
Introduction to
Processing
on Android devices

Introduction to Processing on Android Devices

Saturday, December 18th - 14:15-18:00 - Room E5
Presenters: Jihyun Kim, Andres Colubri

0. Orientation
Brief presentation of the course and reference materials (online/books)

PART I: Introduction to Processing for Android (14:15 – 16:00)
Presenter: Jihyun Kim

1. What is Android? (14:15 – 14:25)
2. What is Processing? (14:25 – 14:35)
3. Basic concepts in Android applications (14:35 – 14:50)
4. First steps with Processing for Android (14:50 – 15:10)
5. Basic Processing use (15:10 – 15:40)
6. Extending Processing (15:40 – 16:00)

COFEE BREAK: 16:00 – **16:15**

PART II: Fast 3D Graphics in Processing for Android
(16:15 – 18:00) Presenter: Andres Colubri

7. OpenGL and Processing (16:15 – 16:30)
8. Geometrical transformations (16:30 – 16:40)
9. Camera and perspective (16:40 – 16:50)
10. Creating 3D objects (16:50 – 17:20)
11. 3D text (17:20 – 17:30)
12. Models: the PShape3D class (17:30 – 18:00)

4. Basic concepts in Android applications
4.1 The manifest file
4.3 Views, Activities, Intents
4.4 Android Software Development Kit (SDK)
4.5 Android Development with Eclipse
4.6 Disadvantages of using Eclipse
4.6 Solution: Processing for Android!
5. How to install the Android SDK
5.1 Installation on Linux
5.2 Installation on Windows
5.3 Installation on Mac OSX
6. How to install Processing
6.1 Android Mode in Processing
6.2 What can we do with Processing for Android
7. Running Processing sketches on Android
7.1 Running on the Android Emulator
7.2 Running on the Android device
8. Basic Processing use
8.1 Drawing with basic 2D primitives (circle, rect, ellipse, shape)
8.2 Color in Processing
8.3 Animation and motion: setup and draw
8.4 Media: images, fonts
8.5 Interaction: keyboard, touchscreen

PART II: Fast 3D Graphics in Processing for Android

9. OpenGL and Processing

9.1 OpenGL ES

9.2 Hardware requirements for 3D in Android

10. The A3D renderer

11. Offscreen drawing

12. Geometrical transformations

12.1 Translations

12.2 Rotations

12.3 Scaling

12.4 The transformation matrix stack

14. Camera and perspective

14.1 Camera placement

14.2 Perspective view

14.3 Orthographic view

15. Creating 3D objects (box, spheres, vertex shapes)

16. Texturing

17. Lighting

18. 3D text

19. Models: the Pshape3D class

19.1 Introduction to Vertex Buffer Objects (VBOs)

19.2 Manual creation of Pshape3D models

19.3 OBJ loading

1.1 Online resources:

Very good Android tutorial: http://www.vogella.de/articles/Android/article.html

Official google resources
Main devel site: http://developer.android.com/index.html
SDK: http://developer.android.com/sdk/index.html
Guide: http://developer.android.com/guide/index.html
OpenGL: http://developer.android.com/guide/topics/graphics/opengl.html
Mailing list: http://groups.google.com/group/android-developers

Developers forums: http://www.anddev.org/
Book: http://andbook.anddev.org/

Cyanogenmod project: http://www.cyanogenmod.com/

GLES benchmarks: http://www.glbenchmark.com/result.jsp

Min3D (framework 3D): http://code.google.com/p/min3d/

Developer's devices: http://www.hardkernel.com/

AppInventor: http://www.appinventor.org/

1.2 Books

Hello, Android: Introducing Google's Mobile Development Platform
Ed Burnette
Pragmatic Bookshelf; 3 edition (July 20, 2010)
http://www.amazon.com/Hello-Android-Introducing-Development-Programmers/dp/1934356565/ref=sr_1_1?ie=

Professional Android 2 Application Development
Rato Meier
Wrox; 1 edition (March 1, 2010)
http://www.amazon.com/Professional-Android-Application-Development-Programmer/dp/0470565527/ref=sr_1_

Pro Android 2
Sayed Hashimi
http://www.amazon.com/Pro-Android-2-Sayed-Hashimi/dp/1430226595/ref=sr_1_1?ie=UTF8&s=books&qid=12

Getting Started with Processing
Casey Reas and Ben Fry.
Published June 2010, O'Reilly Media.
http://oreilly.com/catalog/0636920000570
http://www.amazon.com/gp/product/144937980X?ie=UTF8&tag=processing09-20&linkCode=as2&camp=1789&

Processing: A Programming Handbook
for Visual Designers and Artists
Casey Reas and Ben Fry
Published August 2007, MIT Pres

PART I:
Introduction to Processing for Android

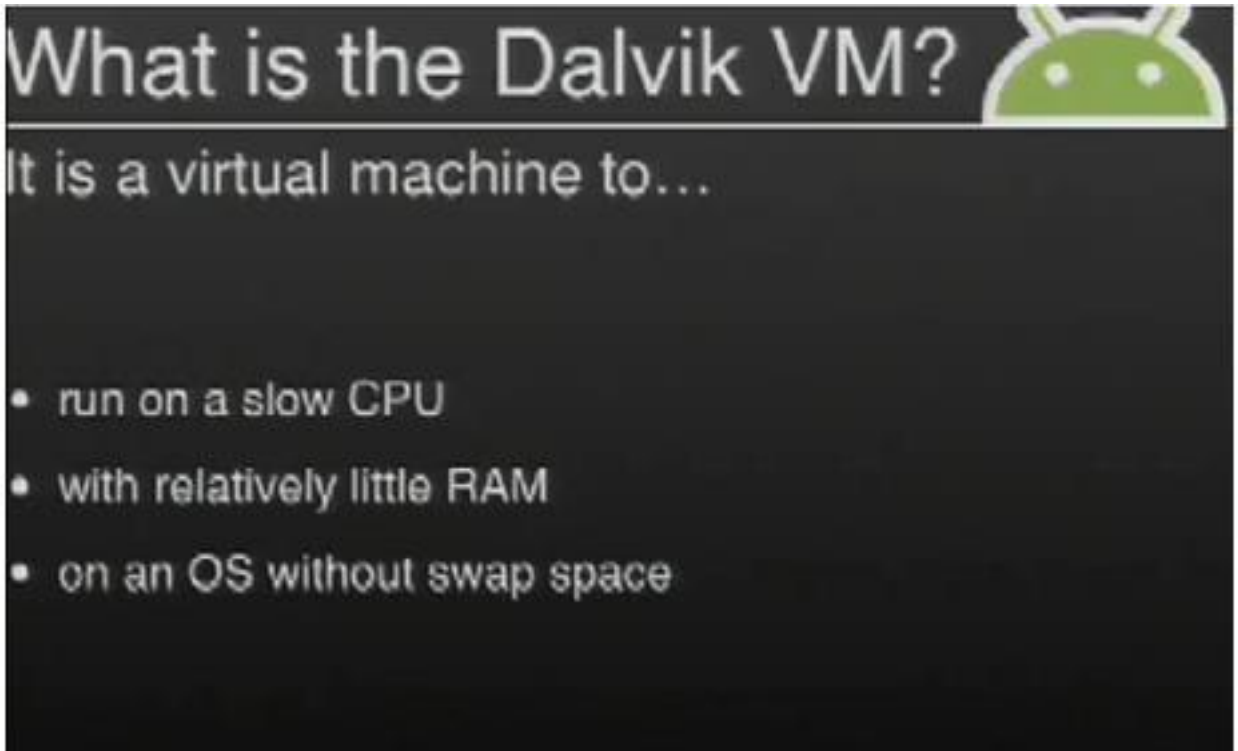
2. What is Android?

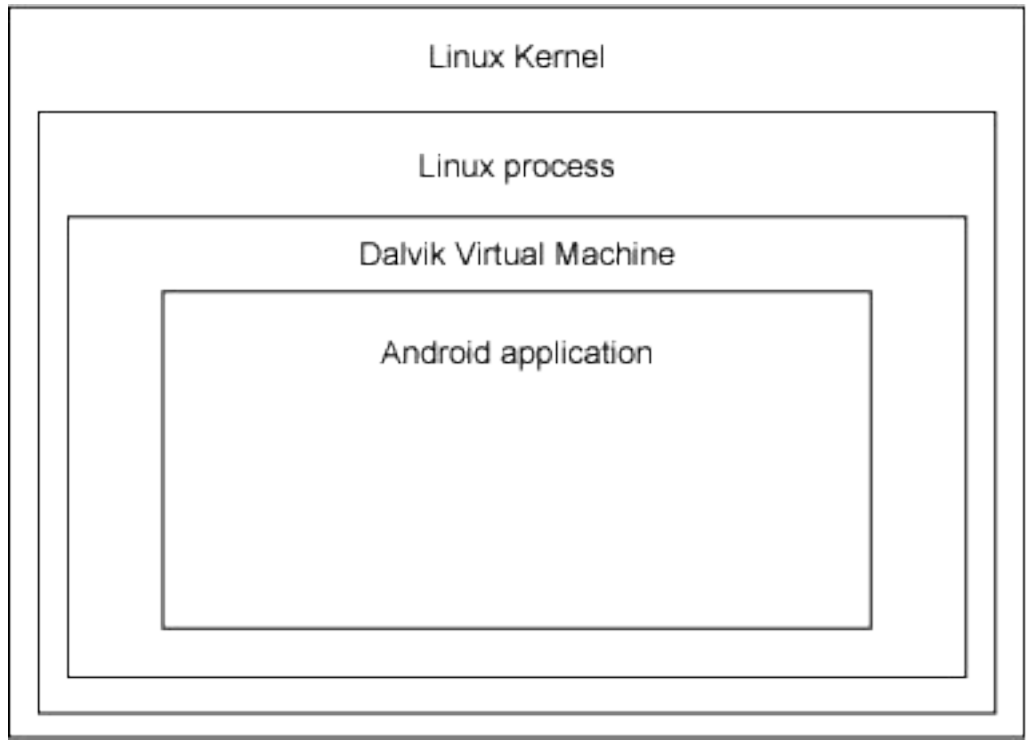
Android is an operating system based on Linux with a Java programming interface. It provides tools, e.g. a compiler, debugger and a device emulator as well as its own Java Virtual machine (Dalvik Virtual Machine - DVM). Android is created by the Open Handset Alliance which is lead by Google.

Android uses a special Java virtual machine (Dalvik) which is based on the Apache Harmony Java implementation. Dalvik uses special bytecode. Therefore you cannot run standard Java bytecode on Android. Android provides a tool "dx" which allows to convert Java Class files into "dex" (Dalvik Executable) files. Android applications are then packed into an .apk (Andro

Java language compiles to -> Dalvik byte-code which runs on -> Dalvik virtual machine -> inside the Android OS (Linux-based)

http://www.youtube.com/watch?v=ptjedOZEXPM
Google I/O 2008 - Dalvik Virtual Machine Internals




Linux Kernel
Linux process
Dalvik Virtual Machine
Android application

2.1 Hardware: Android devices

The Open Handset Alliance publishes a series of hardware specifications that all the devices for Android must comply. For Android 2.2, these are:

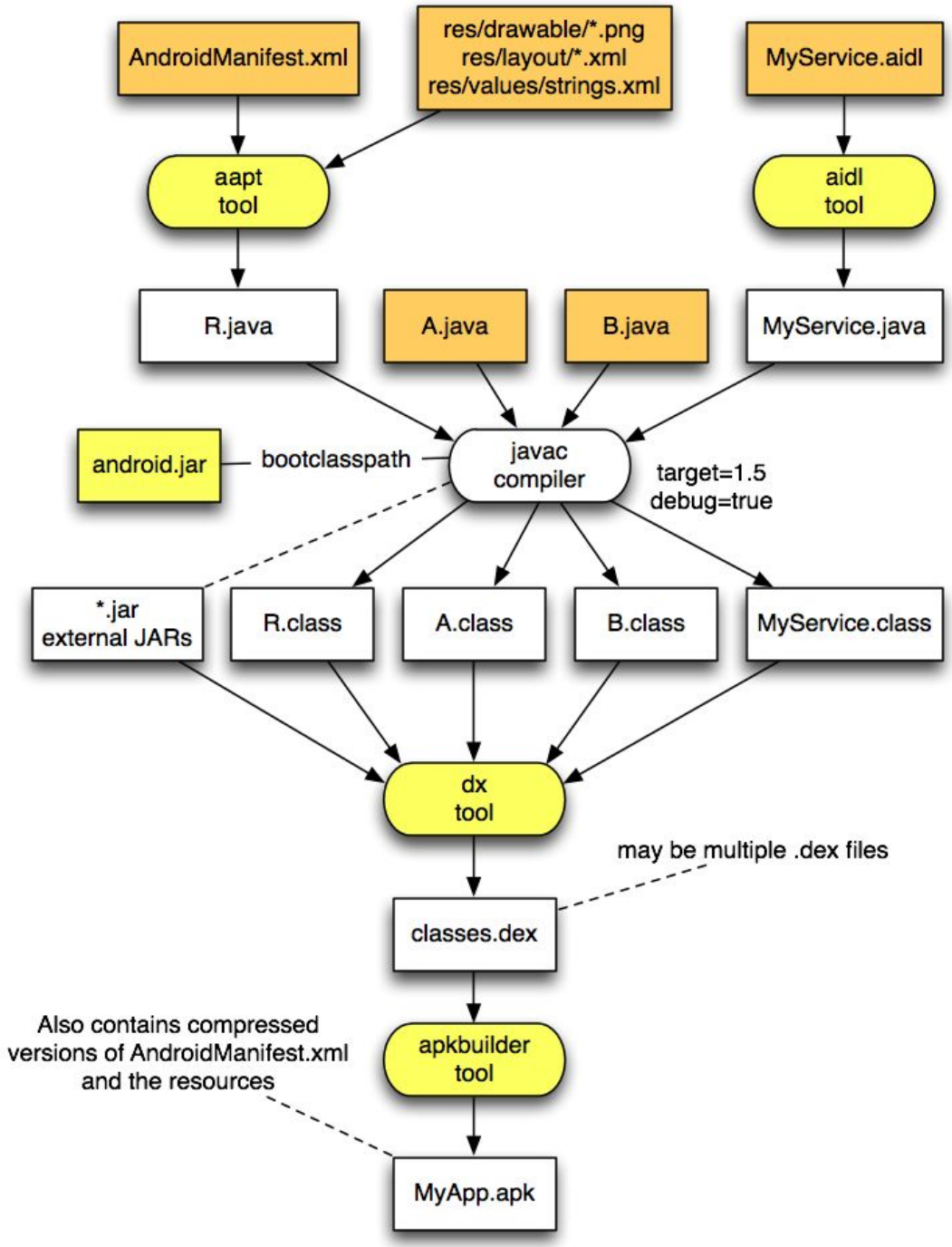
DISPLAY: Minimum QVGA (240x320) with portrait and landscape orientation
KEYBOARD: Must support soft keyboard
TOUCHSCREEN: Must have (not necessarily multitouch)
USB: USB-A port required for communication with host.
NAVIAGTION KEYS: Home, menu and back required
WIFI: Required,implementing one protocol that supports at least 200Kbit/sec
CAMERA: Required, at least one rear-facing with 2MP
ACCELEROMETER: 3-axis accelerometer required
COMPASS: 3-axis compass required
GPS: must include GPS receiver
TELEPHONY: Android 2.2 MAY be used on devices that do not include telephony hardware.
MEMORY AND STORAGE: At least 92Mb memory for kernel, 150Mb for non-volatile storage for user data
APPLICATION SHARED STORAGE: Must provide at least 2GB.

Android Compatibility Program
http://source.android.com/compatibility/overview.htm


11:03 PM
YouTube
Market
Browser
Voice
10:08
10
08
25°
H:26°
N:18°
London
Teils sonnig
Mi, 15. Sep.
htc
menu
Kamera
MOTOROLA
2:55 PM
Google
Messaging
Market
Phone
Contacts
Browser
Maps
Sprint
Google
Voicemail
Sprint Zone
Market
Email
25
Calendar

2.2 Android development process

Basic idea: writing code on host computer (PC/Mac) using text editor/command line or Eclipse (with ADT) (and now Processing).
Testing/debugging on Emulator
Upload to device
ADB allows to debug on device as well.



from
http://stuffthathappens.com/blo
droid-development-flow/

2.3 Application distribution: Android Market

Android Market is an online software store developed by Google for Android devices. An application program ("app") called "Market" is preinstalled on some Android devices and allows users to browse and download apps published by third-party developers, hosted on Android Market. The website, rather than the Market app itself, provides details of some of the available apps, in particular those that are termed "Featured", "Top Paid" and "Top Free".

v.android.com/market/

Percent of Apps on Platform by Normalized Category

Apple App Store 24 Hours After Launch vs. Android Market 24 Hours After Launch

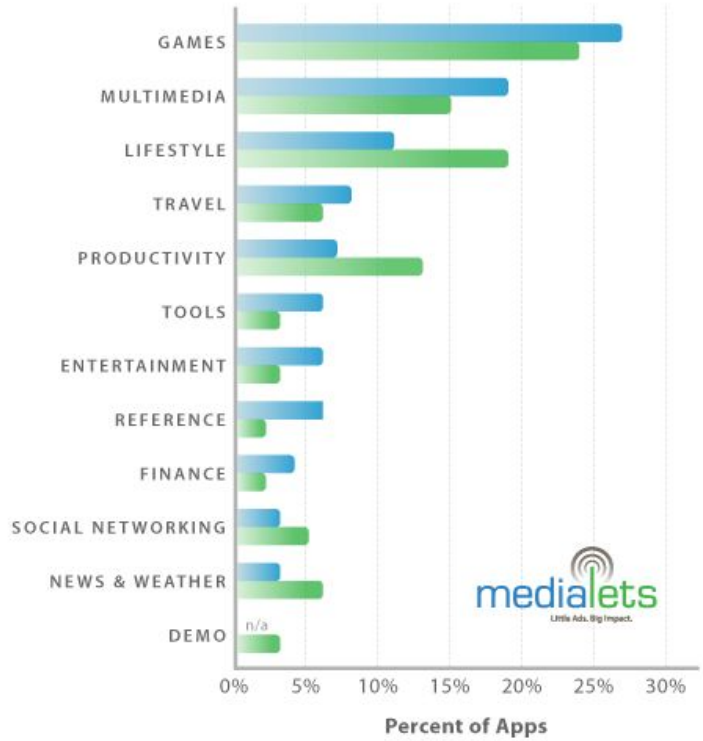
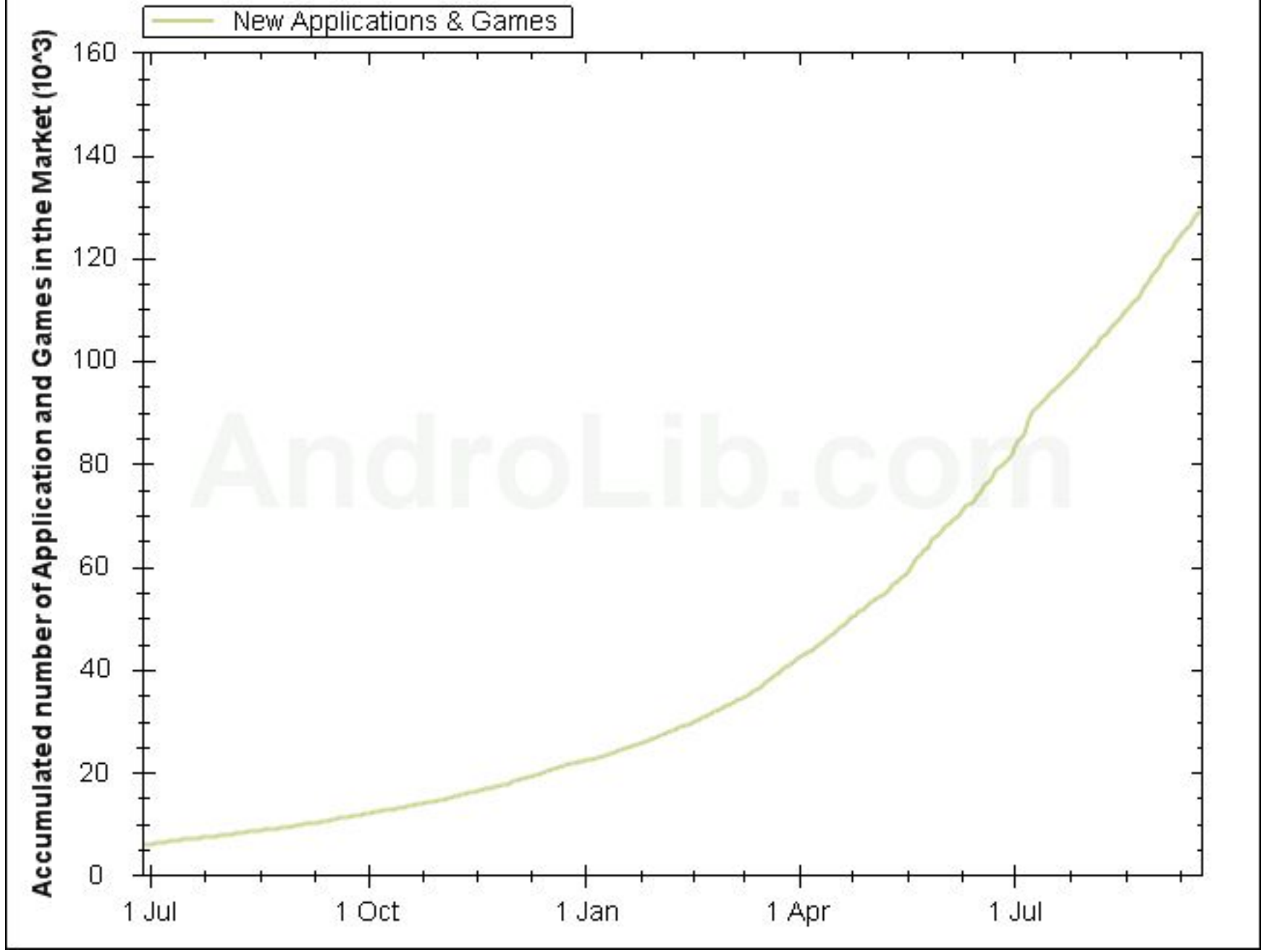
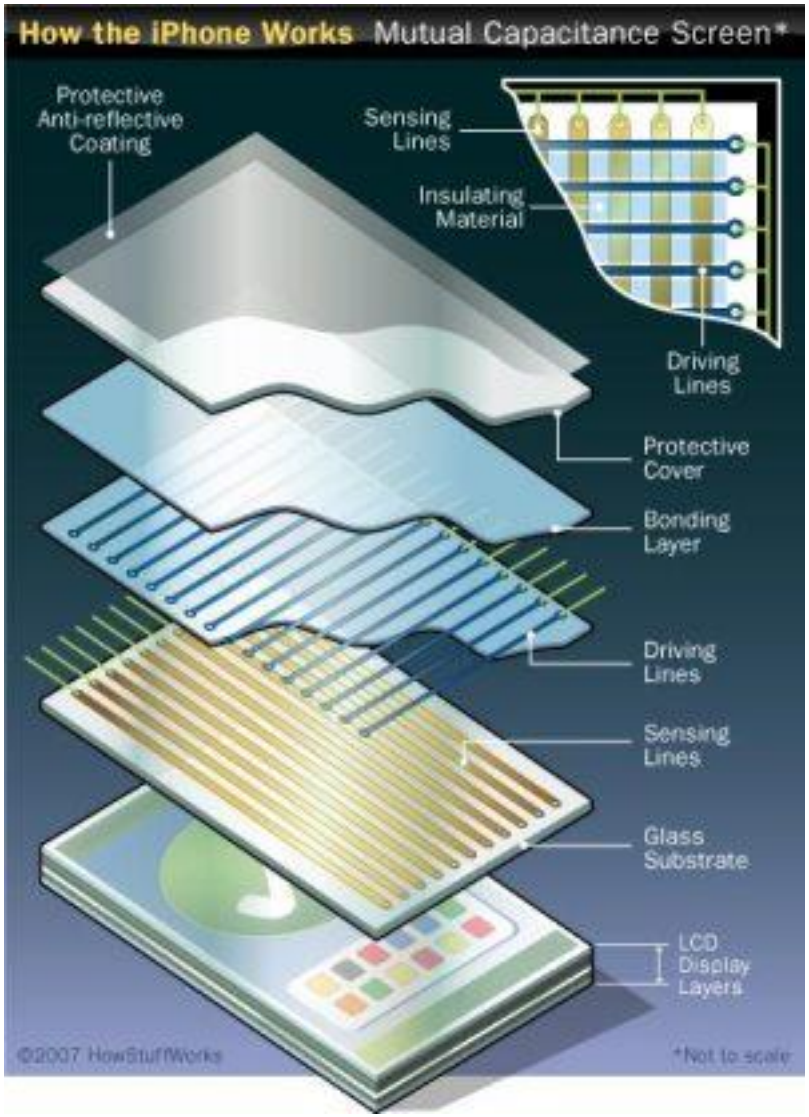


