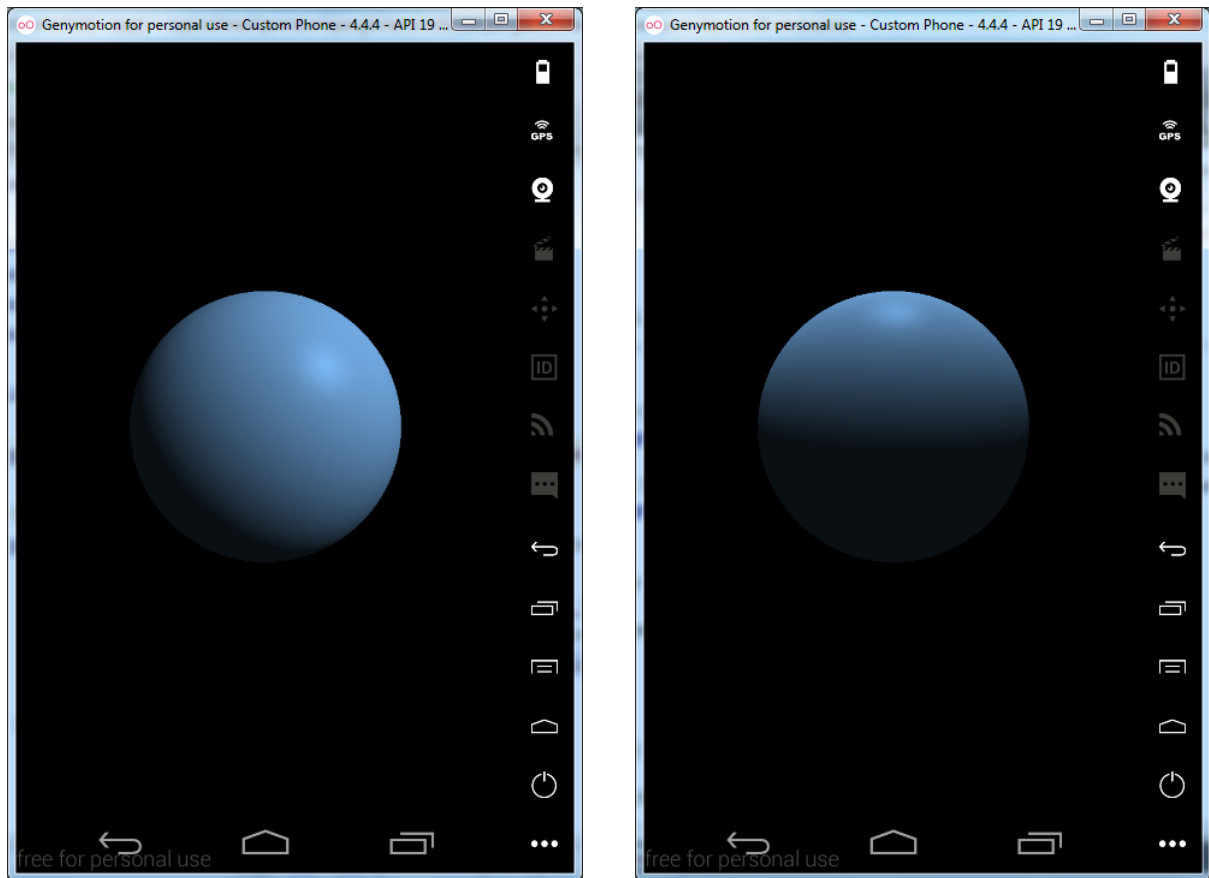
```

42         v[n][2] = (r2 * sin);
43         v[n + 1][0] = (r1 * cos);
44         v[n + 1][1] = (h1);
45         v[n + 1][2] = (r1 * sin);
46
47         vBuf.put(v[n]);
48         vBuf.put(v[n + 1]);
49
50         n += 2;
51
52         if(n>31){
53             vBuf.position(0);
54
55             gl.glVertexPointer(3, GLES20.GL_FLOAT, 0,
56                 vBuf);
57             gl.glNormalPointer(GLES20.GL_FLOAT, 0, vBuf);
58             gl.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 0,
59                 n);
60
61             n = 0;
62             angleB -= step;
63         }
64         vBuf.position(0);
65
66         gl.glVertexPointer(3, GLES20.GL_FLOAT, 0, vBuf);
67         gl.glNormalPointer(GLES20.GL_FLOAT, 0, vBuf);
68         gl.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 0, n);
69     }
70     gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
71     gl.glDisableClientState(GL10.GL_NORMAL_ARRAY);
72 }
73 }

```

---

- Kết quả khi chạy chương trình: các mặt của khối lập phương tự động xoay, được thể hiện trên hình 3.5.