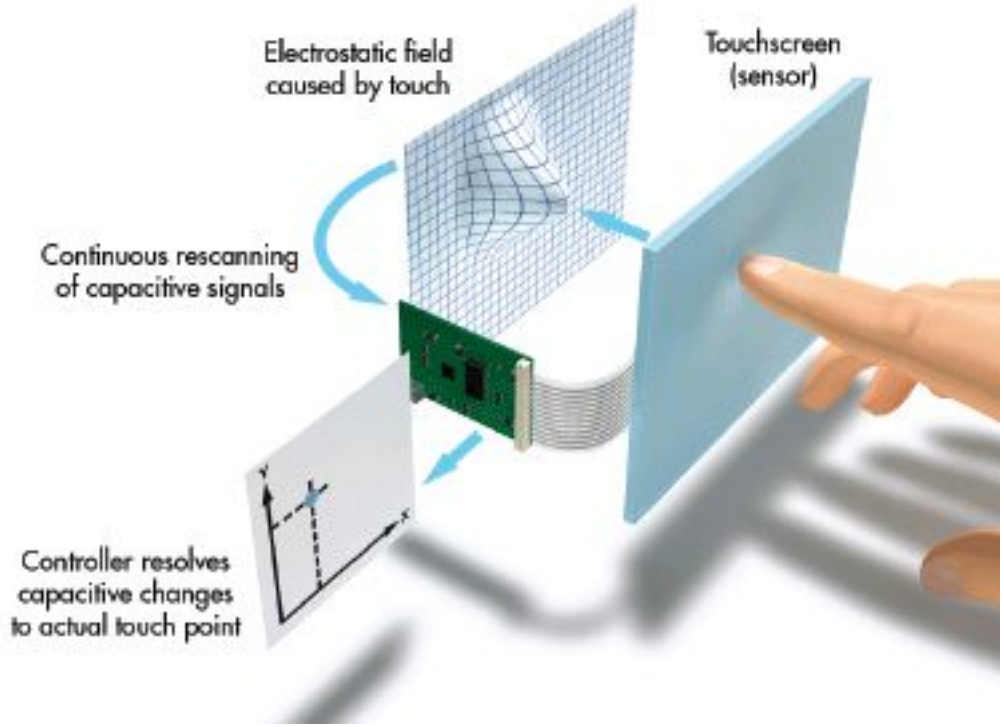
CHART Category	APP STORE Category	ANDROID MARKET Category
GAMES	Games	Games
MULTIMEDIA	Books, Music	Multimedia
LIFESTYLE	Lifestyle, Healthcare & Fitness, Sports, Photography	Lifestyle, Shopping
TRAVEL	Travel, Navigation	Travel
PRODUCTIVITY	Productivity, Business	Productivity
TOOLS	Utilities	Tools, SW Library
ENTERTAINMENT	Entertainment	Entertainment
REFERENCE	Reference, Education	Reference
FINANCE	Finance	Finance
SOCIAL NETWORKING	Social Networking	Social, Communication
NEWS & WEATHER	News,Weather	News & Weather
DEMO	N/A	Demo



http://www.androlib.com/appstats.aspx

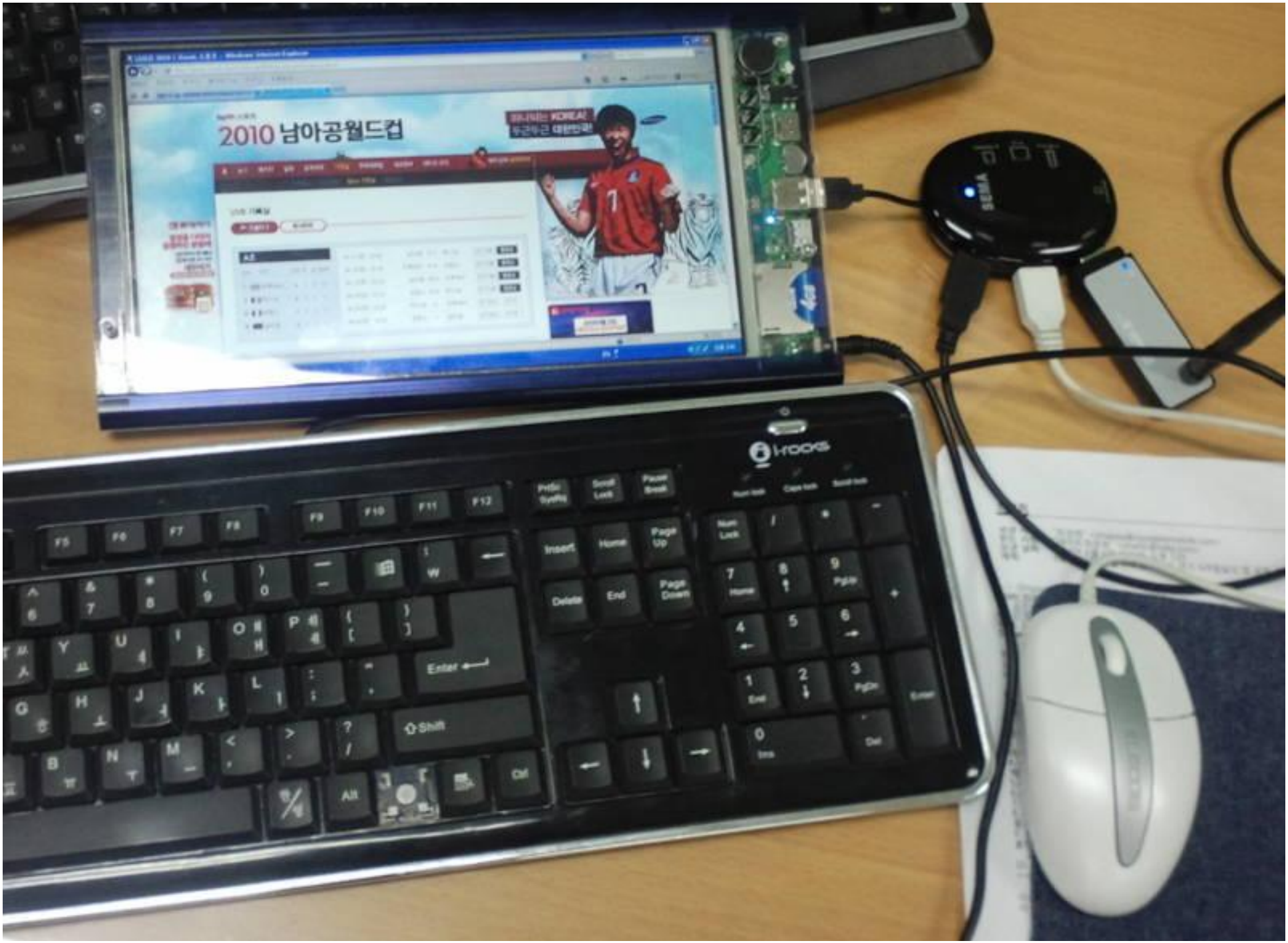
2.4 Android input: touchscreen, accelerometer, GPS, compass


How the iPhone Works Mutual Capacitance Screen*
Protective Anti-reflective Coating
Sensing Lines
Insulating Material
Driving Lines
Protective Cover
Bonding Layer
Driving Lines
Sensing Lines
Glass Substrate
LCD Display Layers
©2007 HowStuffWorks
*Not to scale


Electrostatic field caused by touch
Touchscreen (sensor)
Continuous rescanning of capacitive signals
Controller resolves capacitive changes to actual touch point


T··Mobile·
2:18 PM
FIRST AVE
300 ft
N
Total: 29 mi
First Ave
Main Street
Traffic Alert
2.5 mi
telenav
Main St
MENU

2.5 New form factors: Android tablets (Samsung Tab, Odroid-T)



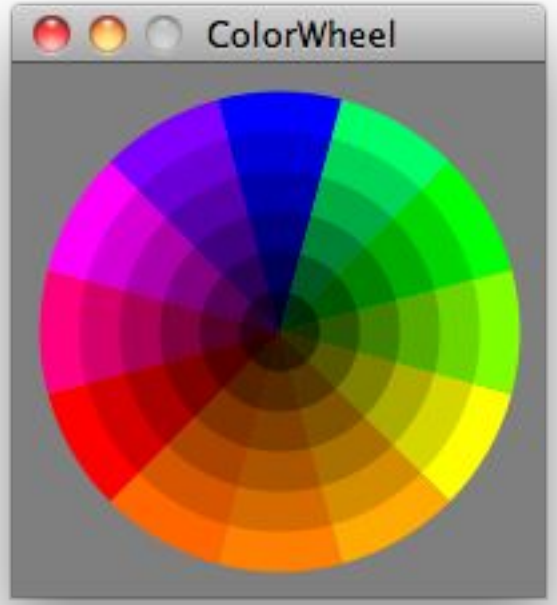
http://www.hardkernel.com/productsodroidt.php

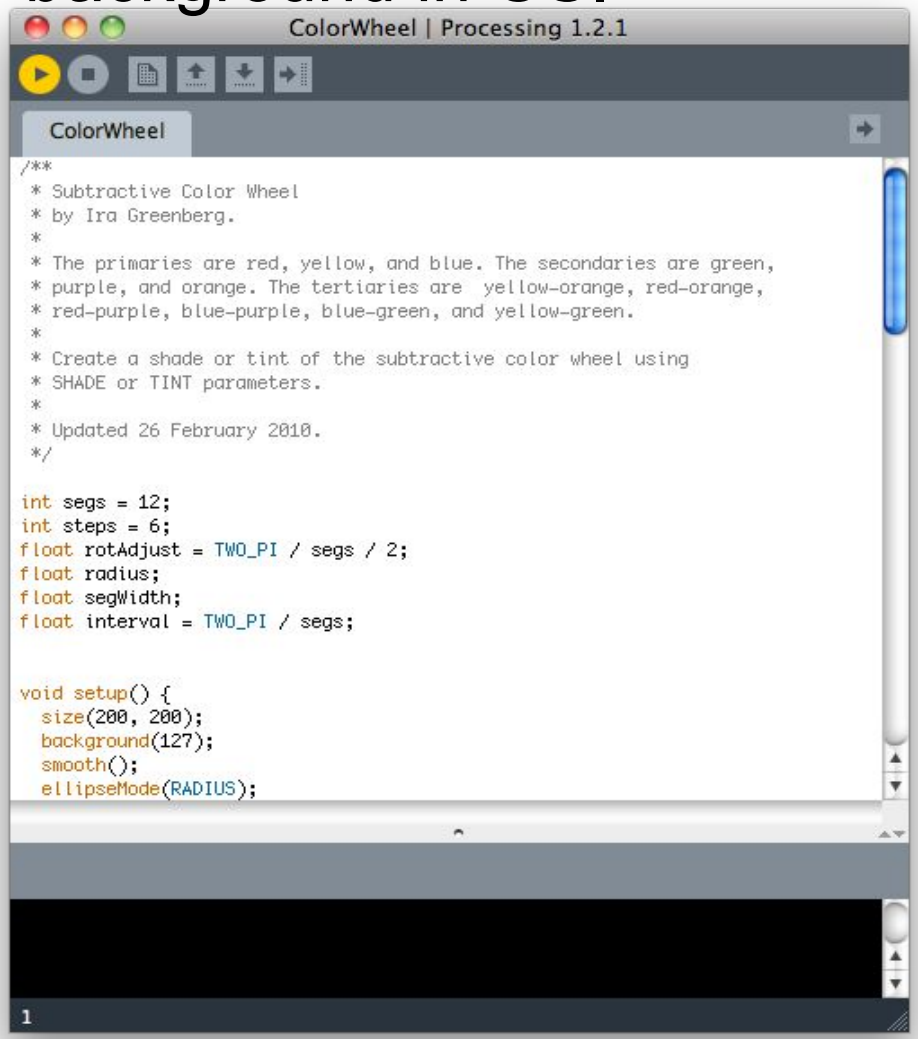
3. What is Processing?

Processing is an open source programming language and environment for people who want to create images, animations, and interactions. Initially developed to serve as a software sketchbook and to teach fundamentals of computer programming within a visual context, Processing also has evolved into a tool for generating finished professional work. Today, tens of thousands of students, artists, designers, researchers, and hobbyists who use Processing for learning, prototyping, and production.

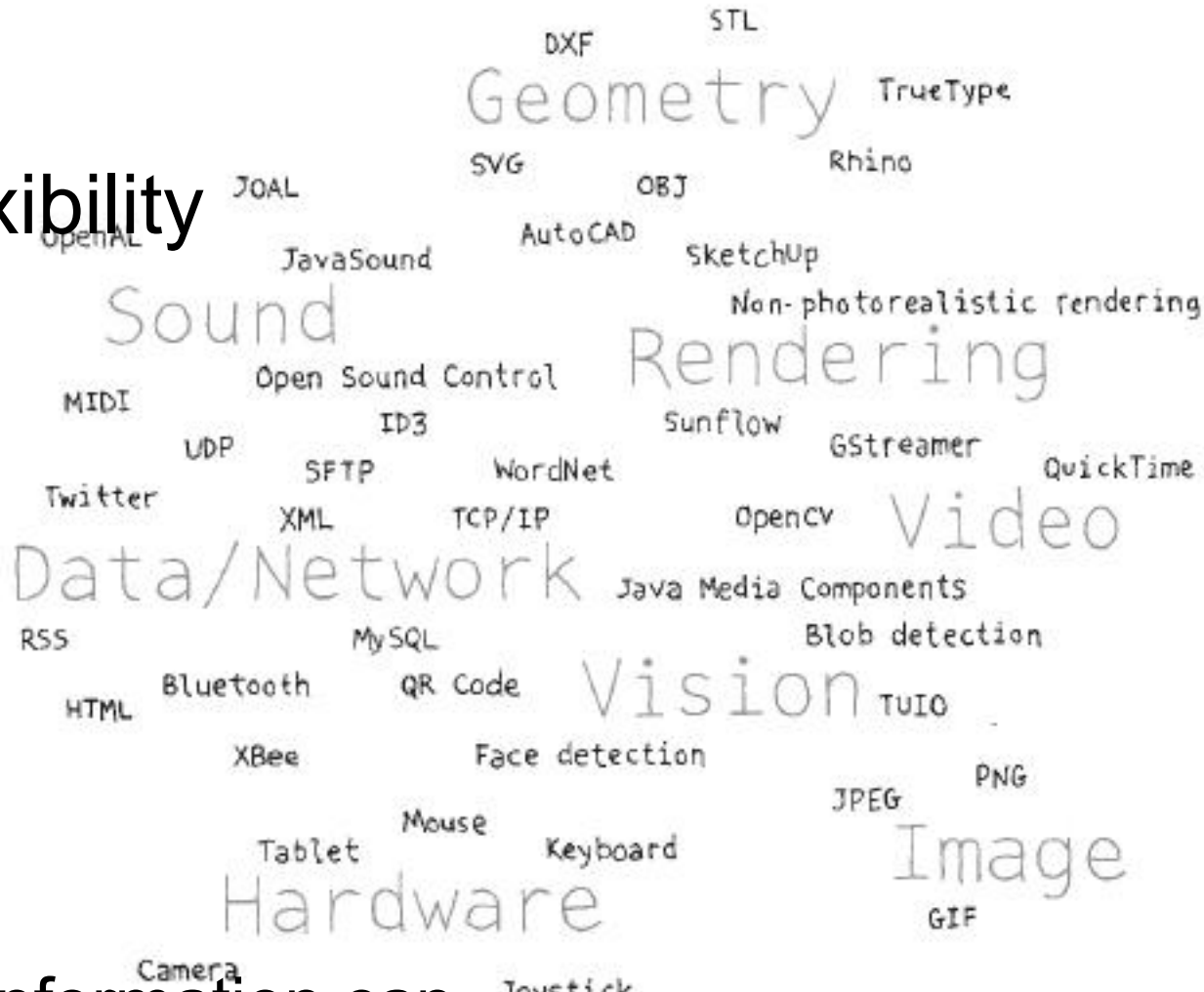
3.1 Main goal of Processing: simplicity

Processing was originally created with the purpose of making programming of graphics and interaction more accessible for people without technical background in CS.





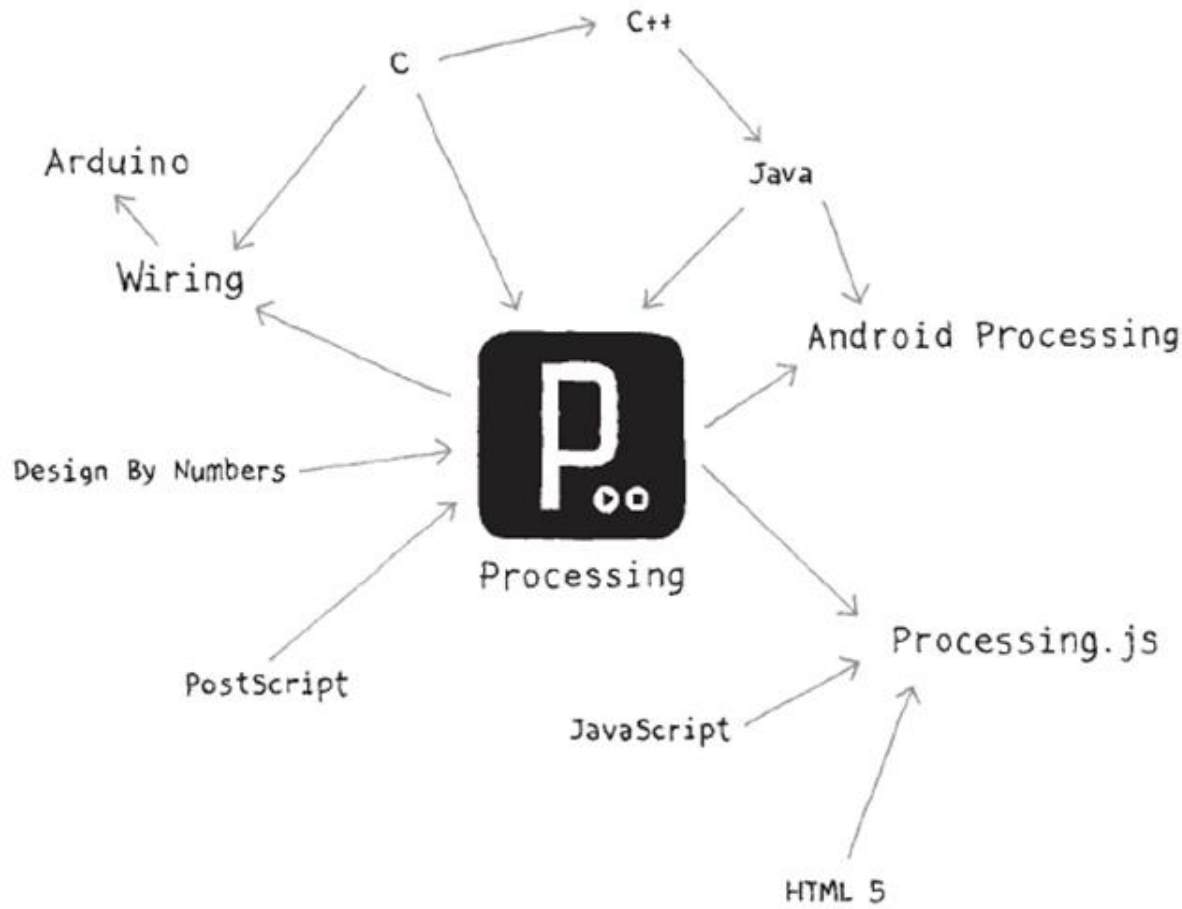
Flexibility



Many types of information can flow in and out of Processing.

(Casey Reas and Ben Fry. <Getting Started with Processing>. O'Really Media, 2010)

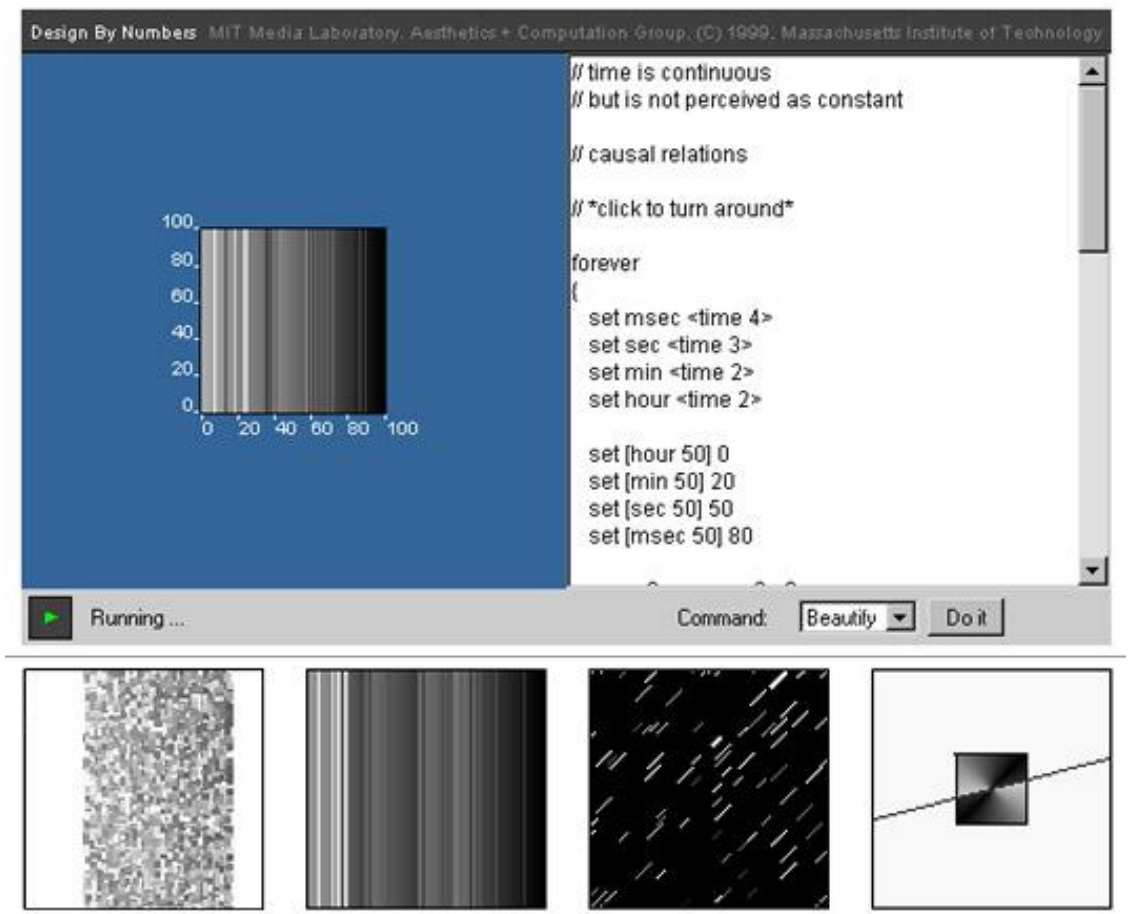
Family Tree



Processing has a large family of related languages and programming environments. (Casey Reas and Ben Fry. <Getting Started with Processing>. O'Really Media, 2010)

3.2 Education with Processing

Processing started as a project at the MIT Media Lab, and its direct ancestor was another language called Design By Numbers (DBN).
The goal of DBN was to teach programming to art and design students.
This was also one of the first applications of Processing (and still is).



http://openprocessing.org/collections/

3.3 Examples of work done with Processing

But Processing is also widely used in prototyping and final production of applications in many different areas: interactivity, generative graphics, physical computing, data visualization

Check a curated selection of work based on Processing at this website: http://processing.org/exhibition/

3.3.1
Interactivity

http://processing.org/exhibition/curated_page_11.html

Oasis: Yunsil Heo, Hyunw

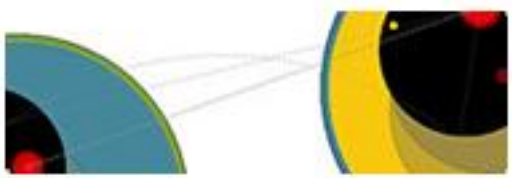
human / data

http://everyware.k
nts/09_oasis/

Oasis
by Yunsil Heo, Hyunwoo Bang

A playful space where people discover and explore virtual life.

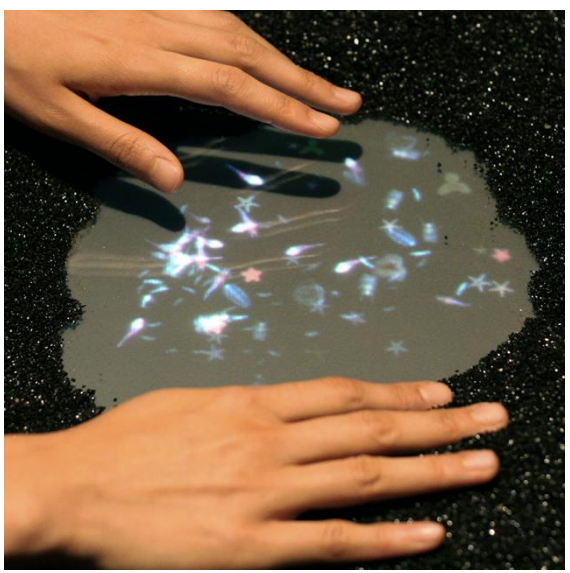
Links: everyware



People Mover
by Aeolab

A clockwork of colorful 'mobiles' assembles and unwinds according to data collected by three users of Nokia N95 smartphones over a period of several months.

Links: Aeolab



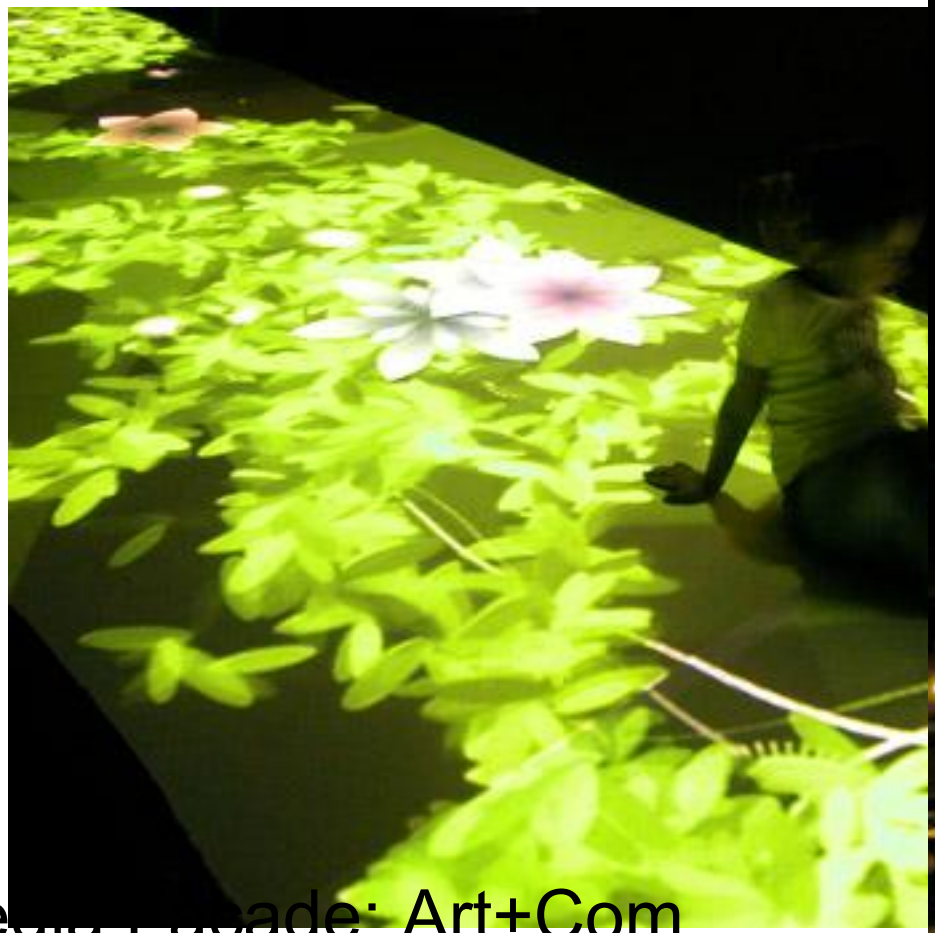
3.3.2 Data visualization

In the Air: by Victor Viña, Nerea Calvillo
http://www.intheair.es/

3.3.3 Physical Computing

Random International: Light Roller
http://www.random-international.com/temporary-graffiti-light-roller-2/

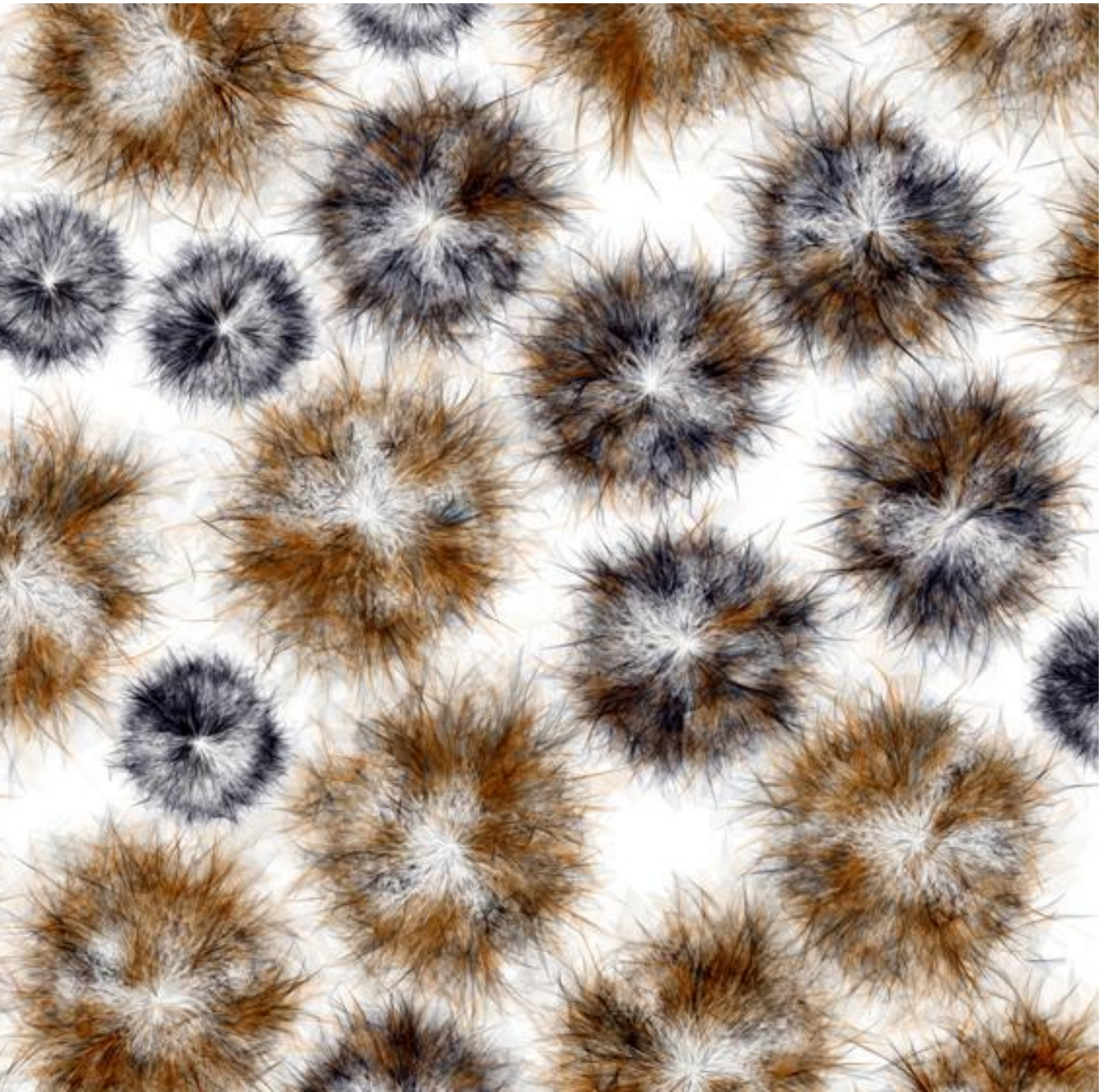
3.3.4 Real-time graphics and video



Vattnfall Media Facade: Art+Com

http://www.artcom.de/index.php?option=com_acprojects&page=5&id=30&Itemid=144&details=&imageReq

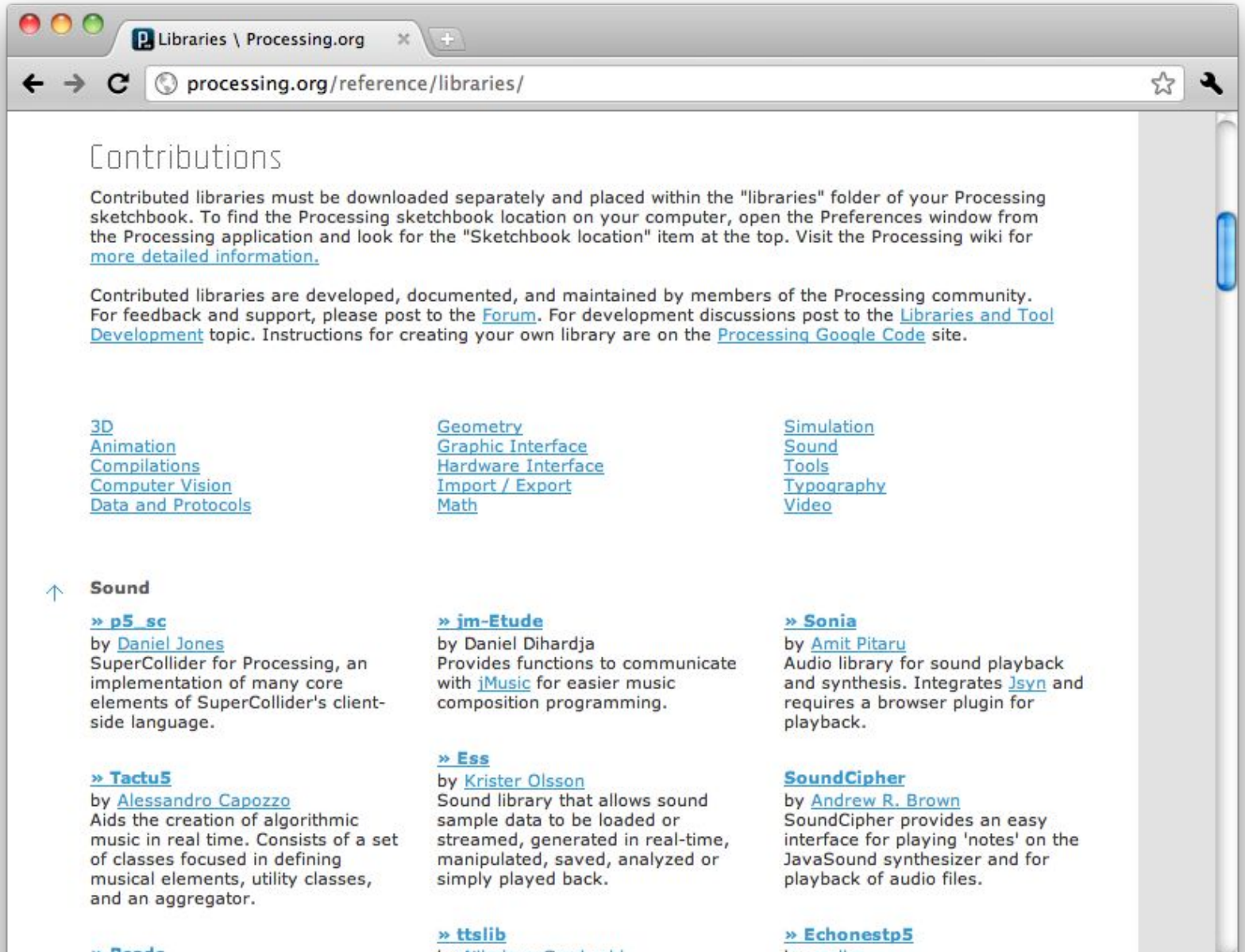
3.3.5 Generative graphics



Process 6: Casey Rea
http://reas.com/catego
tion=works

3.6 Processing is extensible with libraries

Processing allows to extend the core functionality by means of external libraries. There are more than 100 libraries contributed by the community.



3.7 Processing is Java-based!

From a more technical perspective, Processing is two things:

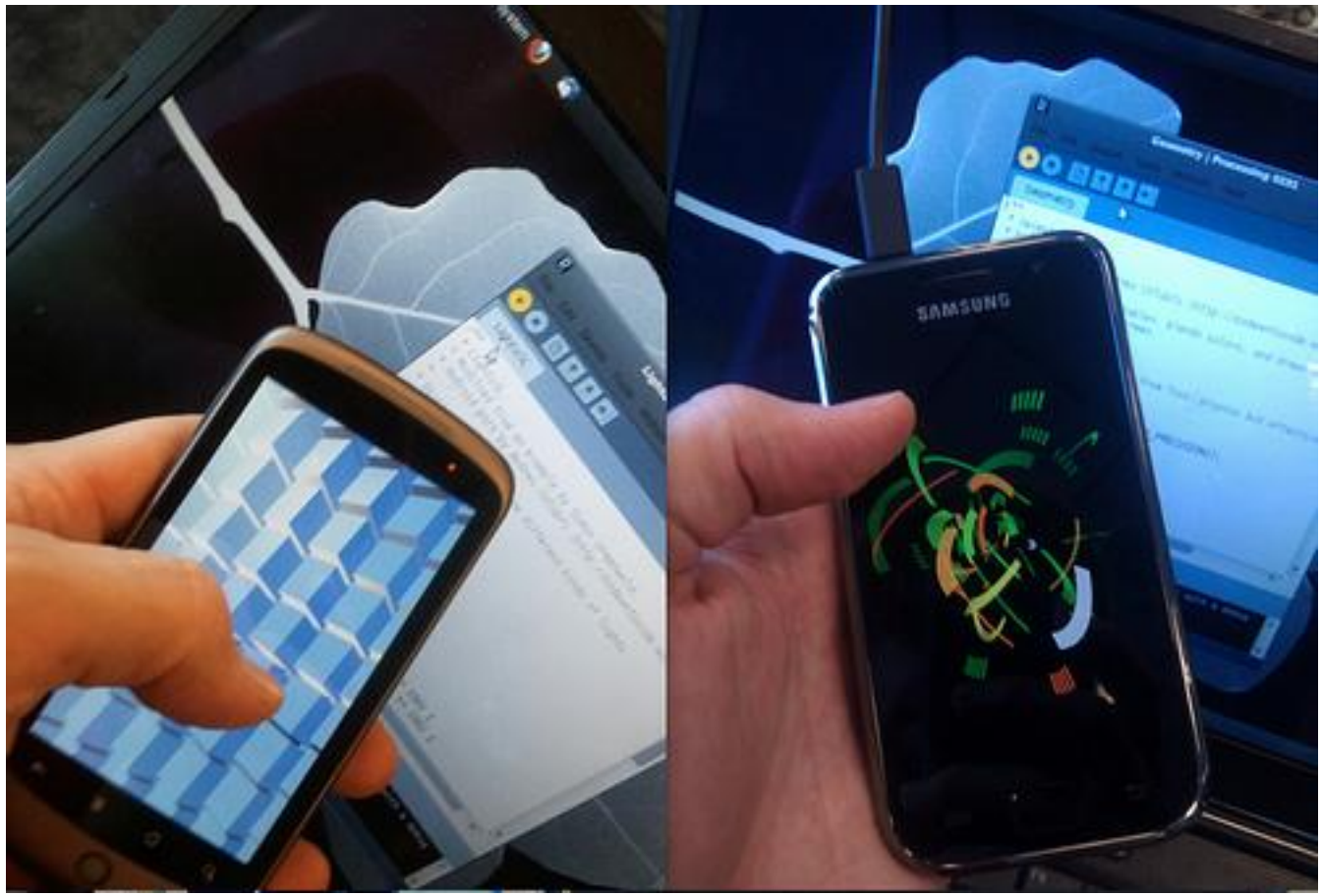
1) A minimal Development Environment (Called PDE), that favors ease of use over functionality

2) A programming language, and as such is basically a dialect built on top of Java to make graphics programming less cumbersome thanks to a simple API.

```
void setup()
{
  size(200, 200);
  img = createImage(120, 120, ARGB);
  for(int i=0; i < img.pixels.length; i++) {
    img.pixels[i] = color(0, 90, 102, i%img.width * 2);
  }
}

void draw()
{
  background(204);
  image(img, 33, 33);
```

A consequence of this is that we can use Eclipse as the IDE for more advanced development and....

... now we have the port of Processing for Android!



4. Basic concepts in Android applications

An Android application consists out of the following parts:

* Activity - A screen in the Android application
* Services - Background activities without UI
* Content Provider - provides data to applications, Android contains a SQLLite DB which can serve as data provider
* Broadcast Receiver - receives system messages, can be used to react to changed conditions in the system

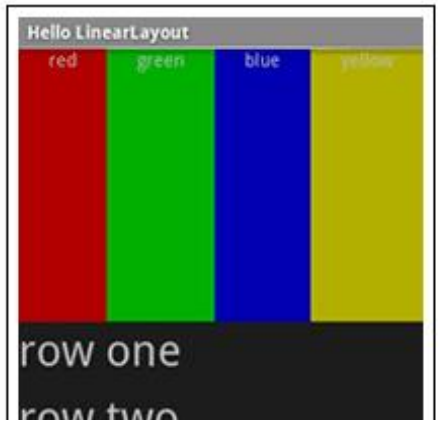
Intends allow the application to request and / or provide services . For example the application call ask via an intent for a contact application. Application register themself via an IntentFilter. Intends are a powerful concept as they allow to create loosely coupled applications.

https://sites.google.com/site/androidappcourse/

view: layout

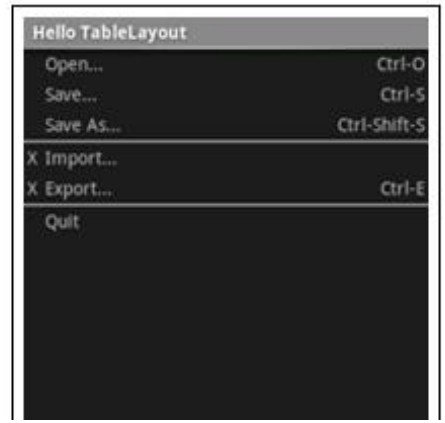
http://developer.android.com/resources/tutorials/views/index.html

Linear Layout

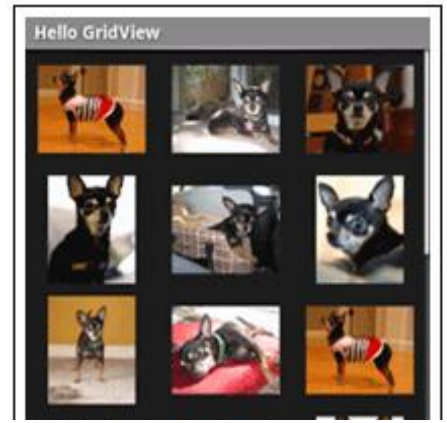


Relative Layout

Table Layout



Grid View



Auto Complete

Gallery

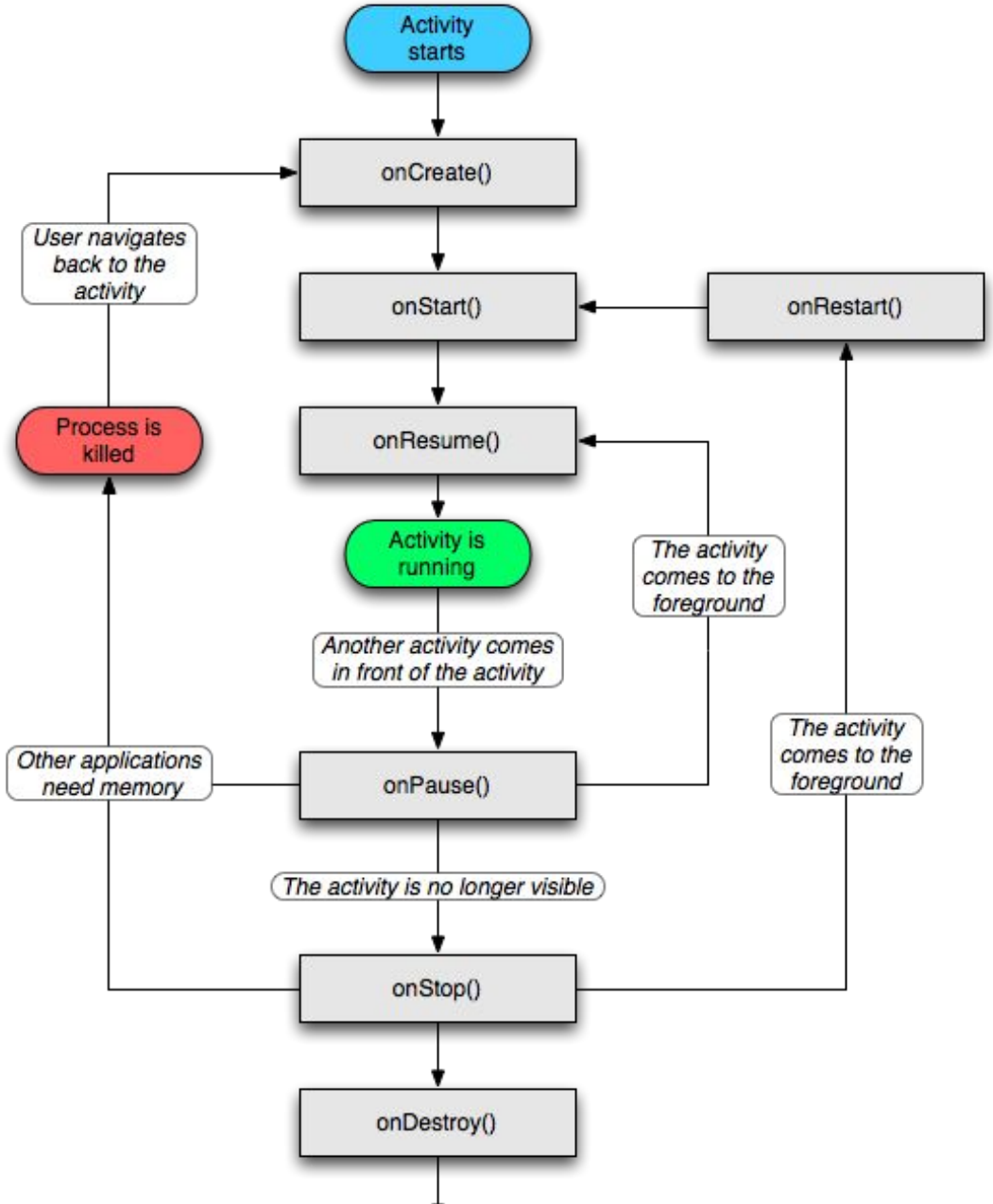
Google Map View

Web View

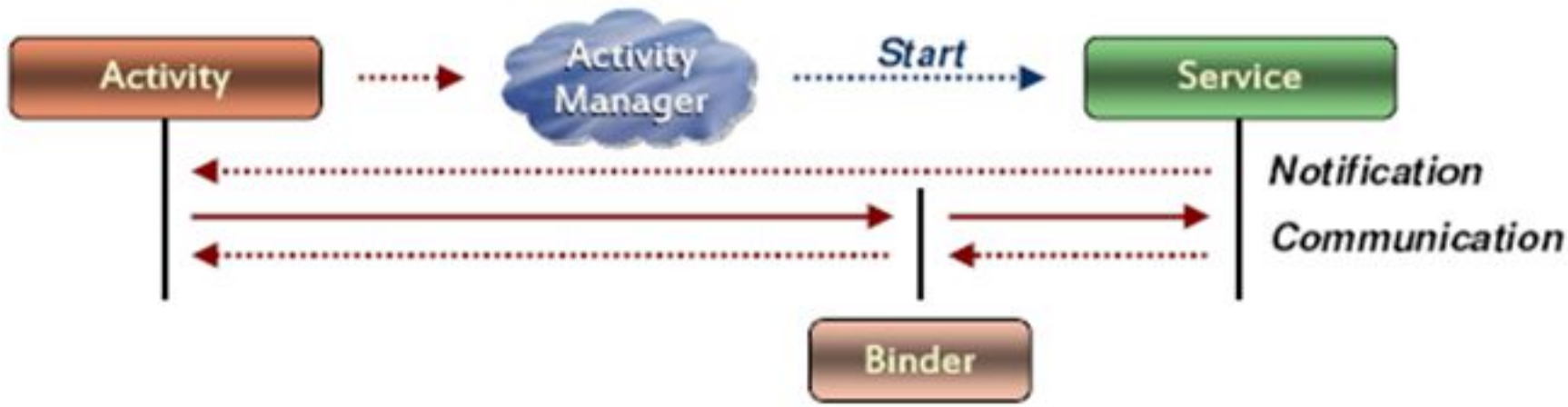
Activities An *activity* presents a visual user interface for one focused endeavor the user can undertake. For example, an activity might present a list of menu items users can choose from or it might display photographs along with their captions. A text messaging application might have one activity that shows a list of contacts to send messages to, a second activity to write the message to the chosen contact, and other activities to review old messages or change settings. Though they work together to form a cohesive user interface, each activity is independent of the others.An application might consist of just one activity or, like the text messaging application just mentioned, it may contain several.
Each activity is given a default window to draw in. Typically, the window fills the screen, but it might be smaller than the screen and float on top of other windows.

The visual content of the window is provided by a hierarchy of views — objects derived from the base View class. Each view controls a particular rectangular space within the window.

Activity lifecycle (image from google
website: http://developer.android.com/reference/android/app/Activity.html



Services **A *service* doesn't have a visual user interface, but rather runs in the background for an indefinite period of time. For example, a service might play background music as the user attends to other matters, or it might fetch data over the network or calculate something and provide the result to activities that need it. Each service extends the Service base class.**



Nice images here about Activities, intents, services
http://blog.gbinghan.com/2010/08/android-basics-quick-start.html

Broadcast receiversA *broadcast receiver* is a component that does nothing but receive and react to broadcast announcements. Many broadcasts originate in system code — for example, announcements that the timezone has changed, that the battery is low, that a picture has been taken, or that the user changed a language preference. Applications can also initiate broadcasts — for example, to let other applications know that some data has been downloaded to the device and is available for them to use.

Content providersA *content provid* specific set of the application's data other applications. The data can be file system, in an SQLite database, other manner that makes sense

Activating components: intents

Content providers are activated when they're targeted by a request from a ContentResolver. The other three components — activities, services, and broadcast receivers — are activated by asynchronous messages called *intents*. An intent is an Intent object that holds the content of the message. For activities and services, it names the action being requested and specifies the URI of the data t

GUI design

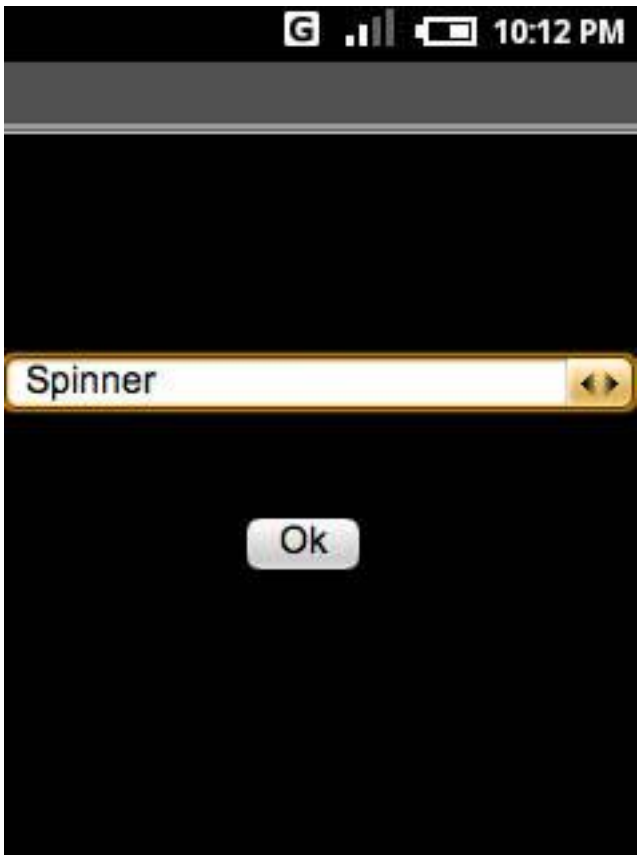
s an XML based markup language to define user interface layouts,
's similar to UIML. XML is used to create flexible interfaces which
modified and wired up in the Java code. Mozilla's XUL, Windows
Foundation XAML, and Macromedia Flex's MXML (and to some
SVG) all operate similarly.

n the XML tree corresponds to a screen object or layout container
ear in the rendered interface.

, the XML below defines the interface seen here.

```
<?xml version="1.0" encoding="utf-8"?> <AbsoluteLayout
android:id="@+id/myAbsoluteLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="@drawable/black"
xmlns:android="http://schemas.android.com/apk/res/android">
<Spinner android:id="@+id/mySpinner"
android:layout_width="fill_parent"
android:layout_height="wrap_content" android:layout_x="0px"
android:layout_y="82px" > </Spinner> <Button
d="@+id/myButton" android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:background="@drawable/darkgray" android:text="Ok"
android:layout_x="80px" android:layout_y="122px" > </Button>

</AbsoluteLayout>
```