Hình 3.5: Xoay khối cầu sử dụng OpenGL trên Android

# Chương 4

## Kết quả

### 4.1 Kết quả đạt được

Qua quá trình thực hiện niên luận, nhóm chúng em đạt được một số kết quả sau:

- Sử dụng được một số lệnh cơ bản trong thư viện OpenGL ES 2.0 để vẽ các đối tượng cơ bản 2D và 3D.
- Nhúng các mã lệnh của OpenGL vào chương trình Android.

### 4.2 Hạn chế

Bên cạnh kết quả đạt được, nhóm chúng em còn gặp một số hạn chế như sau:

- Còn hạn chế về kiến thức đồ họa và các kiến thức toán liên quan đến đồ họa.
- Chỉ mới sử dụng được OpenGL ở mức độ cơ bản, chưa khai thác được hết các lệnh trong thư viện.

### 4.3 Hướng phát triển

Nhóm em có định hướng phát triển đề tài như sau: “*Xây dựng phần mềm vẽ hình không gian trên điện thoại di động bằng cách sử dụng thư viện OpenGL ES phục vụ mục đích học tập.*”

Ý tưởng xây dựng phần mềm:

- Người vẽ có thể chọn vị trí các đỉnh trong hệ tọa độ rồi nối các đỉnh lại với nhau để tạo thành hình hoàn chỉnh.
- Sau khi đã vẽ được hình hoàn chỉnh, người vẽ có thể xoay hình theo các hướng để quan sát dễ dàng hơn.
- Có thể cắt khối hình ra để quan sát chi tiết bên trong.

- Giải quyết một số bài toán liên quan đến hình học không gian, phục vụ học tập: tính thể tích, diện tích, tìm tọa độ điểm, phương trình đường thẳng, mặt phẳng,...
- Giải quyết vấn đề liên quan đến vẽ kỹ thuật:
  - Từ một mô hình của vật thể trong không gian được người dùng vẽ lên, thực hiện phép chiếu theo một tiêu chuẩn cụ thể nào đó, thì chương trình sẽ tự động tìm ra được hình chiếu đứng hình, hình chiếu bằng, hình chiếu cạnh của vật thể.
  - Hoặc ngược lại từ hình chiếu đứng, hình chiếu bằng, hình chiếu cạnh, chương trình có thể tự động vẽ lại được mô hình của vật thể trong không gian.

# Tài liệu tham khảo

- [1] **Aaftab Munshi – Dan Ginsburg – Dave Shreiner**, OpenGL ES 2.0 Programming Guide – Chapter 1, Pearson Education, Inc., 2009.
- [2] **Android Developer**, Displaying Graphics with OpenGL ES  
<https://developer.android.com/training/graphics/opengl/index.html> (25/11/2016)
- [3] **Chua Hock-Chuan**, Android Programming 3D Graphics with OpenGL ES (Including Nehe's Port), Programming notes  
[https://www3.ntu.edu.sg/home/ehchua/programming/android/Android\\_3D.html#zz-2.3](https://www3.ntu.edu.sg/home/ehchua/programming/android/Android_3D.html#zz-2.3) (22/11/2016)
- [4] Asked: **CoderOfHonor** - Answered: **Reigertje**, Where is the documentation for OpenGL ES 2.0 on Android?, Stackoverflow.  
<http://stackoverflow.com/questions/17288113/where-is-the-documentation-for-opengl-es-2-0-on-android> (15/11/2016)
- [5] **Hoàng Vũ Nam**, OpenGL ES với Android, Rikkeisoft Blog  
<http://blog.rikkeisoft.com/opengl-es-voi-android/> (15/11/2016)
- [6] **Kevin Brothaler**, OpenGL ES 2 for Android – A Quick-Start Guide, The Pragmatic Programmers, LLC., 2013.
- [7] **Khoa Phạm**, Lập trình Android cơ bản: Cài đặt Android Studio, Youtube  
<https://www.youtube.com/watch?v=n1iOKgVhMoM&list=PLzrVYRai0riTIWPxOEhi1-2QmvLiw0DCb> (15/11/2016)
- [8] **Marco Dinacci**, Create a spinning cube with OpenGL ES and Android.  
<http://www.intransitione.com/blog/create-a-spinning-cube-with-opengl-es-and-android/> (09/12/2016)
- [9] **OpenGL ES Software Development Kit**, OpenGL ES 2.0 Reference Pages.  
<https://www.khronos.org/opengles/sdk/docs/man/> (20/11/2016)
- [10] **Programering**, Android OpenGL ES draw sphere.  
[www.programering.com/a/MDM3cjNwATU.html](http://www.programering.com/a/MDM3cjNwATU.html) (09/12/2016)
- [11] **SebastianJay**, Android Sample GLSurfaceView Cube, GitHub Git.  
<https://gist.github.com/SebastianJay/3316001#file-glcubeview-java> (09/12/2016)