from http://vis.berkeley.edu/course
dex.php/Getting_Started_with_And



4.1 The manifest file

An Android application is described the file "AndroidManifest.xml". This files contains all activities application and the required permissions for the application. For example if the application requires network access it must be specified here. "AndroidManifest.xml" can be thought as the deployment descriptor for an Android application.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="org.openwebvancouver.schedule" android:versionCode="3" android:versionName="1.0.2">
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <application android:icon="@drawable/icon" android:label="@string/app_name"
        android:debuggable="false">
        <activity android:name=".DroidGap"
                  android:label="@string/app_name" android:configChanges="orientation|keyboardHidden">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="3" />
</manifest>
```

4.4 Android Software Development Kit (SDK)

The Android SDK includes a comprehensive set of development tools.[77] These include a debugger, libraries, a handset emulator (based on QEMU), documentation, sample code, and tutorials. Currently supported development platforms include x86-architecture computers running Linux (any modern desktop Linux distribution), Mac OS X 10.4.8 or later, Windows XP or Vista. Requirements also include Java Development Kit, Apache Ant, and Python 2.2 or later. The officially supported integrated development environment (IDE) is Eclipse (3.2 or later) using the Android Development Tools (ADT) Plugin, though developers may use any text editor to edit Java and XML files then use command line tools to create, build and debug Android applications as well as control attached Android devices (e.g., triggering a reboot, installing software package(s) remotely).

ADB = Android Debug Bridge

http://developer.android.com/guide/developing/tools/adb.html

Android Debug Bridge (adb) is a versatile tool lets you manage the state of an emulator instance or Android-powered device. It is a client-server program that includes three components:

- A client, which runs on your development machine. You can invoke a client from a shell by issuing an adb command. Other Android tools such as the ADT plugin and DDMS also create adb clients.
- A server, which runs as a background process on your development machine. The server manages communication between the client and the adb daemon running on an emulator or device.
- A daemon, which runs as a background process on each emulator or device instance.

Graphical interfaces for ADB:
Linux: ADBUix (http://kodieg.com/?x=projects/adbuix)
Windows: WinADB (http://solo-dev.deviantart.com/art/WinADB-177848431)

4.5 Android Development with Eclipse

Eclipse can be used as the IDE for Android development. To do so, you need to install the SDK and then the: ADT (Android Development Tools)

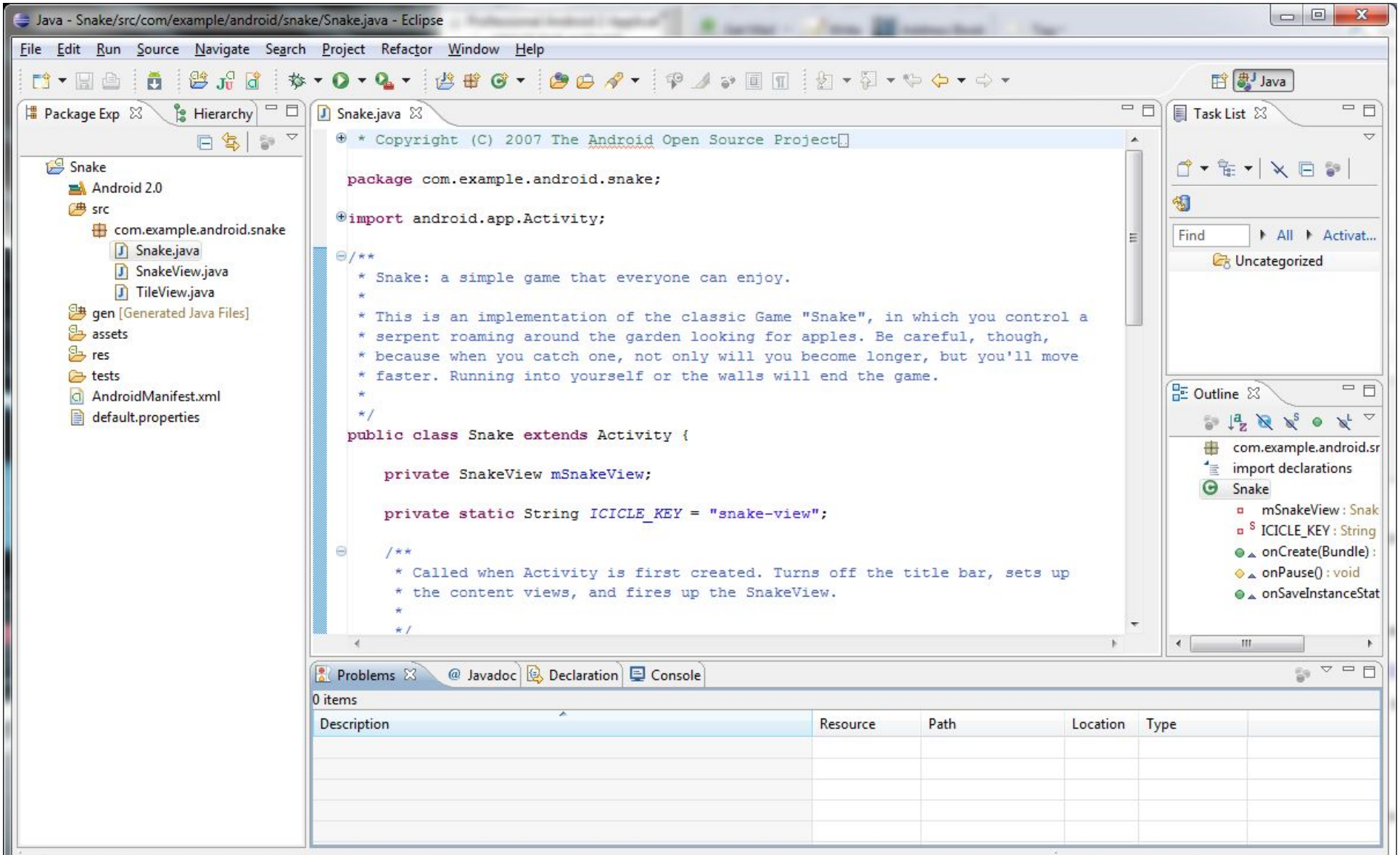
Android Development Tools (ADT) is a plugin for the Eclipse IDE that is designed to give you a powerful, integrated environment in which to build Android applications.

ADT extends the capabilities of Eclipse to let you quickly set up new Android projects, create an application UI, add components based on the Android Framework API, debug your applications using the A
export signed (or unsigned) APKs in order to distribut

Excellent
reference: http://www.vogella.de/articles/Android/article.html

4.6 Disadvantages using Eclipse

For single screen interactive or data sensing applications, Eclipse and the full SDK of Android might result unnecessarily complex, and very challenging for beginners. Also, the drawing API, both in 2D and 3D (OpenGL ES), could be time consuming to learn and use.



4.7 Solution: Processing for Android!

from the Android section in the Processing wiki: "The primary goal of this project is to make it foolishly easy to create Android apps using the Processing API. Once you have Processing on your machine (and the Android developer tools), you can simply write a line of code, hit 'Run' (or Ctrl-R), and have your sketch show up in the emulator as a working Android app. Select 'Present' (or use Ctrl-Shift-R) to have it run on an Android device that you have plugged into your machine. That's good stuff!"

5. How to install the Android SDK

Download the Android SDK from the Android homepage under Android SDK download . The download contains a zip file which you can extract to any place in your file system, e.g. I placed it under "c:\android-sdk-windows" .

The detailed instructions
are http://developer.android.com/sdk/installing.html

5.1 Installation on Windows

0) Make sure that you install the JDK first...

1) Download latest package from http://developer.android.com/sdk/index.html

2) Unzip package at the place of your preference...
C:\Users\andres\Coding\android-sdk-windows

3) Add the environmental variables PATH and ANDROID_SDK
On Windows, right-click on My Computer, and select Properties. Under the Advanced tab, hit the Environment Variables button, and in the dialog that comes up, double-click on Path (under System Variables). Add the full path to the tools/ directory to the path, and add the new env variable: `ANDROID_SDK=`/Users/andres/Coding/Android/android-sdk-mac_x86

3.5) Also, if the path to the bin folder of the JDK is not in the PATH, add it.

4) The last step in setting up your SDK is using a tool included the SDK starter package — the *Android SDK and AVD Manager* . You can start it by double-clicking on android.exe in the tools folder

USB driver installation

Note that this driver provides support only for the following (or similar) devices:

- T-Mobile G1* / ADP1
- T-Mobile myTouch 3G* / Google Ion
- Verizon Droid*
- Nexus One

After downloading the usb driver in the previous step (it gets copied to C:\Users\andres\Coding\android-sdk-windows\usb_driver) you have to install it followir these:

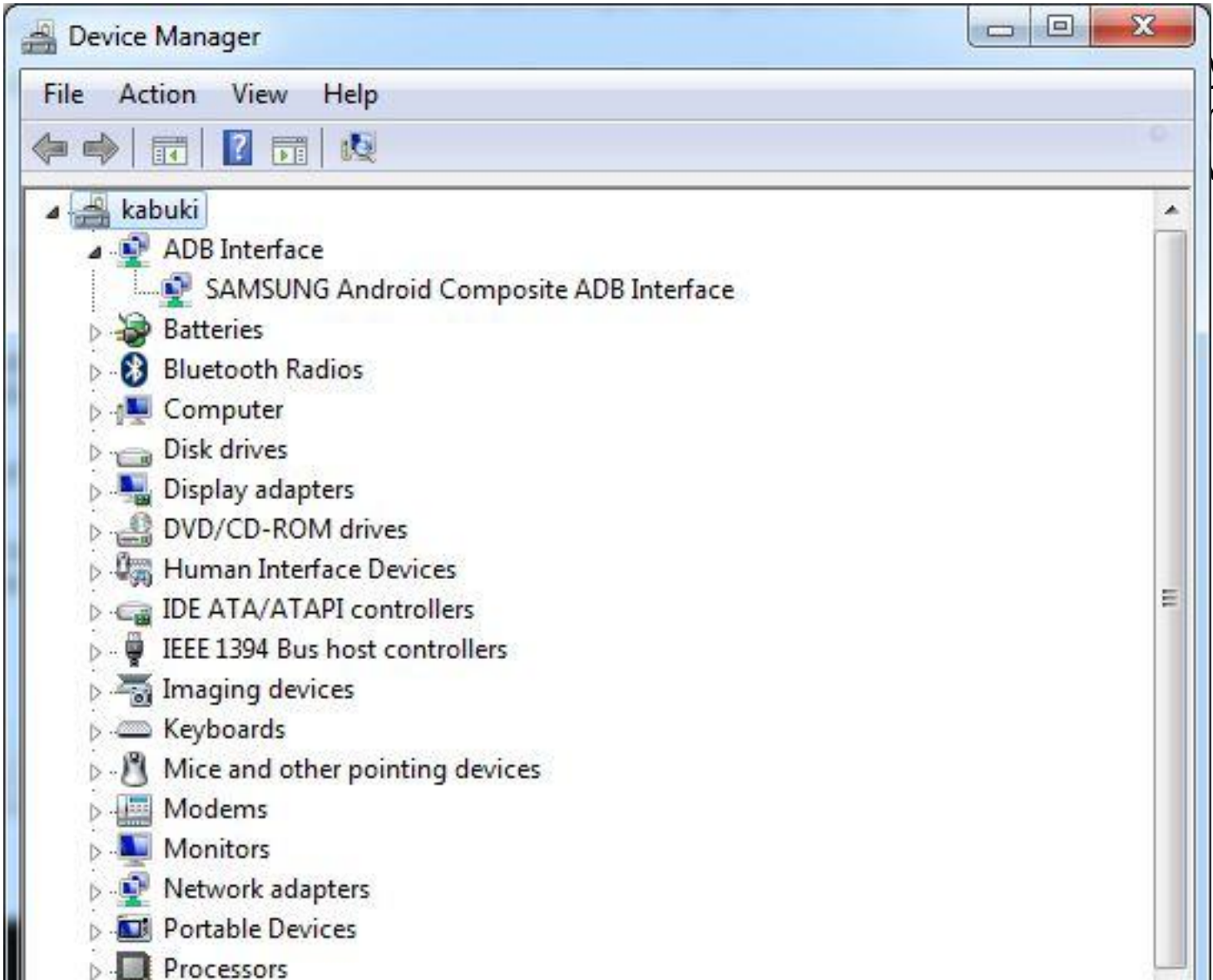
Windows Vista: Perform a fresh installation
To install the Android USB driver on Windows Vista for the first time:

1. Connect your Android-powered device to your computer's USB port. Windows will detect the device and launch the Found New Hardware wizard.
2. Select "Locate and install driver software."
3. Select "Don't search online."
4. Select "I don't have the disk. Show me other options."
5. Select "Browse my computer for driver software."
6. Click "Browse..." and locate the folder where you copied the installation package. A long as you specified the exact location of the installation package, you may leave

For Samsung Galaxy S:

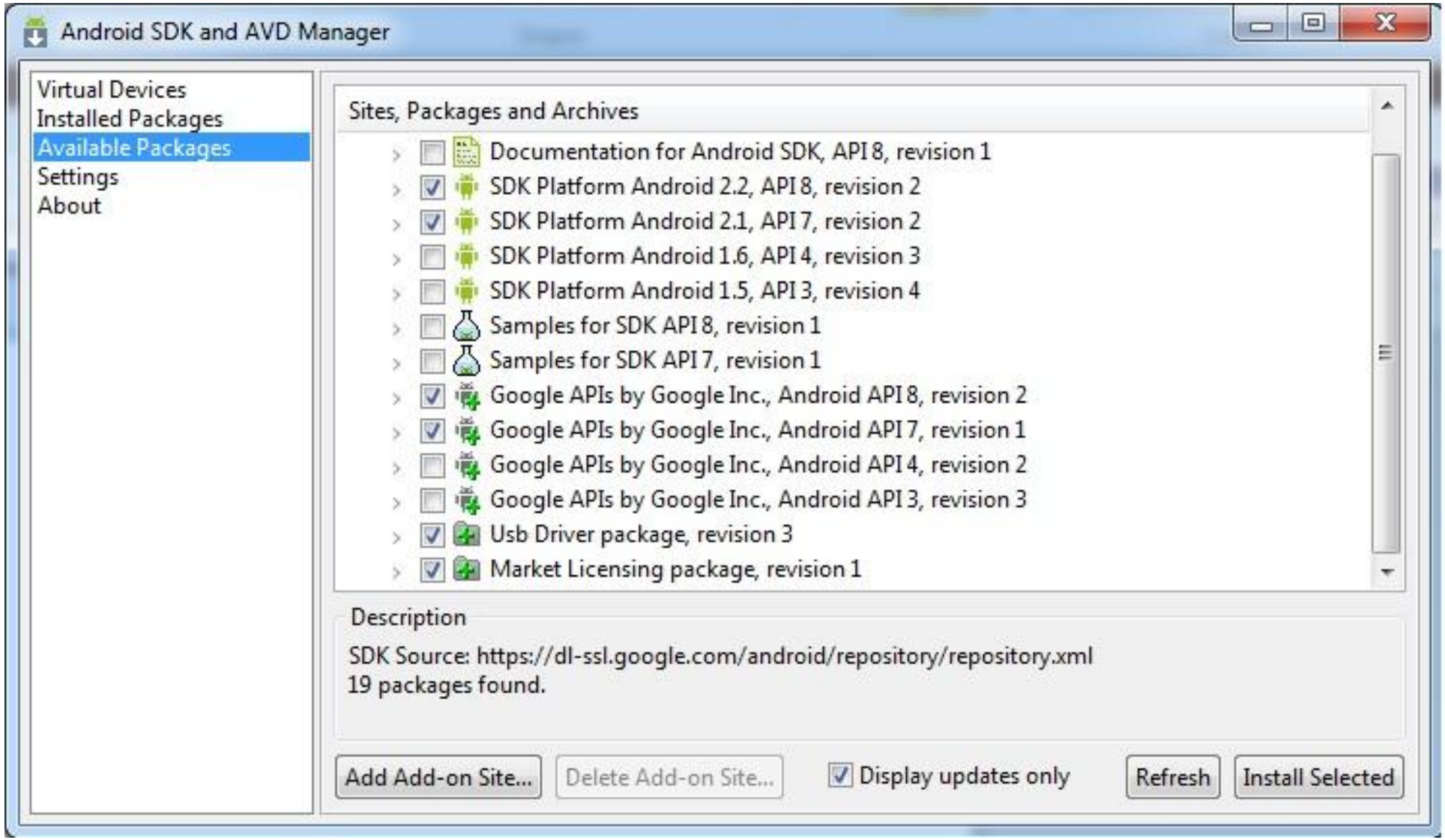
http://forum.xda-developers.com/showthread.php?t=728929

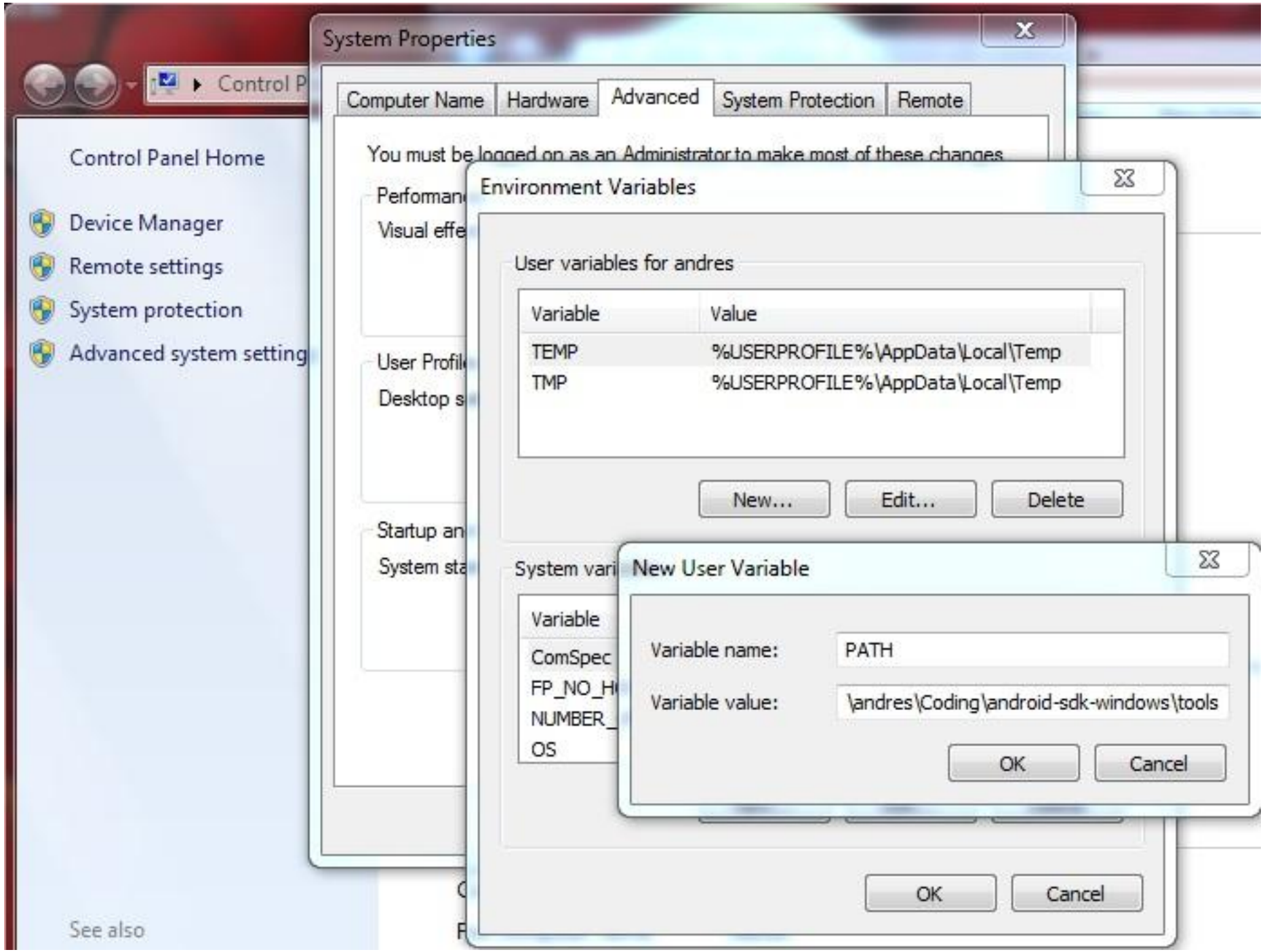
http://www.samsung.com/us/support/downloads/global



y AT-T and the model
nwload page:
ATT

These are the minimal components of the SDK required to use Processing for Android (SDK and APIs 7), and in windows the Usb driver package (very important!)



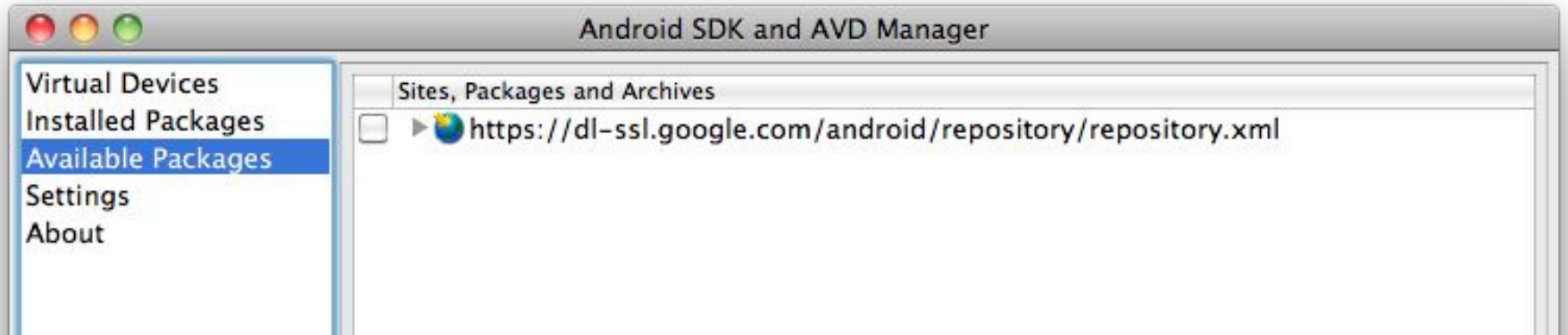

Control P
Control Panel Home
Device Manager
Remote settings
System protection
Advanced system setting
See also
System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
You must be logged on as an Administrator to make most of these changes
Environment Variables
User variables for andres
Variable
Value
TEMP
%USERPROFILE%\AppData\Local\Temp
TMP
%USERPROFILE%\AppData\Local\Temp
New...
Edit...
Delete
System vari
Variable
ComSpec
FP_NO_H
NUMBER_
OS
OK
Cancel
New User Variable
Variable name:
PATH
Variable value:
\andres\Coding\android-sdk-windows\tools
OK
Cancel

5.2 Installation on Mac OSX

1) Download latest package from http://developer.android.com/sdk/index.html

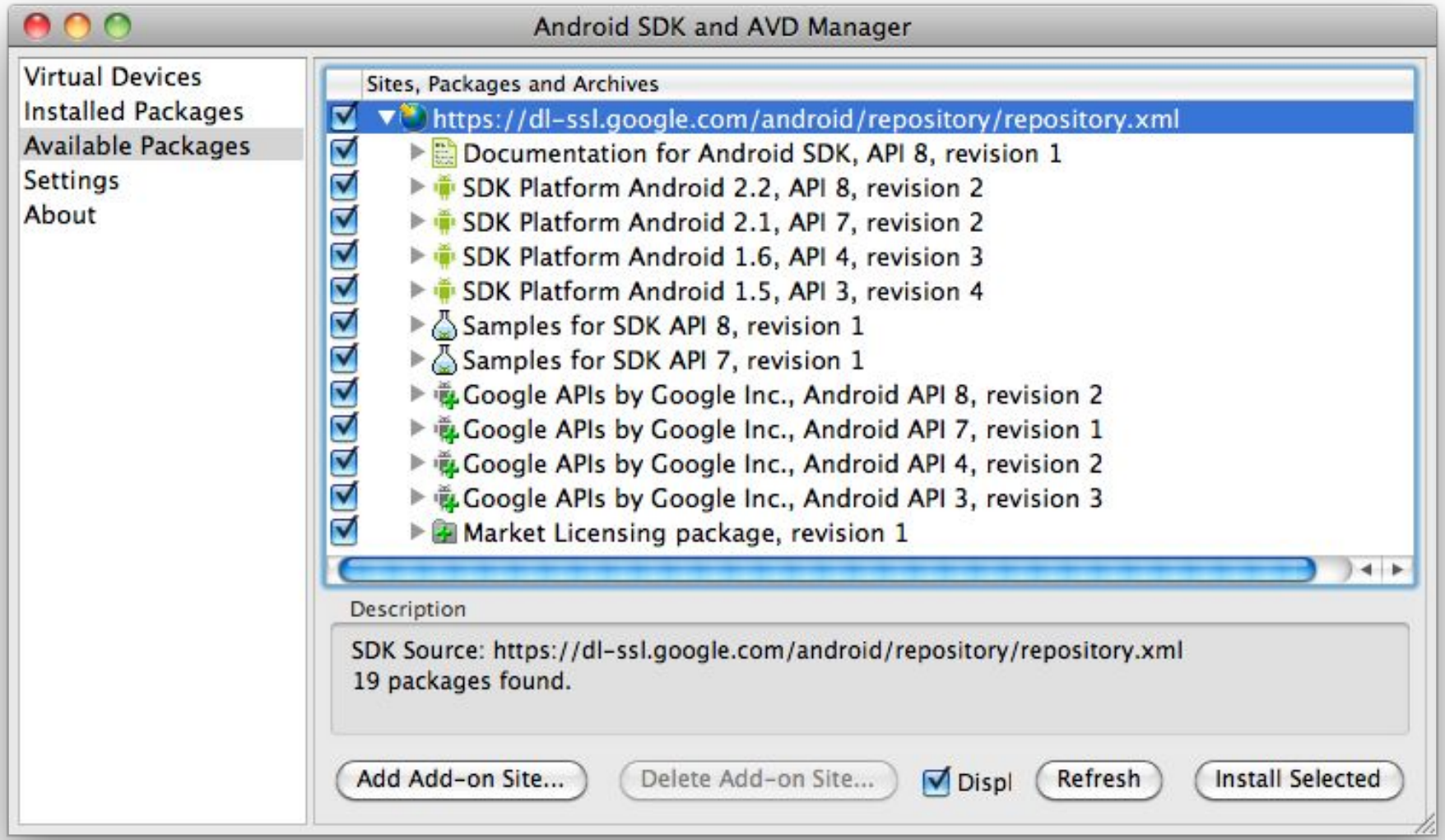
2) Unzip package at the place of your preference...
/Users/andres/Coding/Android/android-sdk-mac_x86

3) On a Mac OS X, look in your home directory for .bash_profile and proceed as for Linux. You can create the .bash_profile if you haven't already set one up on your machine and add
export PATH=${PATH}:<*your_sdk_dir*>/tool`export ANDROID_SDK=${PATH}:<your_sdk_dir>`
in our case this would be:
export PATH=${PATH}:/Users/andres/Coding/Android/android-sdk-mac_x86/tools`export ANDROID_SDK=`/Users/andres/Coding/Android/android-sdk-mac_x86
4) The last step in setting up your SDK is using a tool included the SDK starter package — the *Android SDK and AVD Manager* . You can start it from the terminal by typing android



Make sure to install:

- SDK Platform Android 2.1, API 7
- Google APIs by Google Inc., Android API 7



5.3 Installation on Linux

0) Make sure that you install the sun-java JDK first. On Ubuntu:
sudo apt-get install sun-java6-jdk

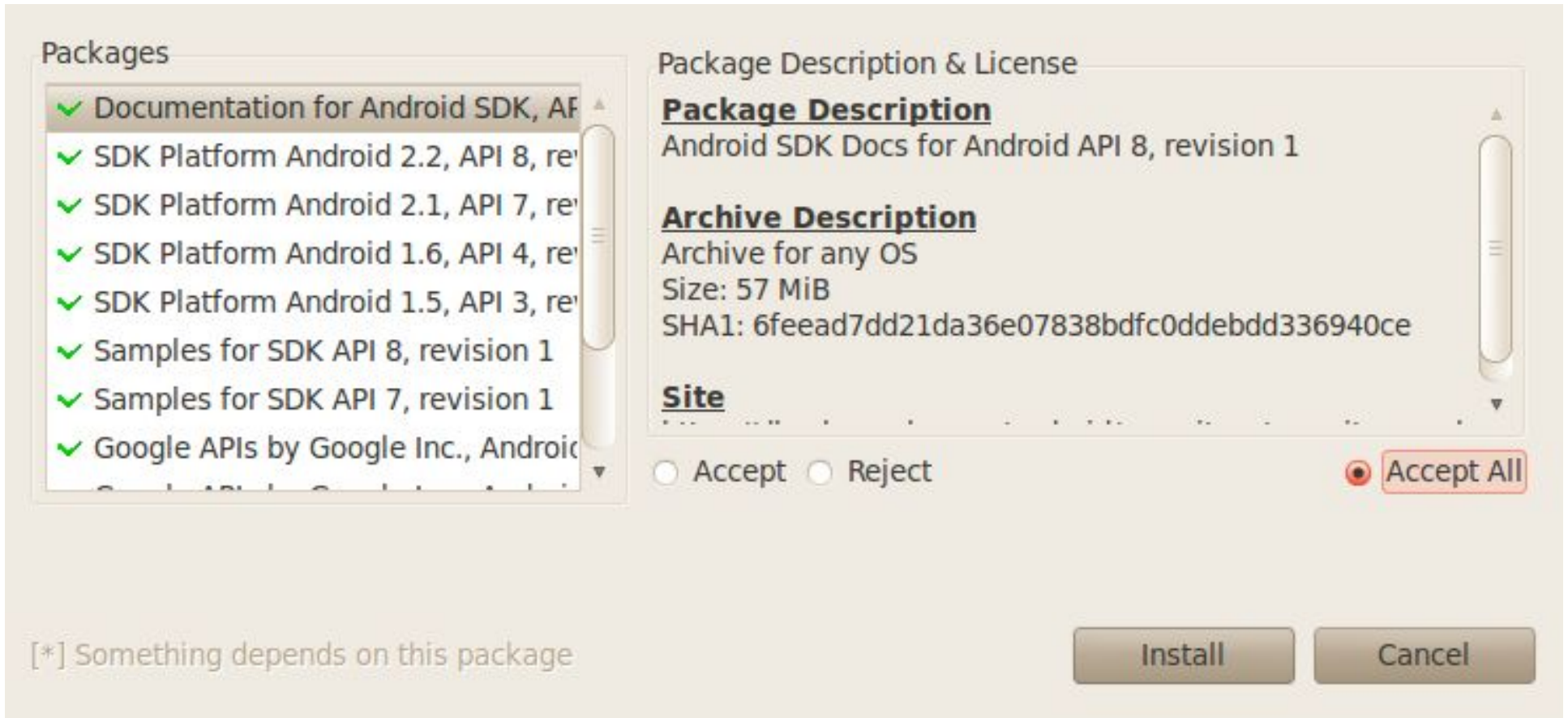
1) Download latest linux package from http://developer.android.com/sdk/index.html

2) Unzip package at the place of your preference...
/home/andres/Coding/Android/android-sdk-linux_x86
On 64 bits machine, you might need to install soma additional packages:
http://stackoverflow.com/questions/2710499/android-sdk-on-a-64-bit-linux-machine

3) Add the environmental variables PATH and ANDROID_SDK

On Linux, edit your ~/.bash_profile or ~/.bashrc file. Look for a line that sets the PATH environment variable and add the full path to the tools/ directory to it. If you don't see a line setting the path, you can add one:
export PATH=${PATH}:<*your_sdk_dir*>/toolsexport ANDROID_SDK=<*your_sdk_dir*>

4) The last step in setting up your SDK is using a tool included the SDK starter package — the *Android SDK and AVD Manager* . You can start it running android from the terminal.


Packages
Documentation for Android SDK, AF
SDK Platform Android 2.2, API 8, re
SDK Platform Android 2.1, API 7, re
SDK Platform Android 1.6, API 4, re
SDK Platform Android 1.5, API 3, re
Samples for SDK API 8, revision 1
Samples for SDK API 7, revision 1
Google APIs by Google Inc., Androic
Package Description & License
Package Description
Android SDK Docs for Android API 8, revision 1
Archive Description
Archive for any OS
Size: 57 MiB
SHA1: 6feead7dd21da36e07838bdfc0ddebdd336940ce
Site
Accept
Reject
Accept All
[*] Something depends on this package
Install
Cancel

6. How to install Processing

Go to the download section of the processing website:
http://processing.org/download/
and grab the version for your platform (OSX, Linux or Windows).

For windows and Linux, uncompress the zip/tgz in the desired location, in the case of OSX open the dmg package and copy to Applications

6.2 What can we do with Processing for Android

Single activity/ single view applications with no layouts!

2D and 3D (GPU-accelerated) graphics

Multitouch and keyboard input

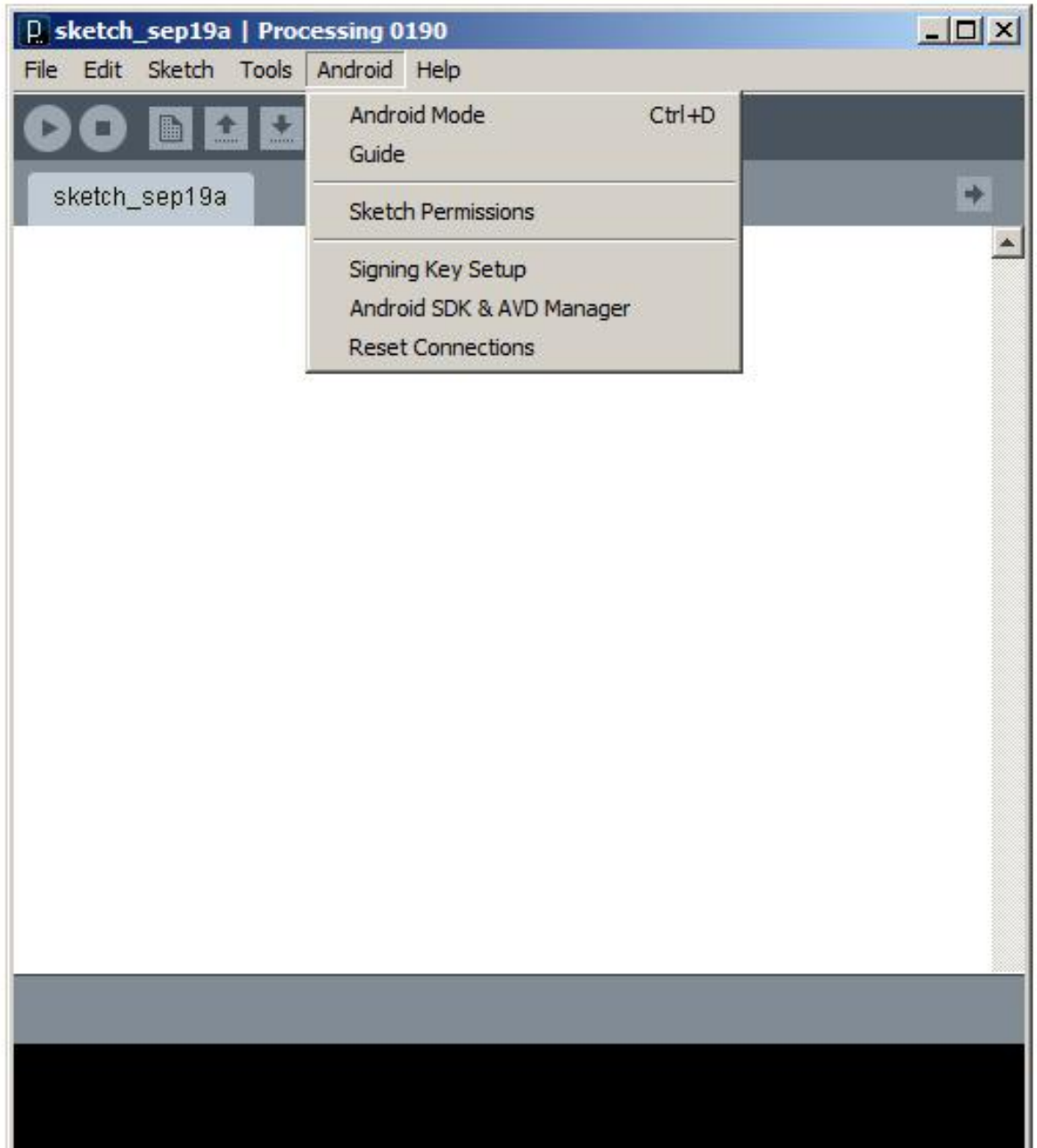
Extensible with libraries.

Lots of useful information in the
wiki: http://wiki.processing.org/w/Android

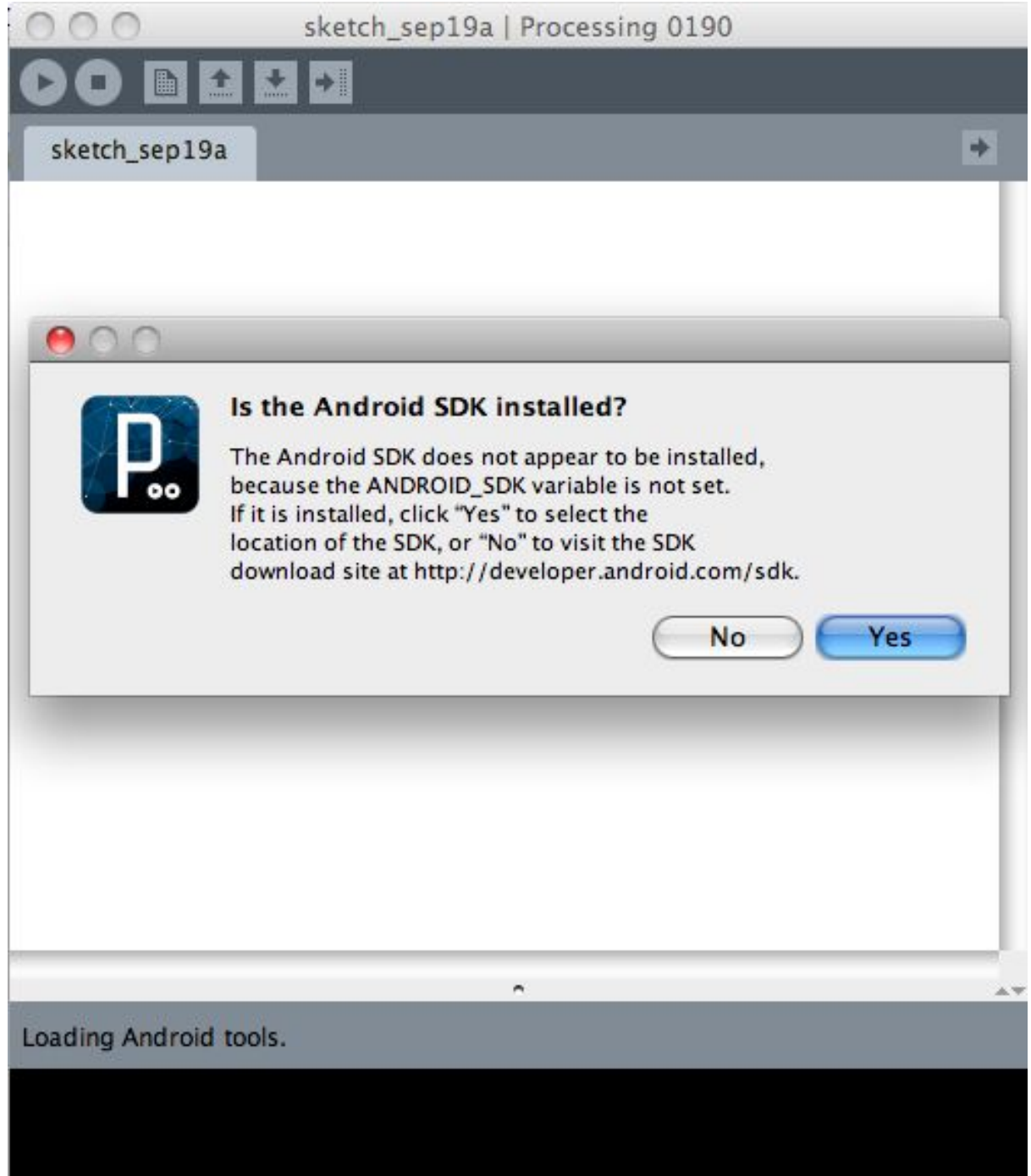
The forum is also useful site to post questions and answers:
http://forum.processing.org/android-processing

6.1 Android Mode in Processing

To run our code on the Android emulator or the actual Android device, we need to set enable the Android mode in the PDE



If missing the ANDROID_SDK variable, just select the folder when asked:

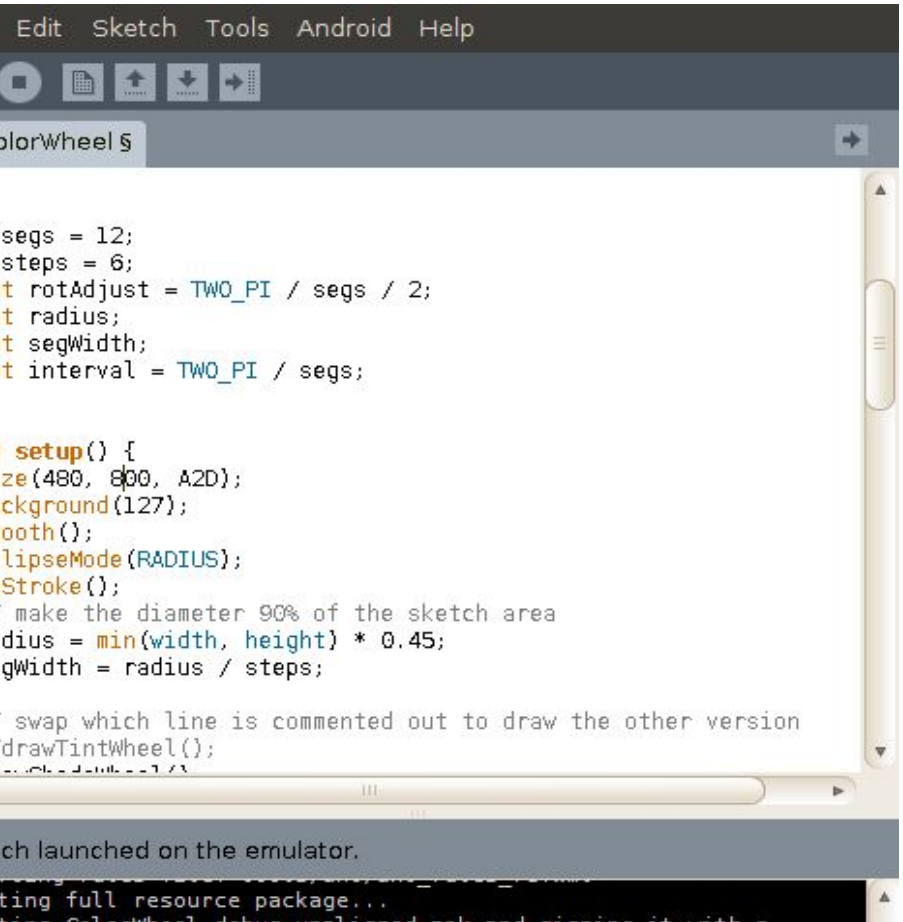


7. Running Processing sketches on Android

- Run - preprocess the current sketch, create an Android project, and run (debug) it in the Android emulator.
- Present - the same as Run, but run on a device (phone) that's attached by USB.
- Export - creates an 'android' folder that contains the files necessary to build an APK using Ant.
- Export to Application - same as export, but creates a signed version of the 'release' build.

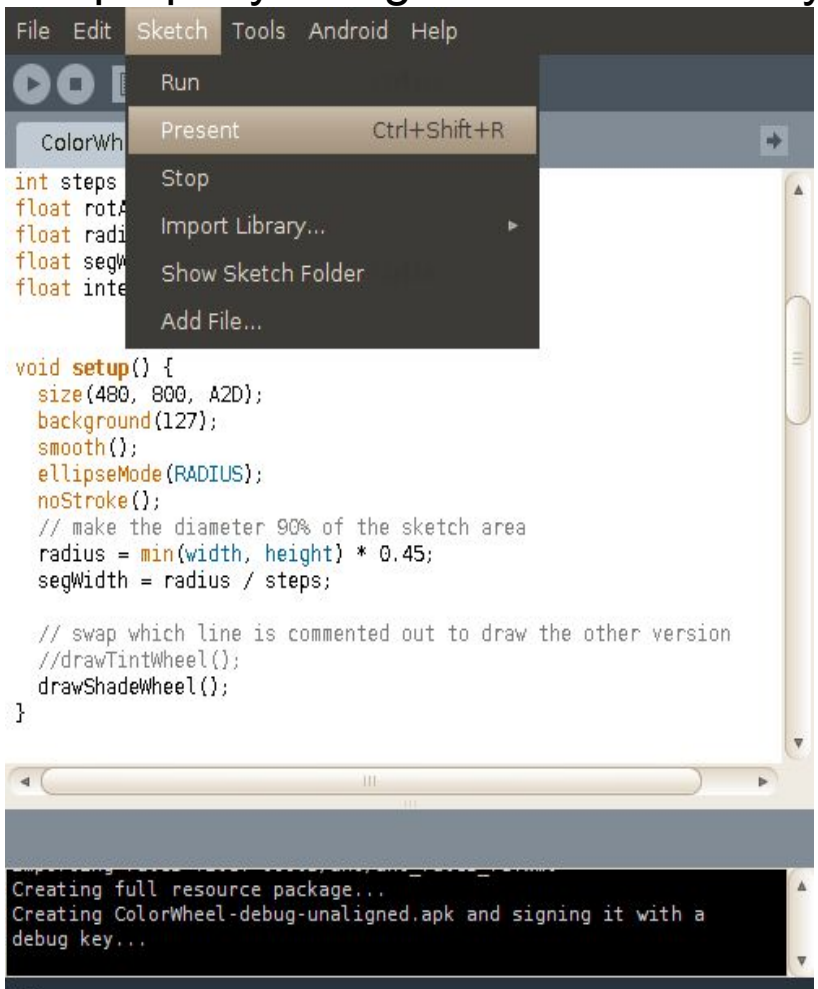
7.1 Running on the Android Emulator

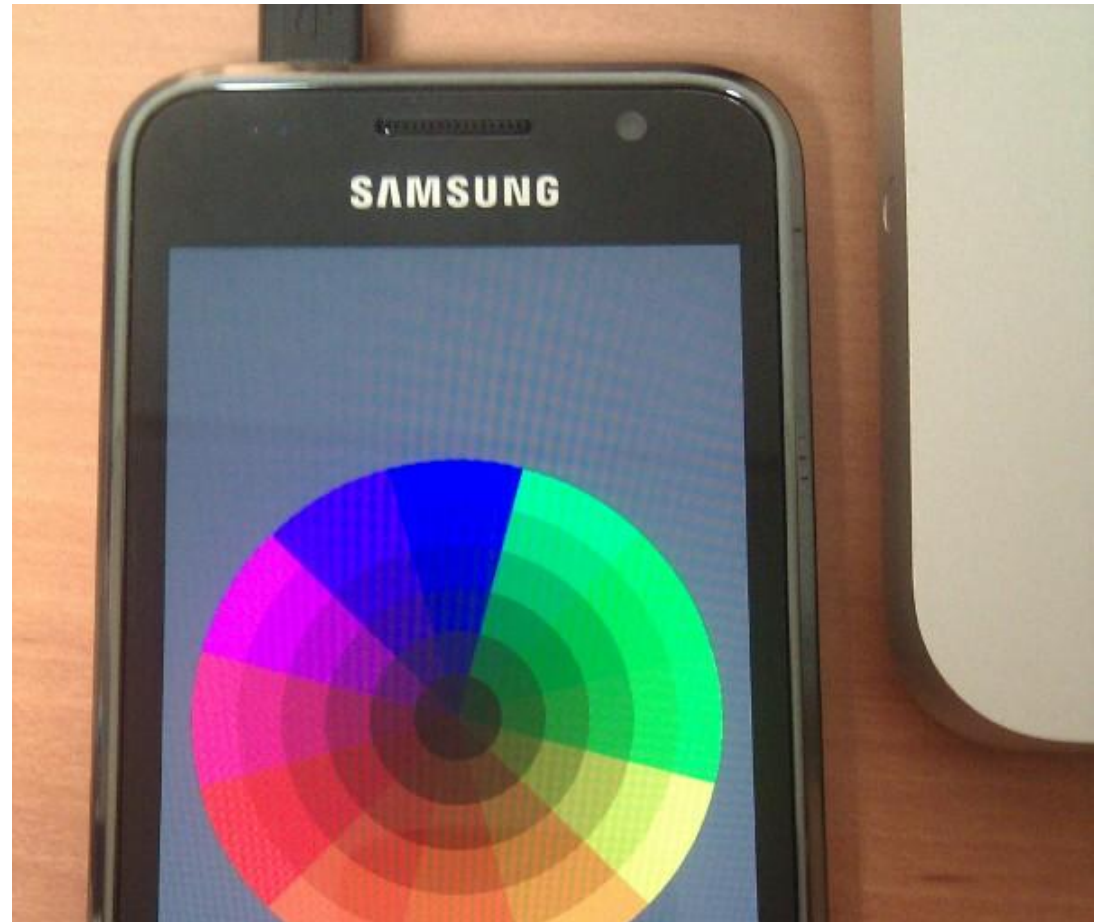
Tips: 1)Use keypad 7 or keypad 9 to rotate the emulator one direction or another. Or if you don't have a keypad or numlock, then press Ctrl-F12 (Ctrl-Fn-F12 on OS X) to rotate the screen.
2) The emulator stays running, even after you hit stop.
3) Pressing ESC in the emulator actually emulates hitting the 'back' key, though it may beep at you first.) By default, this will exit your application



7.2 Running on the Android device

The device has to be properly connected to the USB port.
It must be running at least Android 2.1
On Linux, you might need to run the following commands from the terminal:
adb-killserver
cd <sdk location>/tools
sudo ./adb start-server
to properly recognize the device. Type adb devices to get the list of currently connected



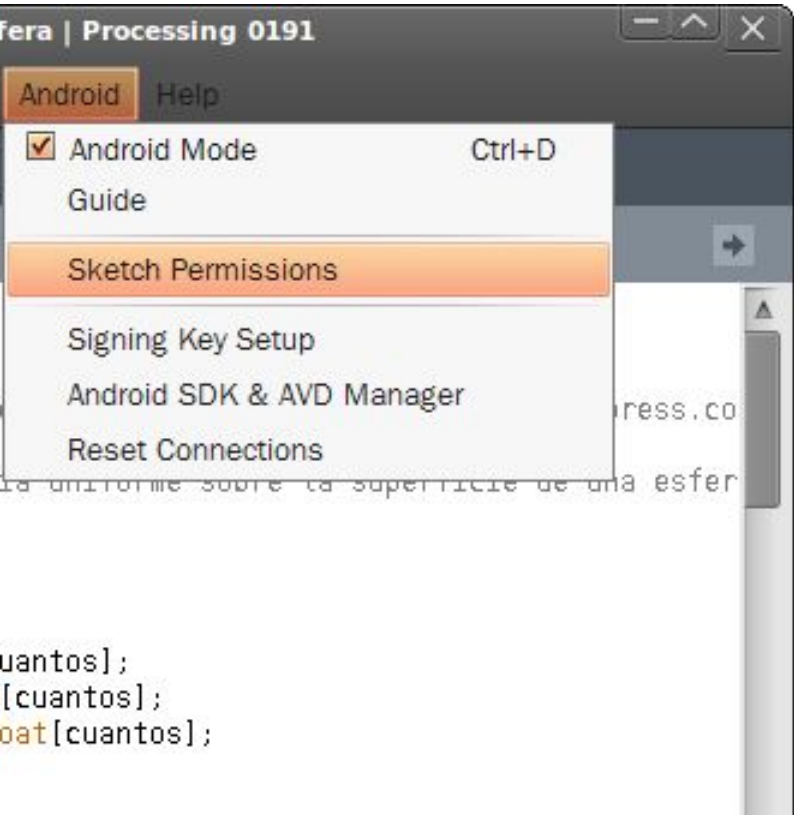


7.3 Handling Android Permissions

If you want to load data from the internet, or otherwise connect to other servers, you'll need to enable INTERNET permission for your sketch. To do so, use Tools → Android Permissions to bring up the permissions editor. Check the box next to internet.

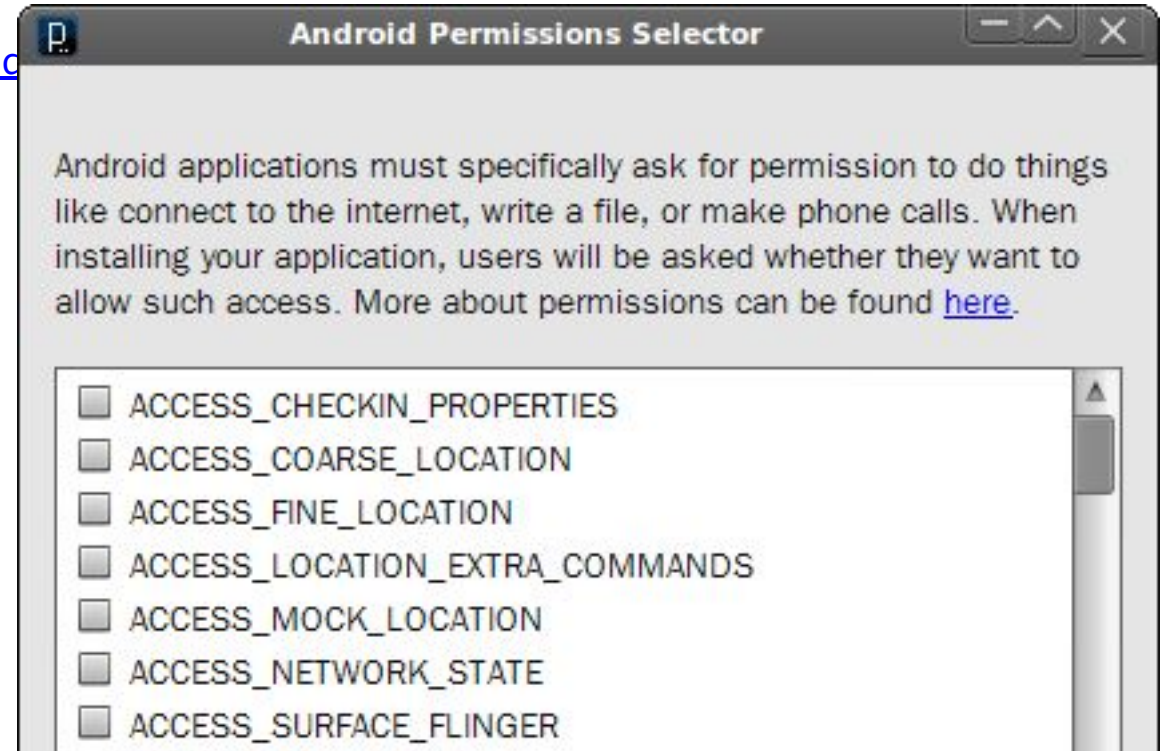
If you want to use methods like saveStrings() or createWriter, you'll need to enable WRITE_EXTERNAL_STORAGE so that you can save things to the built-in flash a plug-in card.

There are similar permissions for access to the phone, compass, etc. Look through the list in the permissions dialog, or check out other documentation that explains Android permissions in greater detail:



topics/security/security.html#permissions

ce/anc



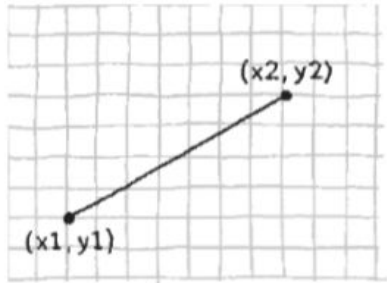
8. Basic Processing use

I will fill out from here to page 75

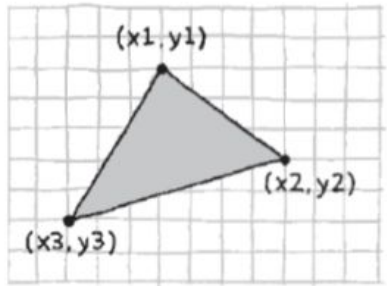
8.1 Drawing with basic 2D primitives (circle, rect, ellipse, shape)

Functions and Parameters

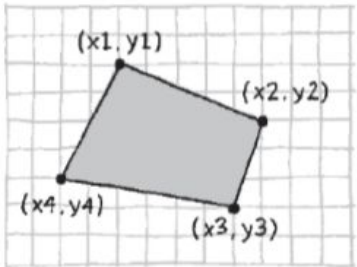
size(), point(), line(), Triangle(),
quad(), rect(), ellipse(), arc(), vertex()



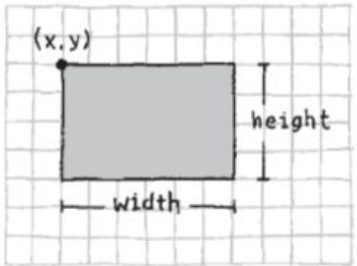

```
line(x1, y1, x2, y2)
```



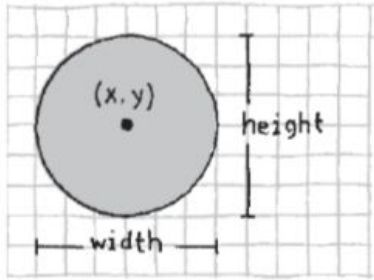

```
triangle(x1, y1, x2, y2, x3, y3)
```



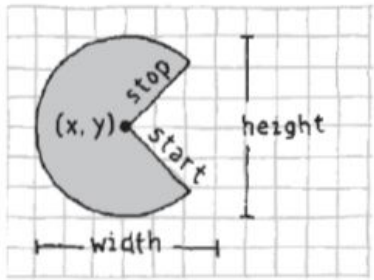

```
quad(x1, y1, x2, y2, x3, y3, x4, y4)
```



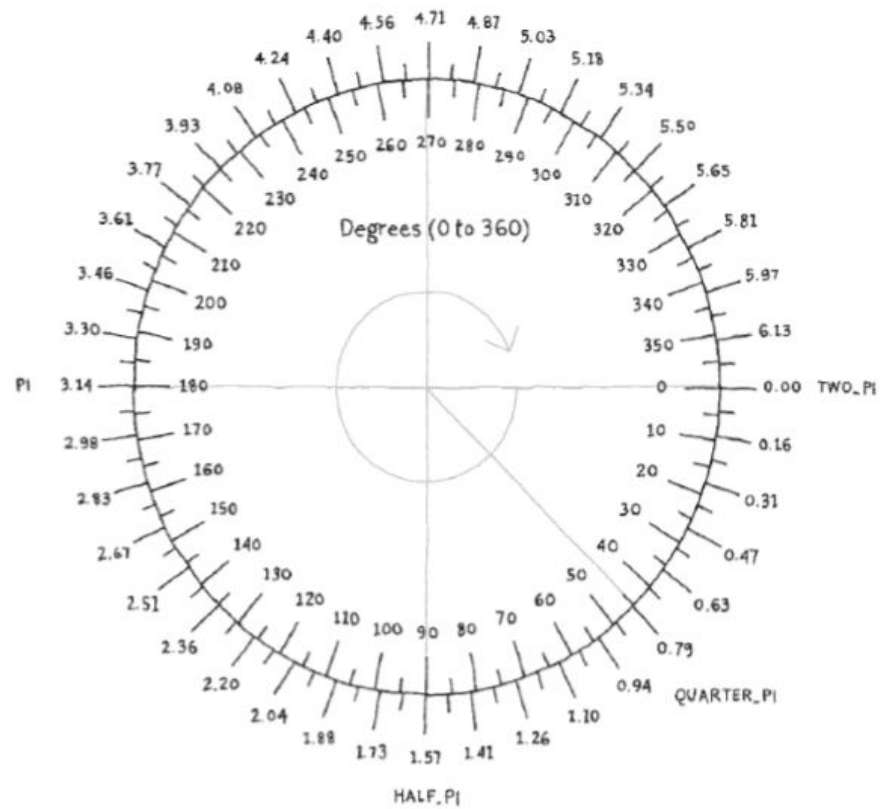

```
rect(x, y, width, height)
```



ellipse(x, y, width, height)

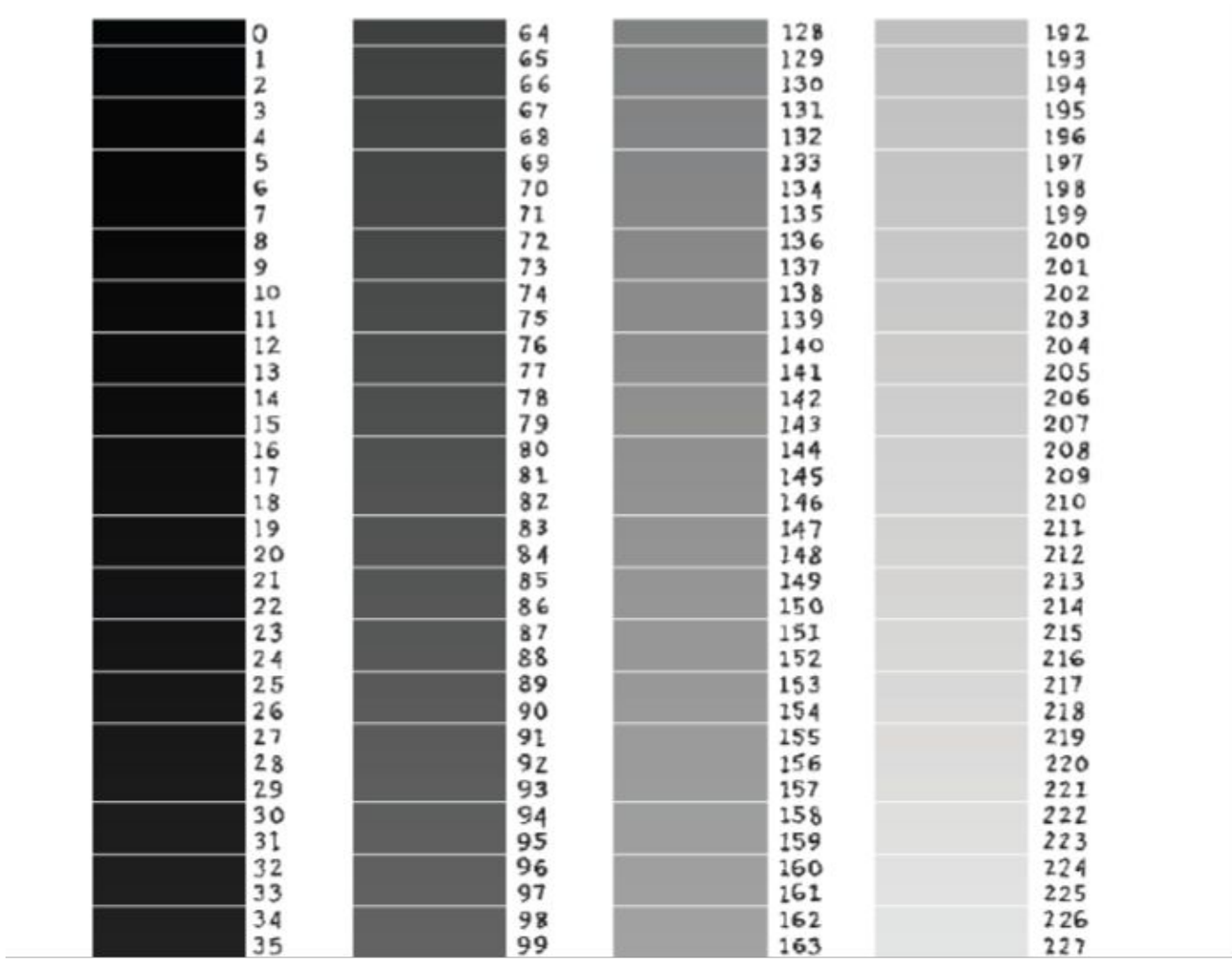


arc(x, y, width, heigh



8.2 Color in Processing

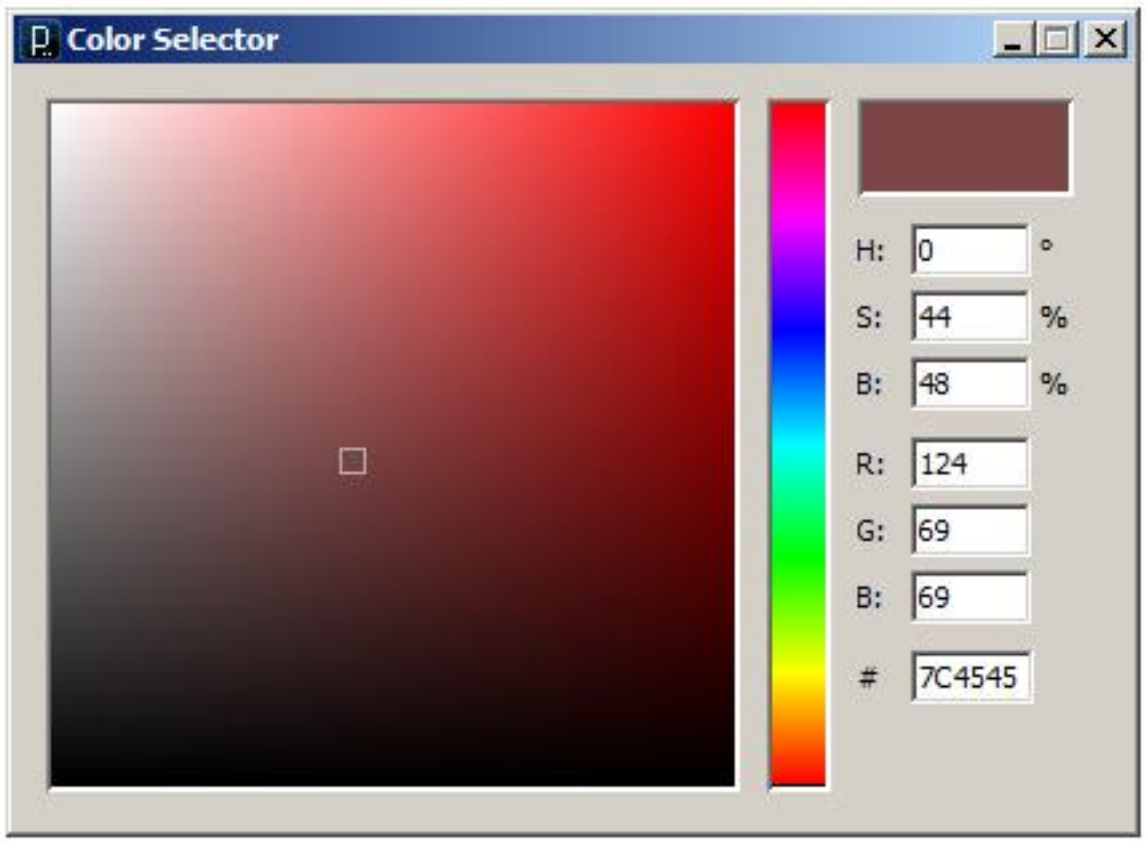
Gray scale



0(black) – 255(white)

fill(R,G,B);
fill(R,G,B, T);

RGB, HBS



8.3 Animation and motion: setup and draw

8.4 Media: images, fonts

8.5 Interaction: keyboard, touchscreen

from touchscreen i
need your help

8.6 Multitouch handling

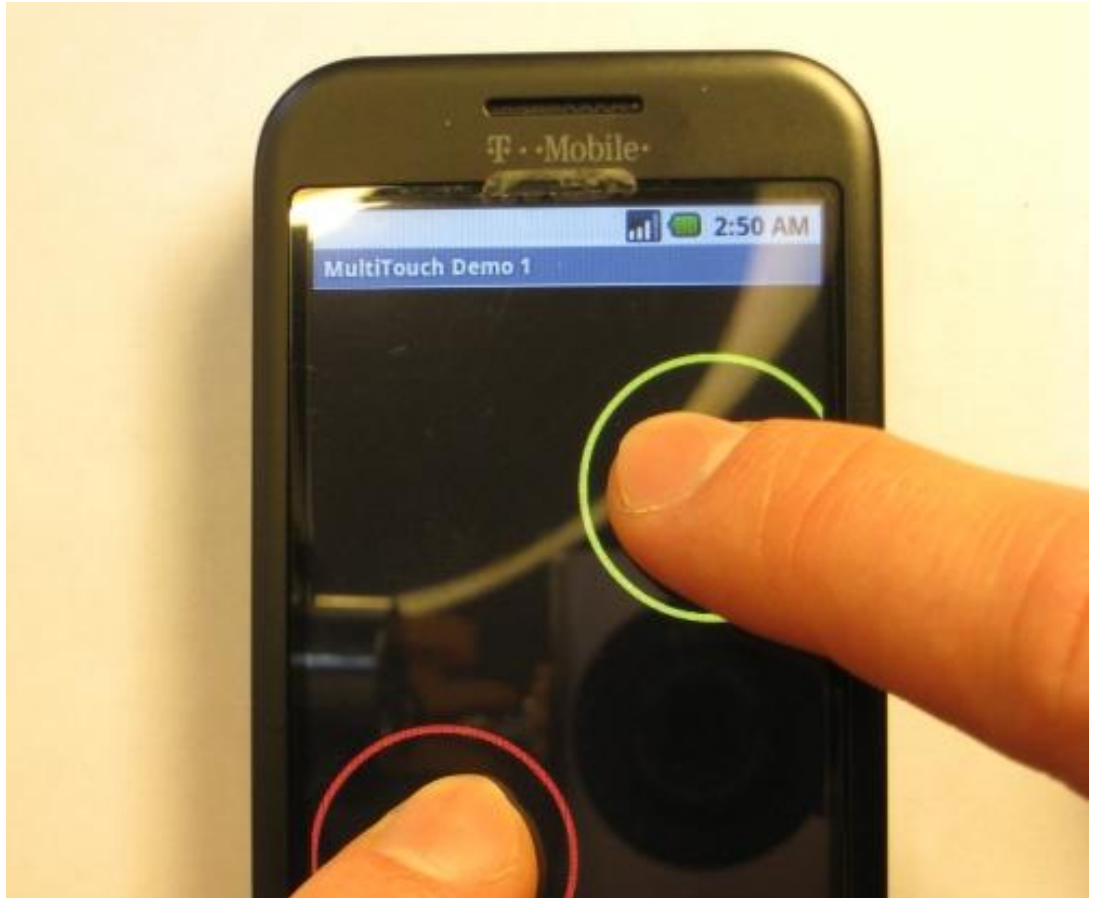
The simplest way to handle multitouch in Processing is by overloading the PApplet.surfaceTouchEvent(MotionEvent event) method:

```
import android.view.MotionEvent;


...
boolean surfaceTouchEvent(MotionEvent event) {
    switch (event.getAction() & MotionEvent.ACTION_MASK) {
      case MotionEvent.ACTION_POINTER_DOWN:
        // User is pressing down another finger.
        break;
      case MotionEvent.ACTION_POINTER_UP:
        // User is released one of the fingers.
        break;
      case MotionEvent.ACTION_MOVE:
        // User is moving fingers around. We can calculate the distance
        // between the first two fingers, for example:
        float x = event.getX(0) - event.getX(1);
        float y = event.getY(0) - event.getY(1);
        float d = sqrt(x * x + y * y);
        break;
      }
    }

    return super.surfaceTouchEvent(event);
  }
```

Another way is to use the code from the Android Multitouch Controller:

http://code.google.com/p/android-multitouch-controller/

It filters out "event noise" on Synaptics touch screens
It simplifies the somewhat messy and inconsistent MotionEvent touch point API
The controller also supports pinch-zoom and pinch-rotate
Anisotropic scaling (using different X and Y scale)



8.6 Extension: libraries (oscP5, Ketai)

Processing for Android retains the extensibility of Processing for PC/Mac, through the use of external libraries. Here we discuss two libraries already available:

* oscP5: It is an implementation of OSC (Open Sound Control) a network protocol developed at cnmat, UC Berkeley. Open Sound Control is a protocol for communication among computers, sound synthesizers, and other multimedia devices that is optimized for modern networking technology and has been used in many application areas.
http://www.sojamo.de/libraries/oscP5/

EXAMPLE

```
scP5.*;
etP5.*;

oscP5;
ress serverLocation;

up() {

 = new OscP5(this, 15000);
Location = new NetAddress("192.168.0.70", 9999);



usePressed() {
essage message = new OscMessage("/press");
ge.add(mouseX);
```

SERVER EXAMPLE

```
import oscP5.*;
import netP5.*;

  OscP5 oscP5;
  NetAddress location;

  void setup() {
    ...
    oscP5 = new OscP5(this, andiamo.config.port);
    location = new NetAddress("192.168.0.70", 9999);
   ...
}

void oscEvent(OscMessage message) {
    // println("received osc message");
    print(message);
```

Ketai: universal sensing

THIS PART IS IN PROGRESS

http://ketai.aa.uic.edu/community/pg/groups/705/daniel-sauter-jesus-duran-android-devices-as-universal-sensors/

This workshop focuses on using the Ketai library for processing, allowing to register the native sensors supported by the Android platform. The workshop covers data capture, processing and export via Ketai Motion, and introduces image capture via Ketai Vision. Bring or share your Android device to take full advantage of your mobile phone as universal sensor. BYOA is highly recomended.

PART II: Fast 3D Graphics in Processing for Android

7. OpenGL and Processing

OpenGL ES

3D drawing in Android is handled by the GPU (Graphic Processing Unit) of the device.

The most direct way to program 3D graphics on Android is by means of OpenGL ES.

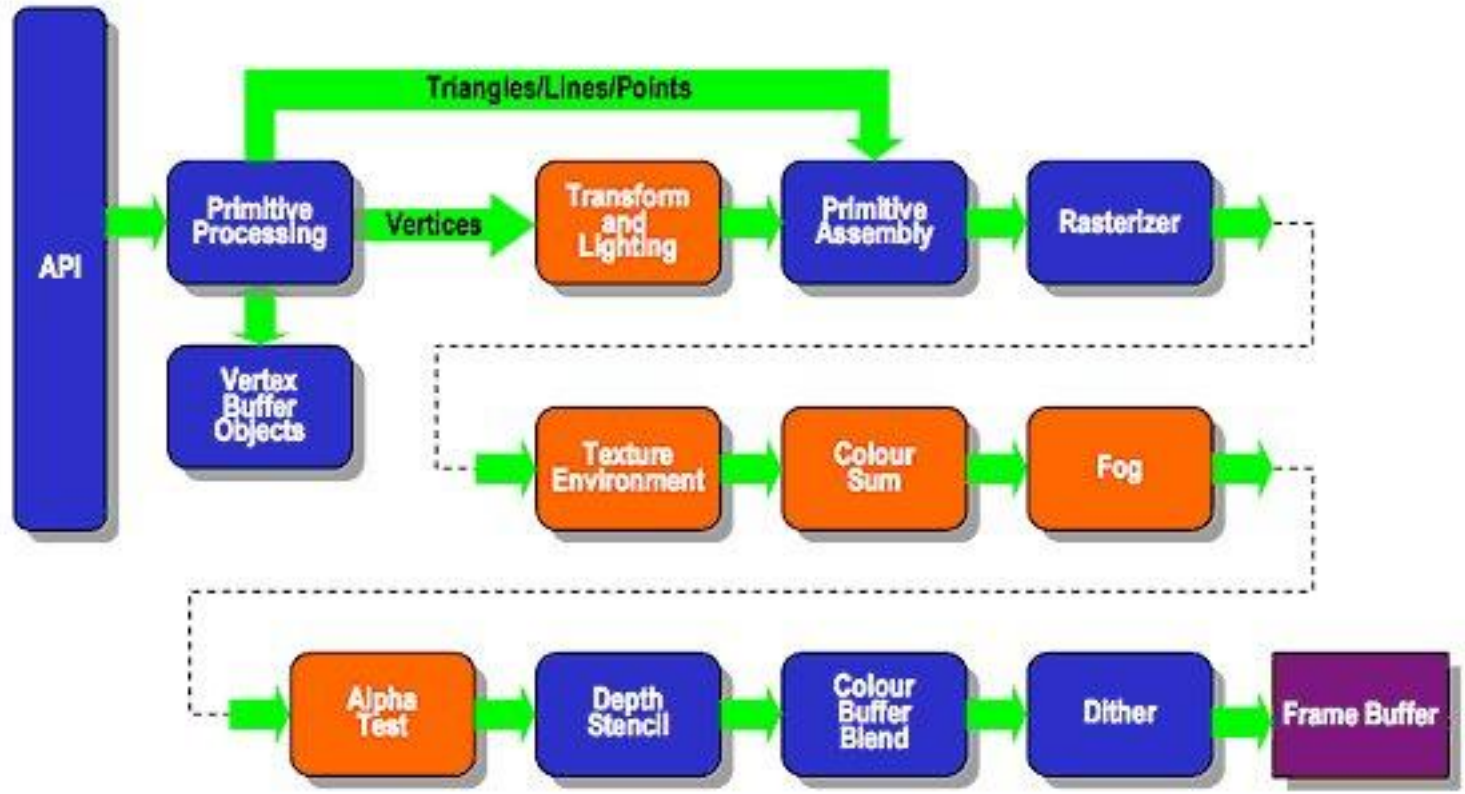
OpenGL ES is a cross-platform API for programming 2D and 3D graphics on embedded devices (consoles, phones, appliances, etc).

OpenGL ES consists in a subset of OpenGL.

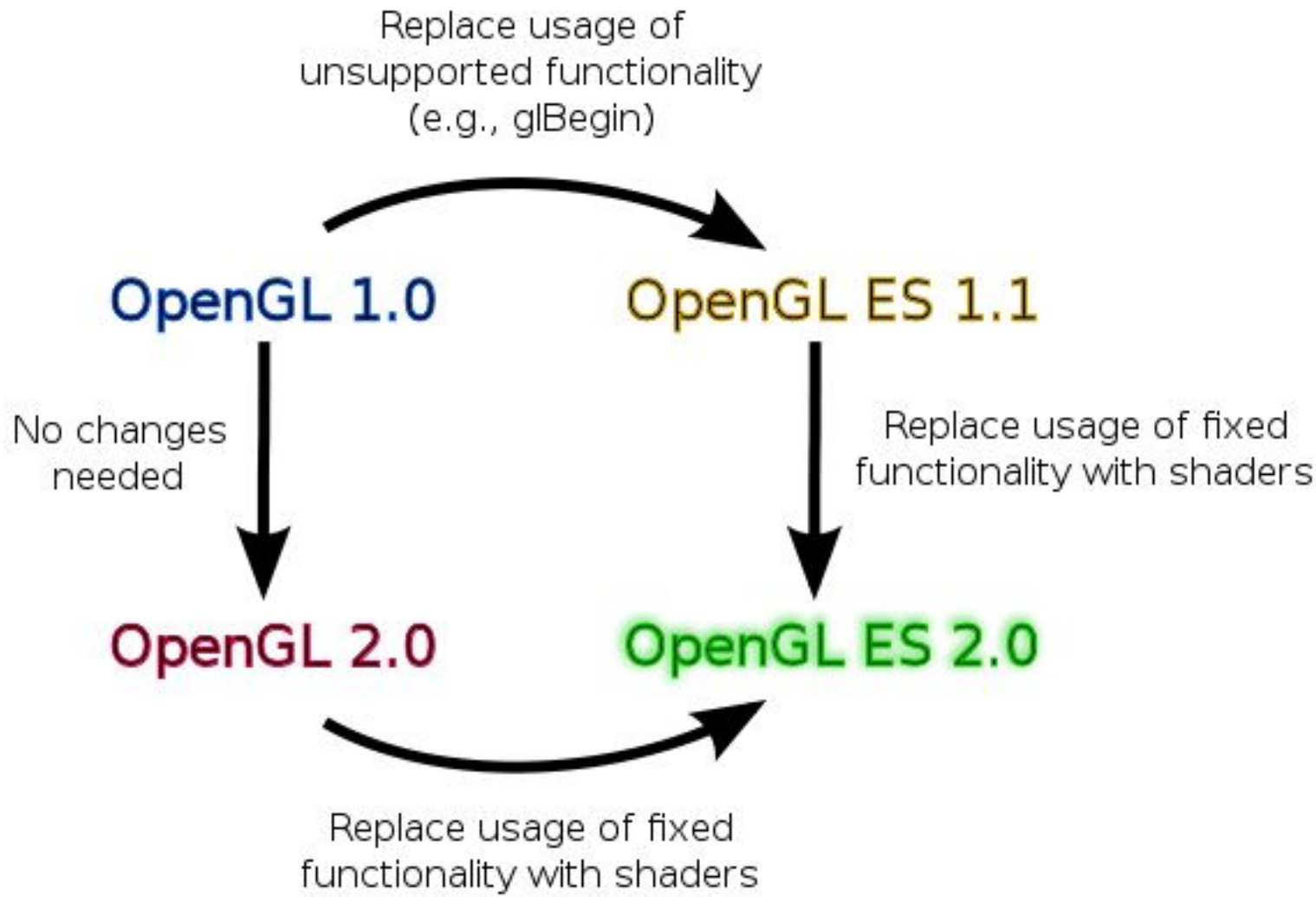
http://developer.android.com/guide/topics/graphics/opengl.html
http://www.khronos.org/opengles/

OpenGL ES is the 3D API for other platforms, such as Nokia and iPhone:

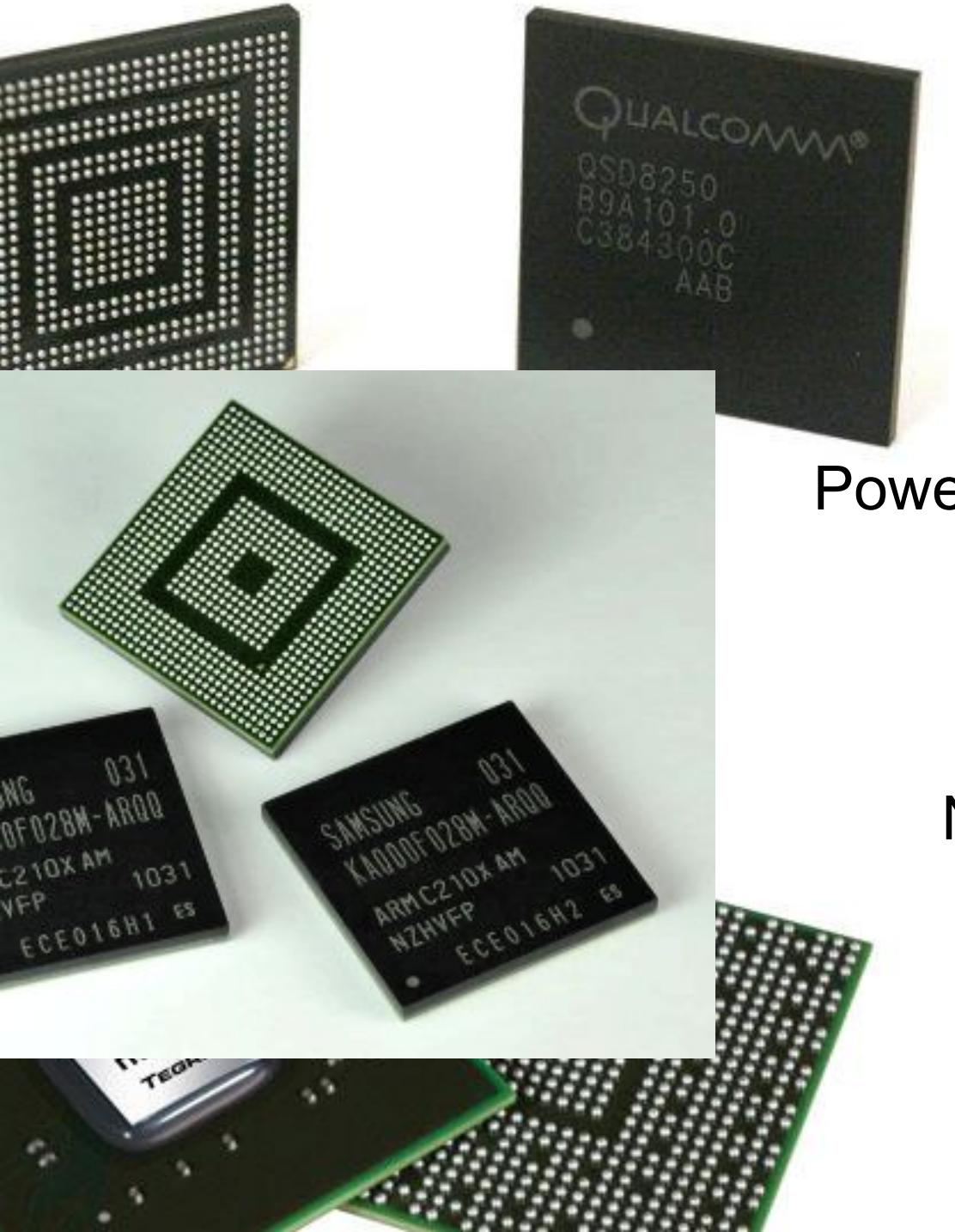
The graphics pipeline is the sequence of steps in the GPU from the data (coordinates, textures, etc) provided through the OpenGL ES API to the final image on the screen (or Frame Buffer)



http://wiki.maemo.org/OpenGL-ES



Mobile GPUs



Qualcomm Adreno

PowerVR SGX

NVidia Tegra

	Medium performance	High performance
Adreno HD)	200 (Nexus 1, HTC Evo, Desire)	205 (HTC Desire
PowerVR Captivate)	SVG 530 (Droid, Droid 2, DroidX)	SVG 540 (Galaxy S, Vibrant,
NVidia core		Tegra 250, dual (none yet)

Useful websites:

http://smartphonebenchmarks.com/

http://www.glbenchmark.com/latest_results.jsp?

Hardware requirements for 3D in Processing for Android

In principle, any GPU that supports OpenGL ES 1.1

GPUs such as the Adreno 200 or PowerVR SVG 540/530 are recommended.

Older GPUs found on devices such as the G1 might work, but the performance is limited.

As a general requirement for Processing, Android 2.1. However, certain OpenGL features were missing in 2.1, so froyo (2.2) is needed to take full advantage of the hardware.

The A3D renderer

In Processing for Android there is no need to use OpenGL ES directly (although it is possible).

The drawing API in Processing uses OpenGL internally when using the A3D (Android 3D) renderer.

The renderer in Processing is the module that executes all the drawing commands.

During the first part of this workshop we used the A2D renderer, which only supports 2D drawing.

The renderer can be specified when setting the resolution of the output screen with the size() command:
size(width, height, renderer), where renderer = A2D or A3D

If no renderer is specified, then A2D is used by default.

Offscreen drawing

We can create an offscreen A3D surface by using the createGraphics() method:

```
PGraphicsAndroid3D pg;

void setup() {
    size(480, 800, A3D);
    pg = createGraphics(300, 300, A3D);
    ...
}
```

The offscreen drawing can be later used as an image to texture an object or to combine with other layers:

```
void draw() {
  pg.beginDraw();
  pg.rect(100, 100, 50, 40);
  pg.endDraw();

  ...
  cube.setTexture(pg.getOffscreenImage());
  ...
}
```

8. Geometrical transformations

1) The coordinate system in Processing is defined as follows with respect to the screen (X axis running from left to right, Y axis from top to bottom and negative Z pointing away from the screen).
2) In particular, the origin is at the upper left corner of the screen.
3) Geometrical transformations (translations, rotations and scalings) are applied to the entire coordinate system.

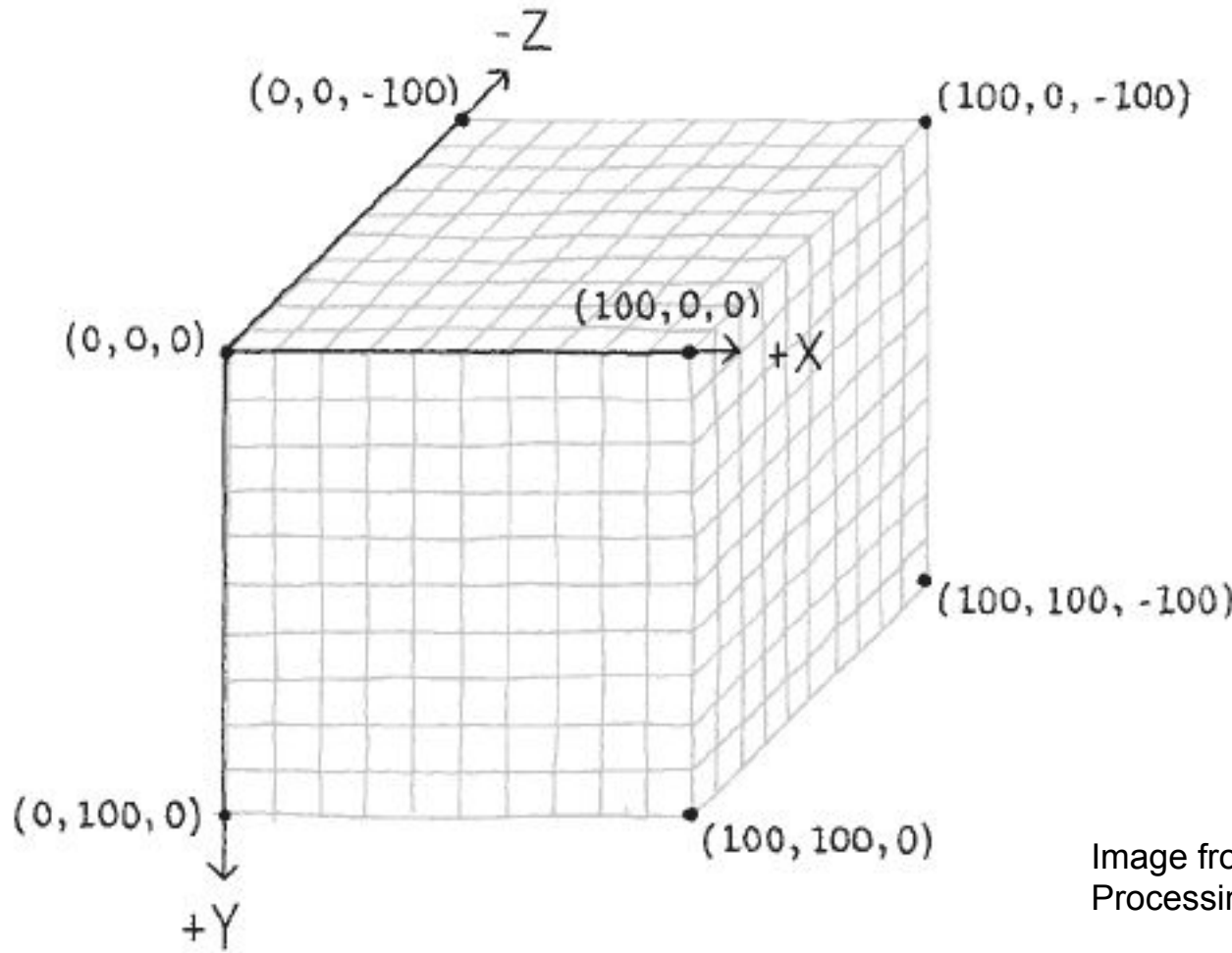
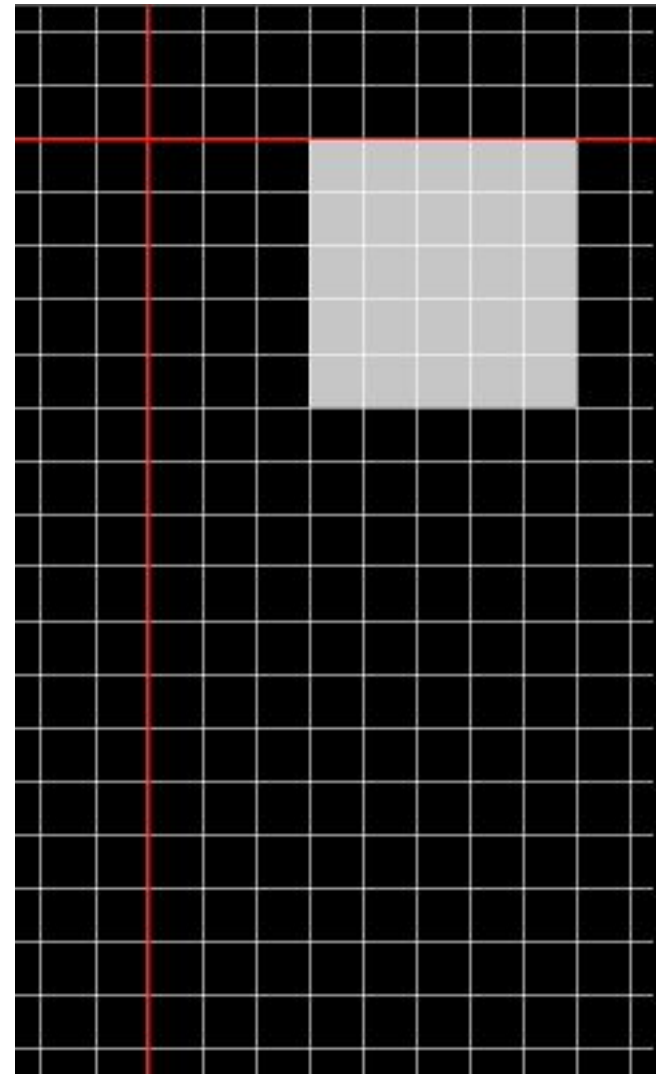


Image from "Getting Started With Processing"

Translations

The translate(dx, dy, dz) function displaces the coordinate system by the specified amount on each axis.

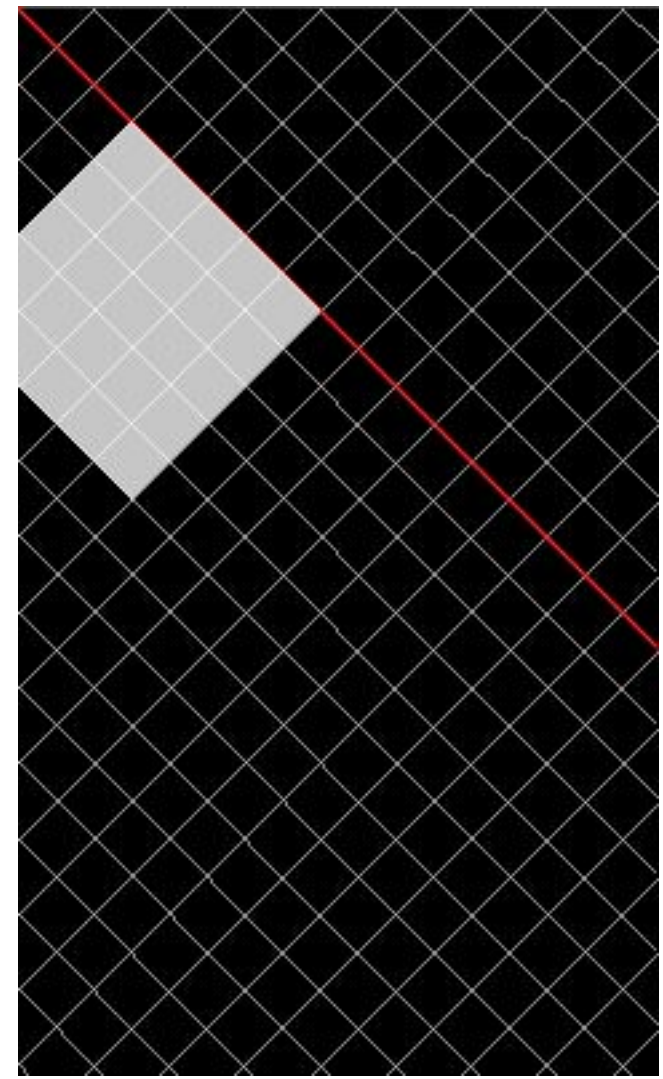
```
void setup() {
  size(240, 400, A3D);
  stroke(255, 150);
}

void draw() {
  background(0);

  translate(50, 50, 0);

  noStroke();
  fill(255, 200);
  rect(60, 0, 100, 100);
}
```



Rotations

Rotations have always a rotation axis that passes through the origin of the coordinate system. This axis could be the X, Y, Z axis, or an arbitrary vector: rotateX(angle), rotateY(angle), rotateZ(angle), rotate(ange, x, y, z)

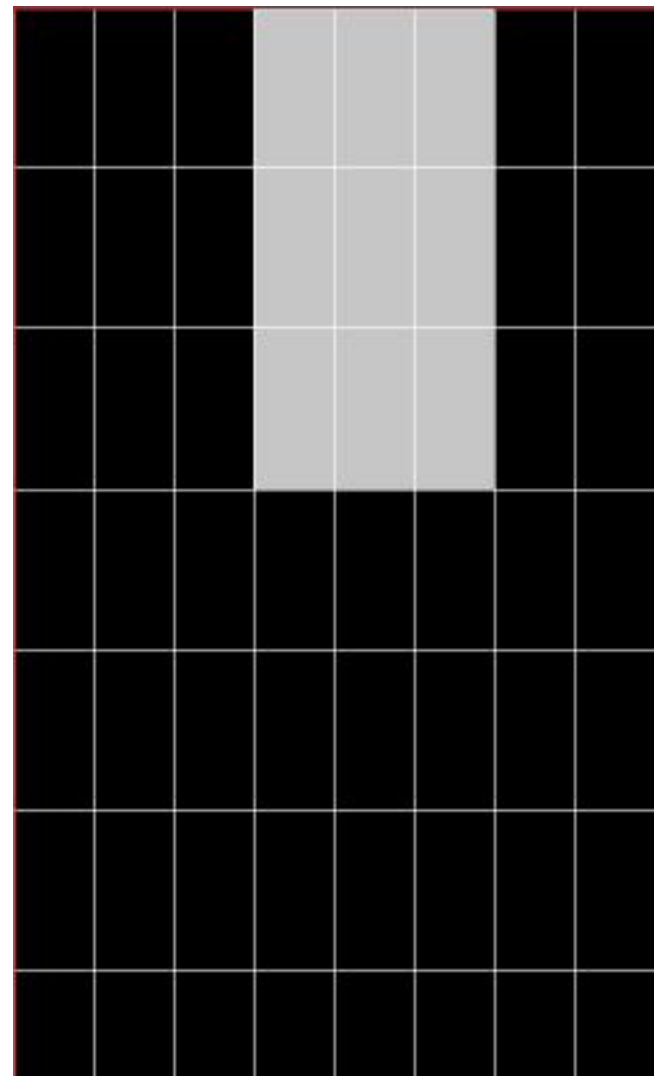
```
void setup() {
  size(240, 400, A3D);
  stroke(255, 150);
}

void draw() {
  background(0);
  rotateZ(PI / 4);
  noStroke();
  fill(255, 200);
  rect(60, 0, 100, 100);
}
```



Scaling

Scaling can be uniform (same scale factor on each axis) or not, since the scale(sx, sy, sz) allows to specify different factors along each direction.

```
void setup() {
  size(240, 400, A3D);
  stroke(255, 150);
}

void draw() {
  background(0);
  scale(1.5, 3.0,
1.0);
  noStroke();
  fill(255, 200);
  rect(60, 0, 60, 60);
}
```

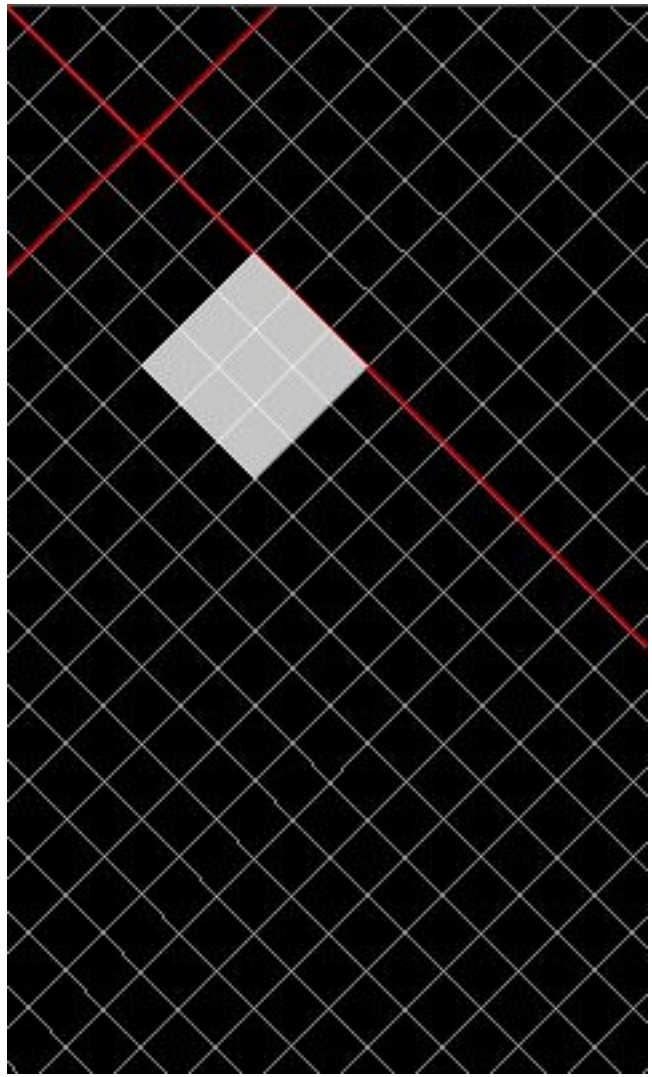


Just a couple of quick points about geometrical transformations...

1) By combining translate() with rotate(), the rotations can be applied around any desired point.

2) The order of the transformations is important

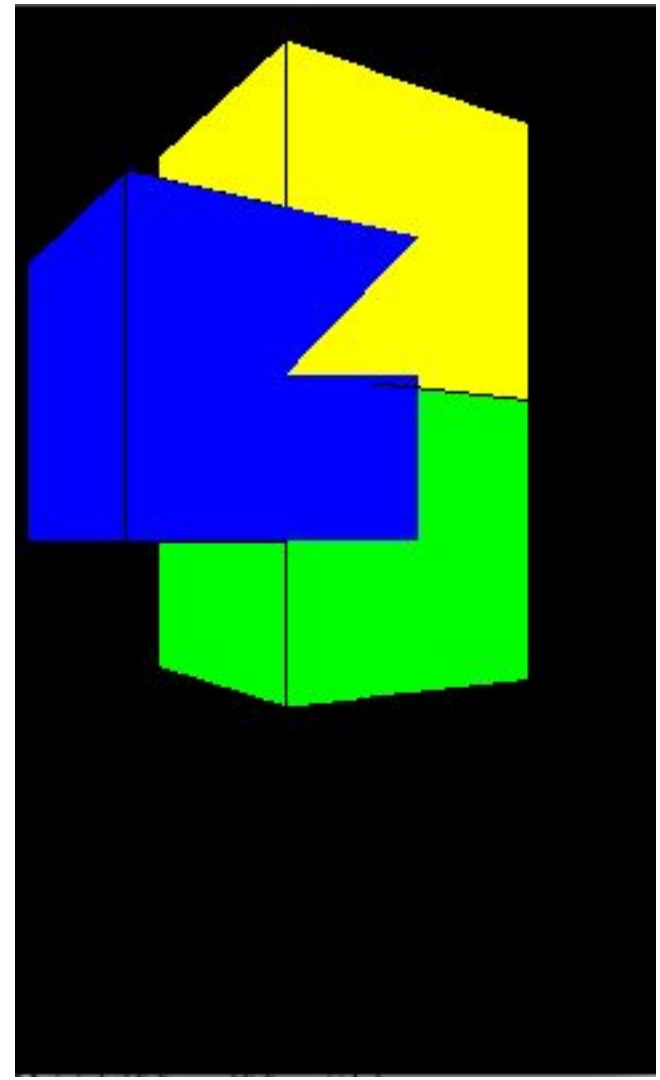
```
void setup() {
  size(240, 400, A3D);
  stroke(255, 150);
}

void draw() {
  background(0);
  translate(50, 50, 0);
  rotateZ(PI / 4);
  noStroke();
  fill(255, 200);
  rect(60, 0, 60, 60);
}
```



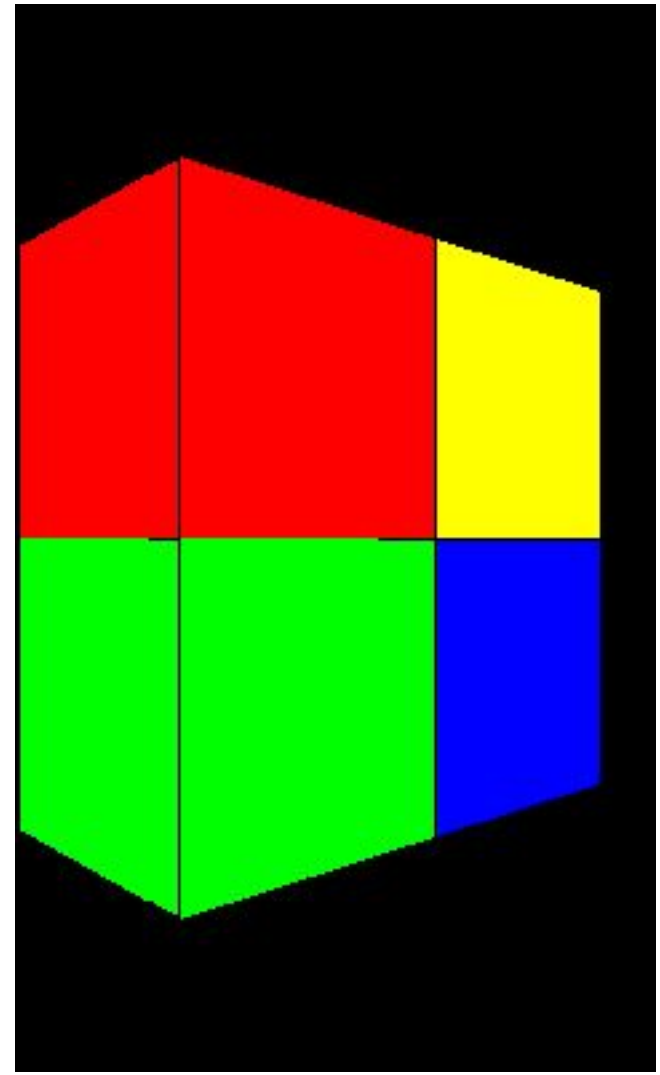
The transformation stack

The transformation stack we have in the 2D mode is also available in A3D through the functions pushMatrix() and popMatrix().
All the geometric transformations issued between two consecutive calls to pushMatrix() and popMatrix() will not affect the object drawn outside.

```
void setup(){
  size(240, 400, A3D);
}
void draw(){
  background(0);
  translate(width/2, height/2);
  rotateY(frameCount*PI/60);
  translate(-50, -50);
  fill(255, 0, 0);
  box(100, 100, 100);
  translate(50, -50);
  fill(255, 255, 0);
  box(100, 100, 100);
  translate(-50, 50);
  fill(0, 0, 255);
  box(100, 100, 100);
  translate(50, 50);
  fill(0, 255, 0);
```

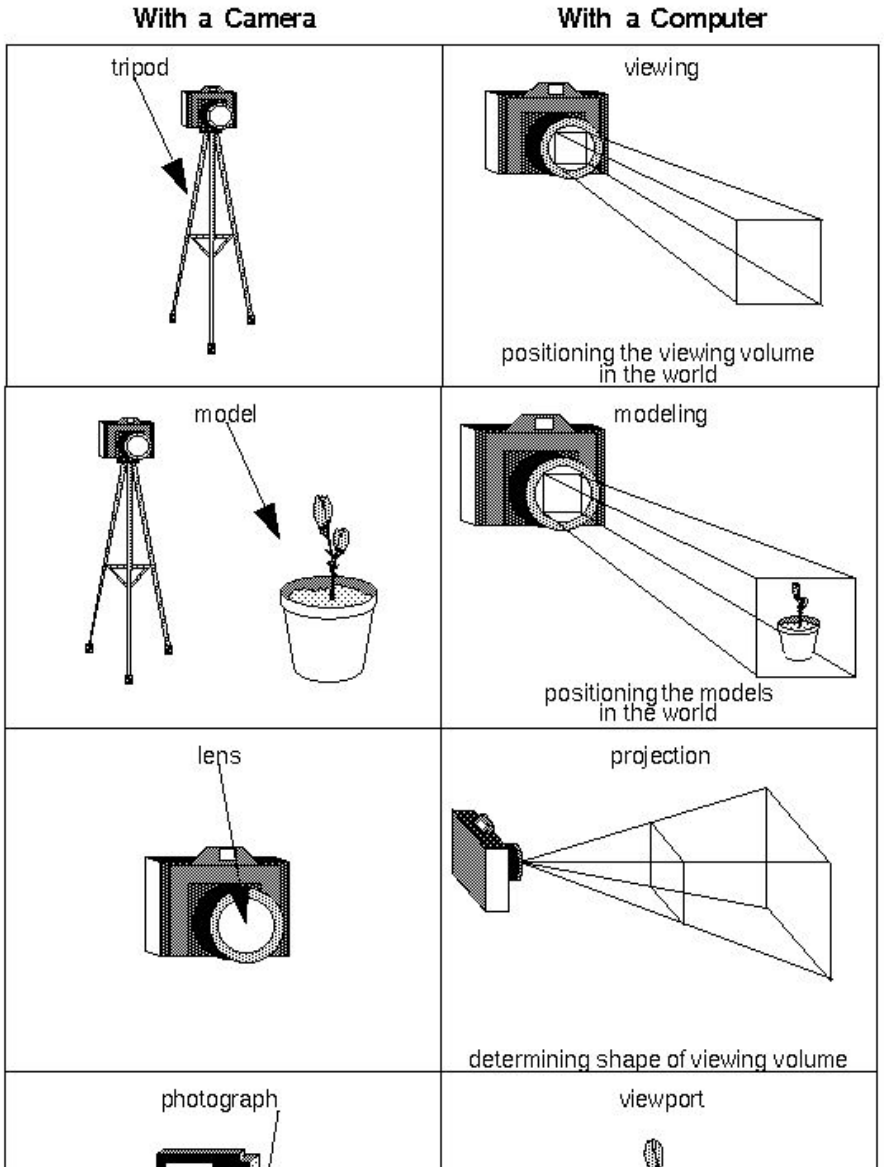


```
void setup(){
  size(240, 400, A3D);
}
void draw(){
  background(0);
  translate(width/2, height/2);
  rotateY(frameCount*PI/60);
  pushMatrix();
  translate(-50, -50);
  fill(255, 0, 0);
  box(100, 100, 100);
  popMatrix();
  pushMatrix();
  translate(50, -50);
  fill(255, 255, 0);
  box(100, 100, 100);
  popMatrix();
  pushMatrix();
  translate(50, 50);
  fill(0, 0, 255);
  box(100, 100, 100);
  popMatrix();
  pushMatrix();
  translate(-50, 50);
  fill(0, 255, 0);
  box(100, 100, 100);
```



9. Camera and perspective

Setting the view of the scene in A3D requires setting the camera location and the viewing volume.
This can be compared with setting a physical camera in order to take a picture:




```
camera(eyeX, eyeY, eyeZ,
centerX, centerY, centerZ,
upX, upY, upZ)
```

(from the OpenGL Red Bo

perspective(fov, aspect, zNear, zFar)ortho
right, bottom, top, near, far)

Camera placement

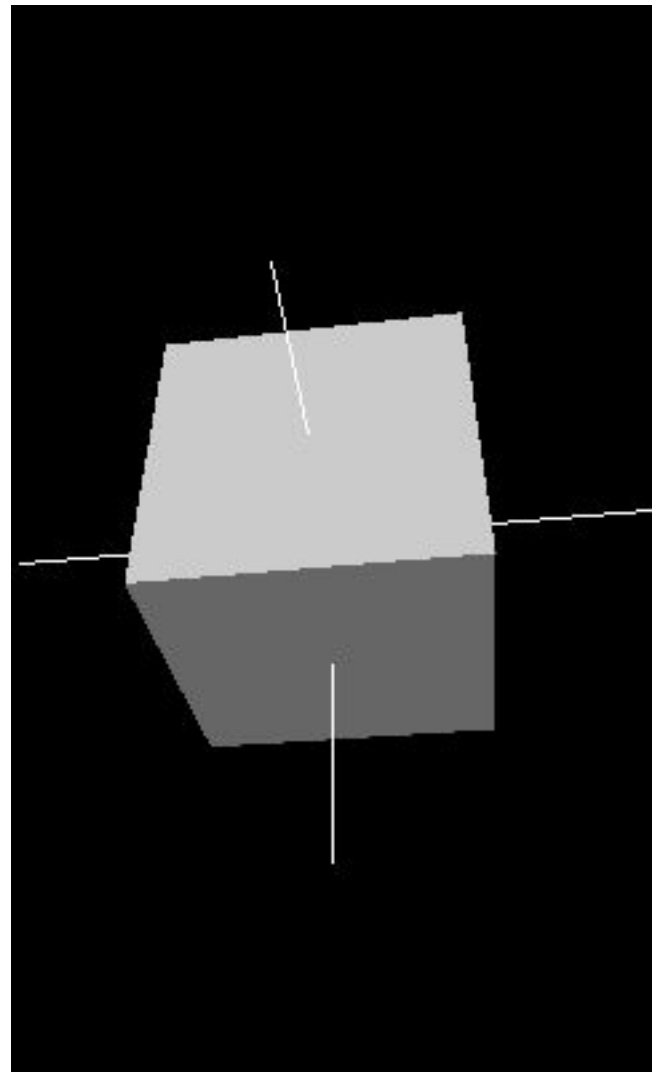
The camera placement is specified by the eye position, the center of the scene and which axis is facing upwards:

```
camera(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ)
```

If camera() is not called, A3D automatically does it with the following values:

width/2.0, height/2.0, (height/2.0) / tan(PI*60.0 / 360.0), width/2.0, height/2.0, 0, 0, 1, 0

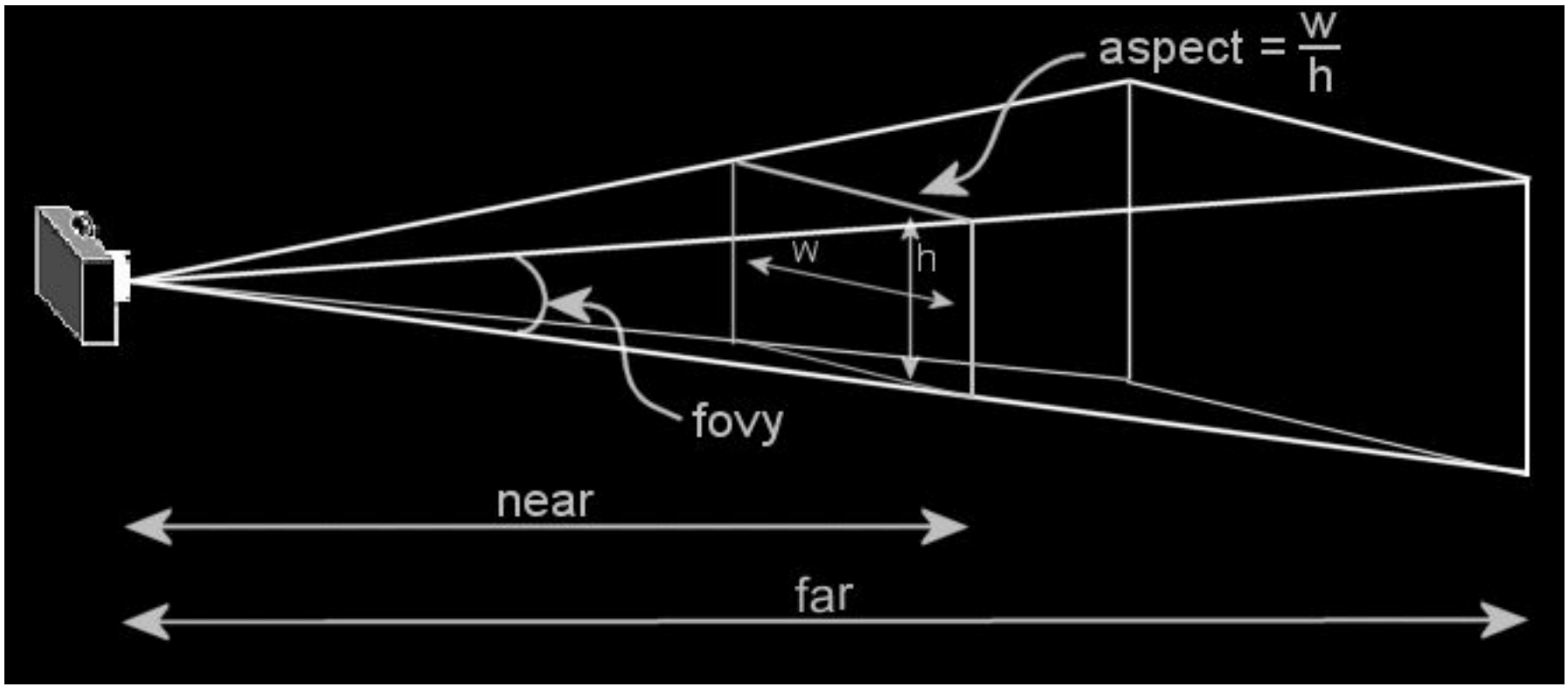
```
void setup() {
  size(240, 400, A3D);
  fill(204);
}

void draw() {
  lights();
  background(0);
  camera(30.0, mouseY, 220.0,
         0.0, 0.0, 0.0,
         0.0, 1.0, 0.0);
  noStroke();
  box(90);
  stroke(255);
  line(-100, 0, 0, 100, 0, 0);
  line(0, -100, 0, 0, 100, 0);
  line(0, 0, -100, 0, 0, 100);
}
```



Perspective view

The viewing volume is a truncated pyramid, and the convergence of the lines towards the eye point create a perspective projection where objects located farther away from the eye appear smaller.

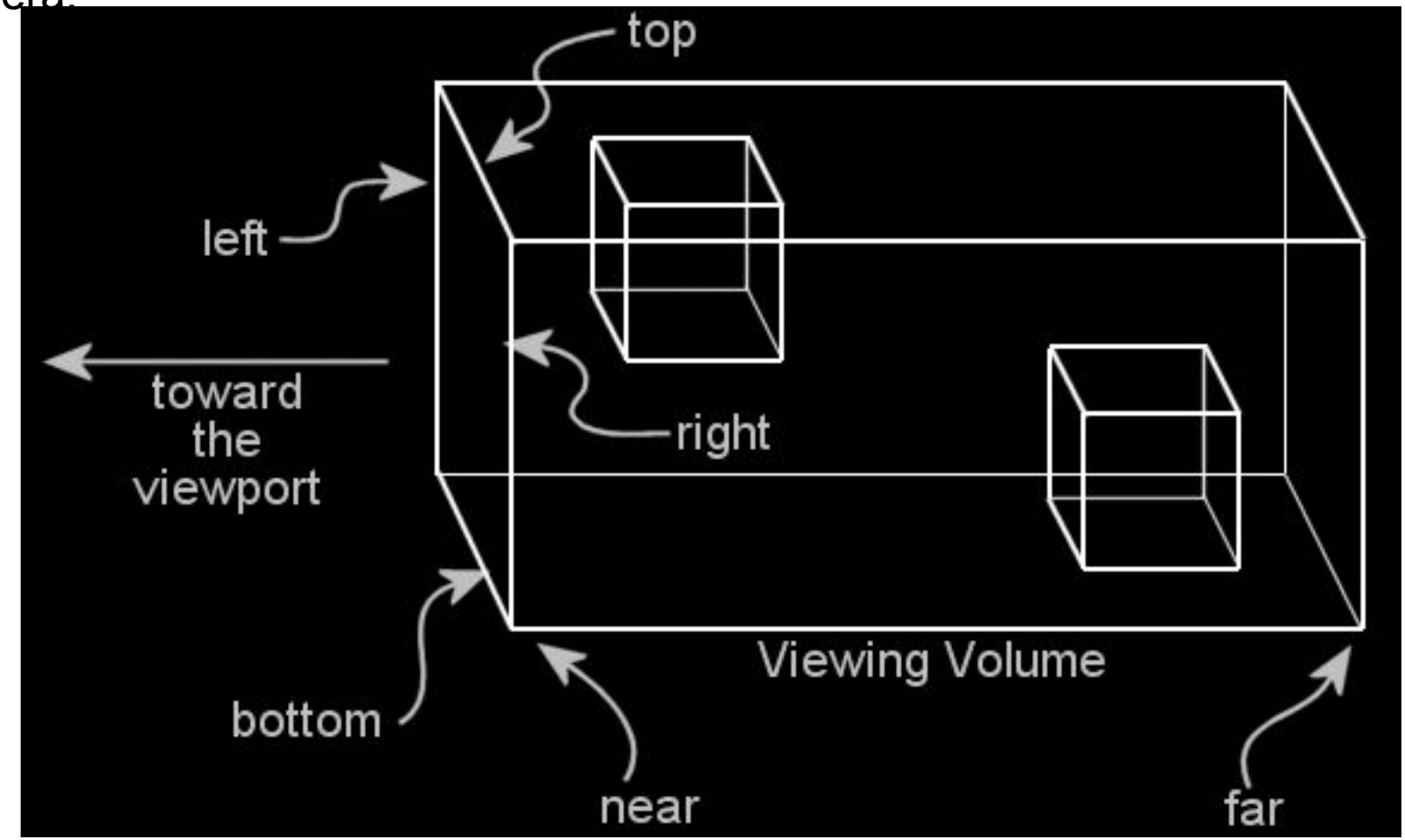
from http://jerome.jouvie.free.fr/OpenGl/Lesson



The default values for perspective used by A3D are:
perspective(PI/3.0, width/height, cameraZ/10.0, cameraZ*10.0) where cameraZ is ((height/2.0) / tan(PI*60.0/360.0))

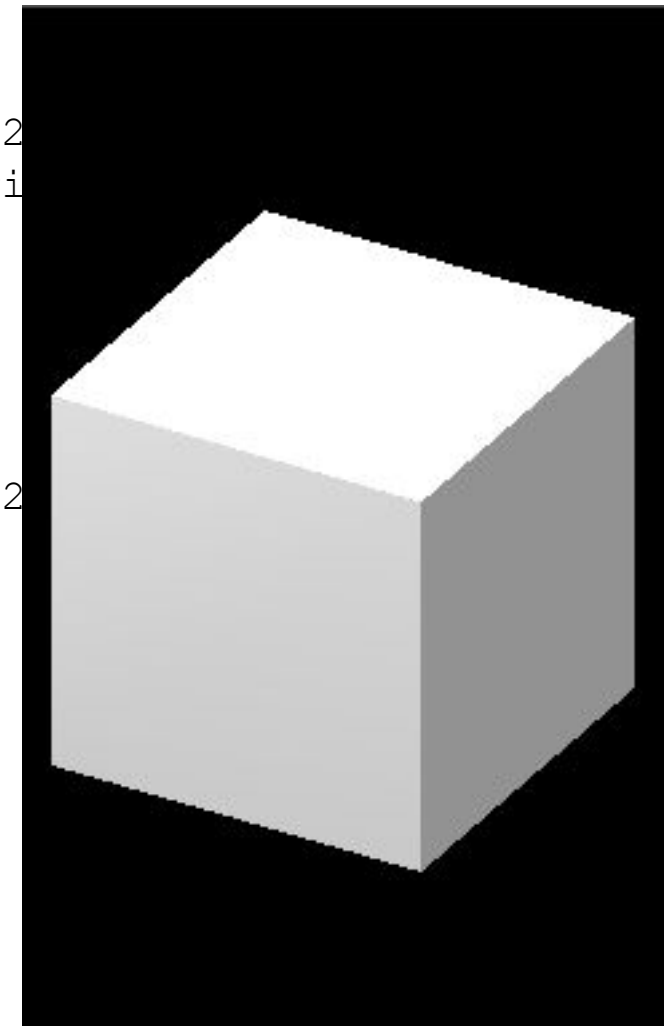
Orthographic view

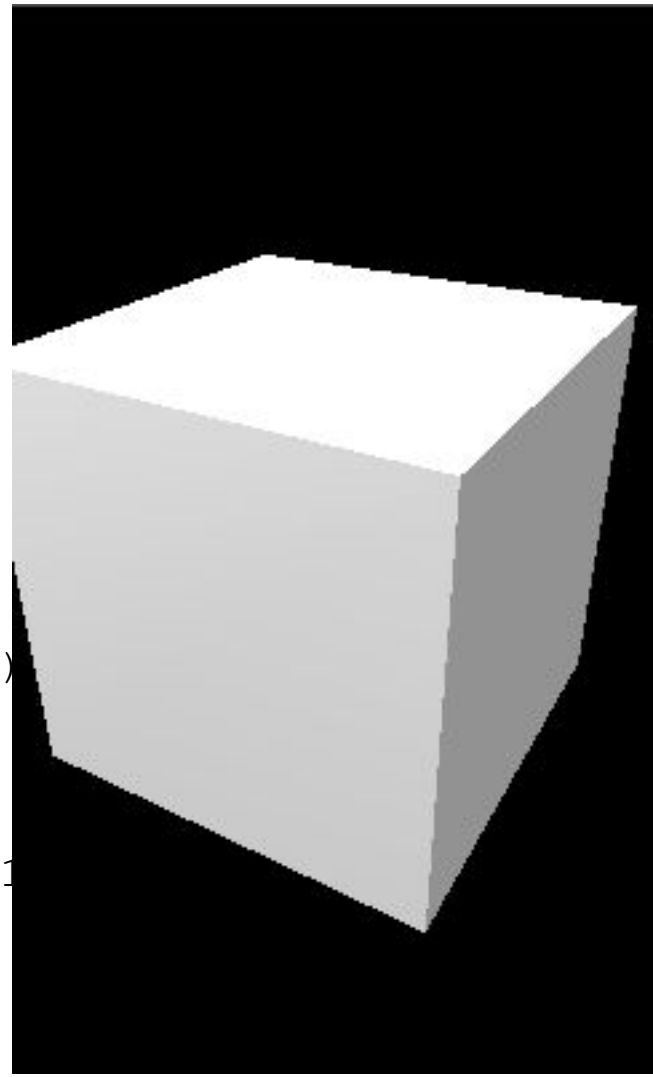
In this case the viewing volume is a parallelepiped. All objects with the same dimension appear the same size, regardless of whether they are near or far from the camera.



ortho(left, right, bottom, top, near, far)ortho(0, width, 0, height, -10, 10)

```
void setup() {
  size(240, 400, A3D);
  noStroke();
  fill(204);
}

void draw() {
  background(0);
  lights();

  if(mousePressed) {
    float fov = PI/3.0;
    float cameraZ = (height/2
    perspective(fov, float(wi
                cameraZ/2.0,
  } else {
    ortho(-width/2, width/2,
  }

  translate(width/2, height/2
  rotateX(-PI/6);
  rotateY(PI/3);
  box(160);
}
```

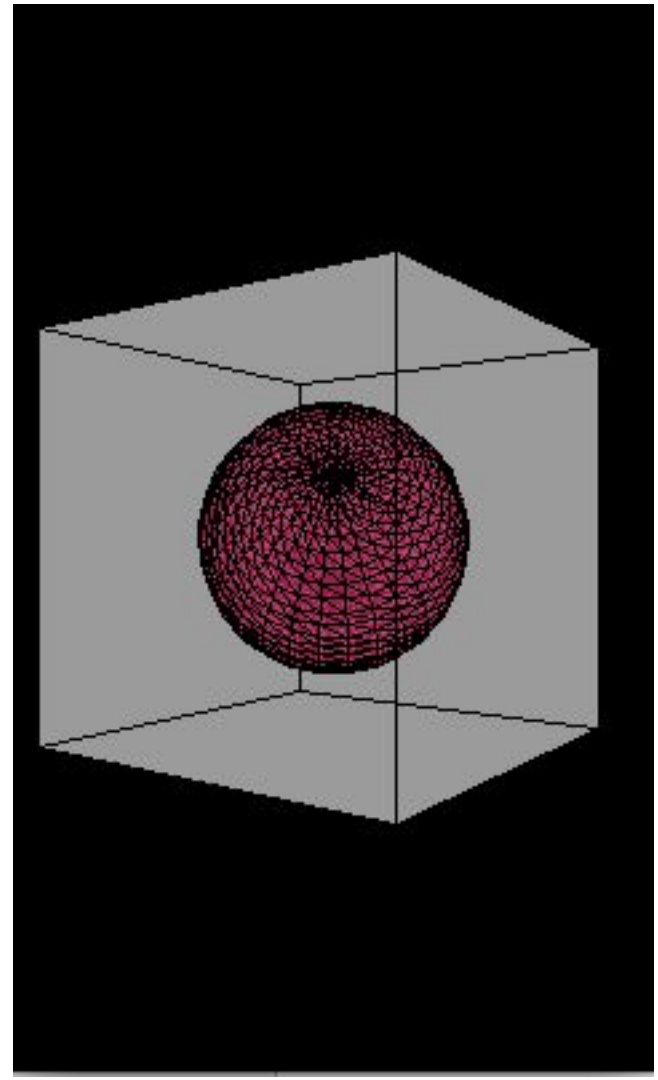




Creating 3D objects (box, spheres, vertex shapes)

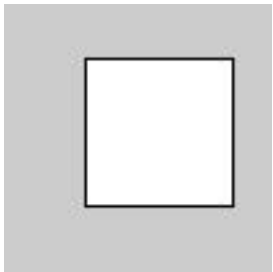
A3D provides some functions for drawing predefined 3D primitives: sphere(r), box(w, h, d)

```
void setup() {
  size(240, 400, A3D);
  stroke(0);
}

void draw() {
  background(0);

  translate(width/2, height/2, 0);

  fill(200, 200);
  pushMatrix();
  rotateY(frameCount*PI/185);
  box(150, 150, 150);
  popMatrix();

  fill(200, 40, 100, 200);
  pushMatrix();
  rotateX(-frameCount*PI/200);
  sphere(50);
  popMatrix();
}
```



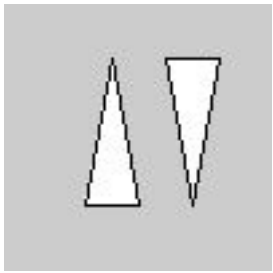
beginShape()/endShape()

The beginShape()/endShape() functions allow us to create complex objects by specifying the vertices and their connectivity (and optionally the normals and textures coordinates for each vertex)
This functionality is already present in A2D, with the difference that in A3D we can specify vertices in 3D space.



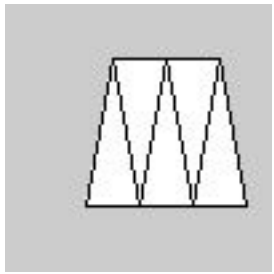
```
beginShape();
vertex(30, 20, 0);
vertex(85, 20, 0);
vertex(85, 75, 0);
vertex(30, 75, 0);
endShape(CLOSE);
```

Closed polygon



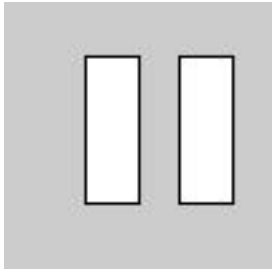
```
beginShape(TRIANGLES);
vertex(30, 75, 0);
vertex(40, 20, 0);
vertex(50, 75, 0);
vertex(60, 20, 0);
vertex(70, 75, 0);
vertex(80, 20, 0);
endShape();
```

Individual triangles



```
beginShape(TRIANGLE_STRIP);
vertex(30, 75, 0);
vertex(40, 20, 0);
vertex(50, 75, 0);
vertex(60, 20, 0);
vertex(70, 75, 0);
vertex(80, 20, 0);
vertex(90, 75, 0);
endShape();
```

Triangle strip



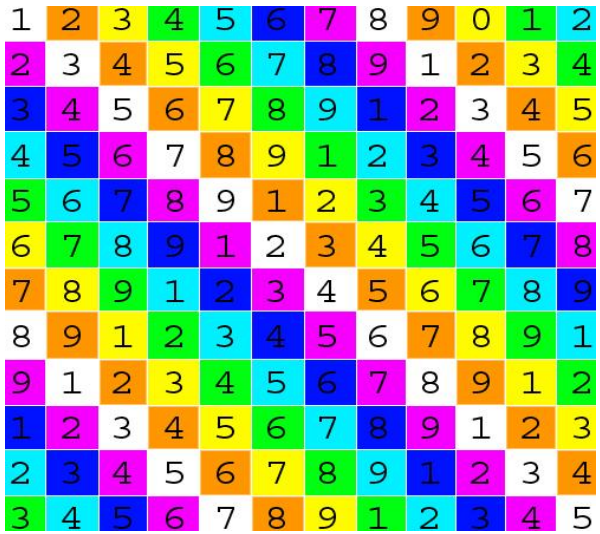
```
beginShape(QUADS);
vertex(30, 20, 0);
vertex(30, 75, 0);
vertex(50, 75, 0);
vertex(50, 20, 0);
vertex(65, 20, 0);
vertex(65, 75, 0);
vertex(85, 75, 0);
vertex(85, 20, 0);
endShape();
```

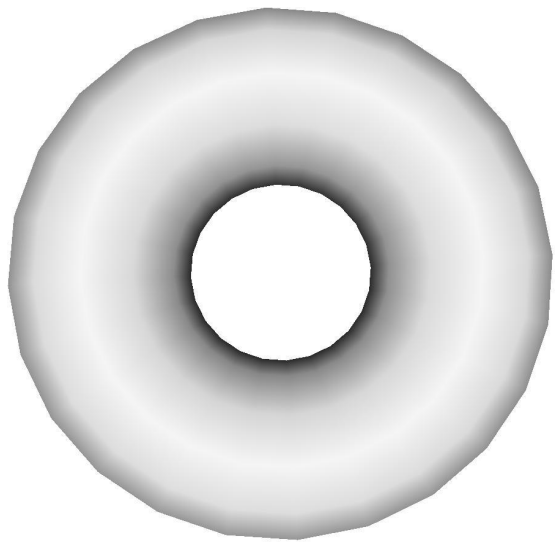
Individual quads

Check the Processing reference for more details:
http://processing.org/reference/beginShape_.html

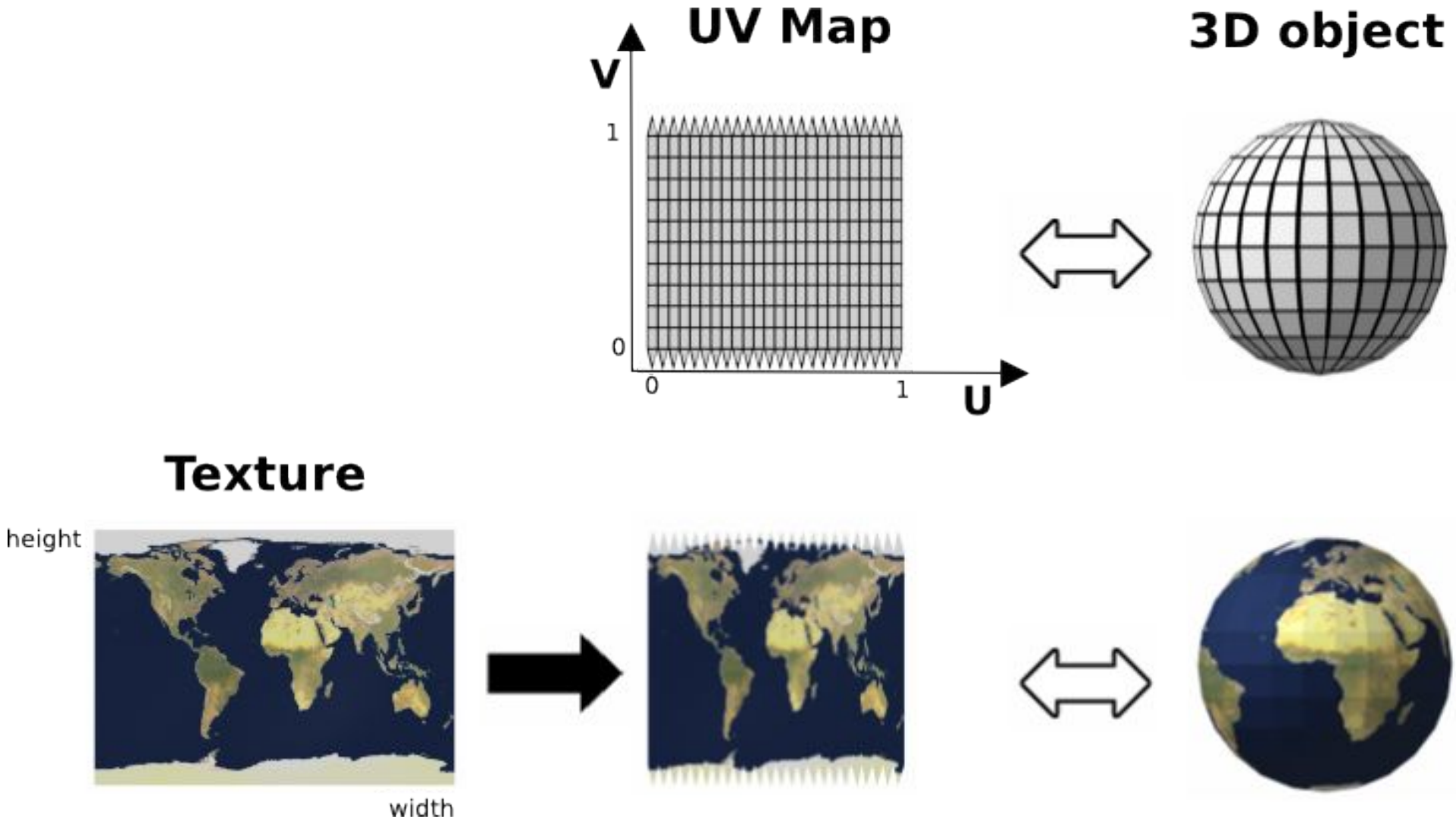
Texturing

Texturing is an important technique in computer graphics consisting in using an image to “wrap” a 3D object in order to simulate a specific material, realistic "skin", illumination effects, etc.





Basic texture mapping:



Adapted from wikipedia.org, UV mapping: http://en.wikipedia.org/wiki/UV_mapping

Texture mapping becomes a very complex problem when we need to texture complicated tridimensional shapes (organic forms).
Finding the correct mapping from 2D image to 3D shape requires mathematical techniques that takes into account edges, folds, etc.

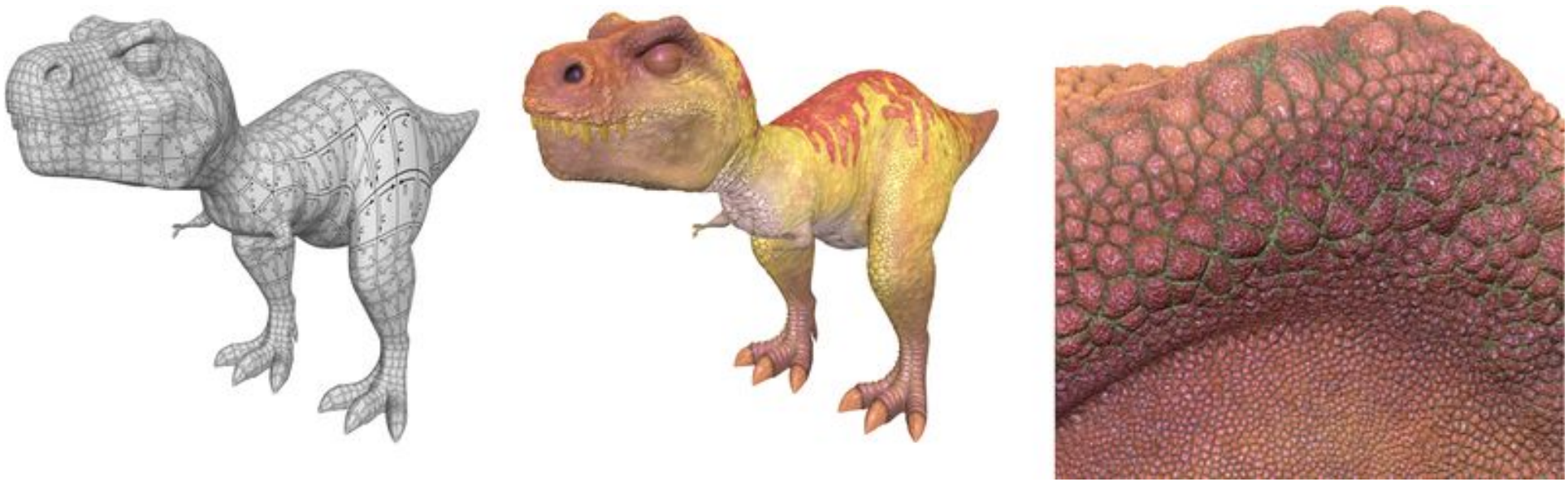


Image from: Ptex: Per-Face Texture Mapping for Production Rendering, by Brent Burley and Dylan Lacewell

Simple shape texturing

Objects created with beginShape()/endShape() can be textured using any image loaded into Processing with the loadImage() function or created procedurally by manipulating the pixels individually.

```
PImage img;

void setup() {
  size(240, 240, A3D);
  img = loadImage("beach.jpg");
  textureMode(NORMAL);
}

void draw() {
  background(0);
  beginShape(QUADS);
  texture(img);
  vertex(0, 0, 0, 0, 0);
  vertex(width, 0, 0, 1, 0);
  vertex(width, height, 0, 1, 1);
  vertex(0, height, 0, 0, 1);
  endShape();
}
```

The texture mode can be NORMAL or IMAGE

Depending on the texture mode, we use normalized UV values or relative to the image resolution.

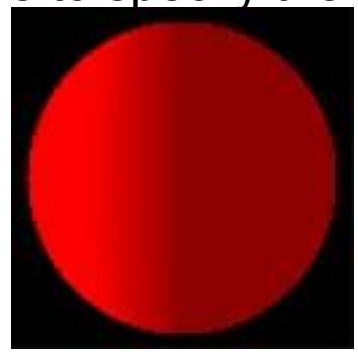
beginShape/endShape in A3D supports setting more than one texture for different parts of the shape:

```
PImage img1, img2;

void setup() {
  size(240, 240, A3D);
  img1 = loadImage("beach.jpg");
  img2 = loadImage("peebles.jpg");
  textureMode(NORMAL);
  noStroke();
}

void draw() {
  background(0);
  beginShape(TRIANGLES);
  texture(img1);
  vertex(0, 0, 0, 0, 0);
  vertex(width, 0, 0, 1, 0);
  vertex(0, height, 0, 0, 1);
  texture(img2);
  vertex(width, 0, 0, 1, 0);
  vertex(width, height, 0, 1, 1);
  vertex(0, height, 0, 0, 1);
  endShape();
}
```

Lighting

A3D offers a local illumination model based on OpenGL's model.
It is a simple real-time illumination model, where each light source has 4 components:
ambient + diffuse + specular + emissive = total
This model doesn't allow the creation of shadows
We can define up to 8 light sources.
Proper lighting calculations require to specify the normals of an object

Ambient



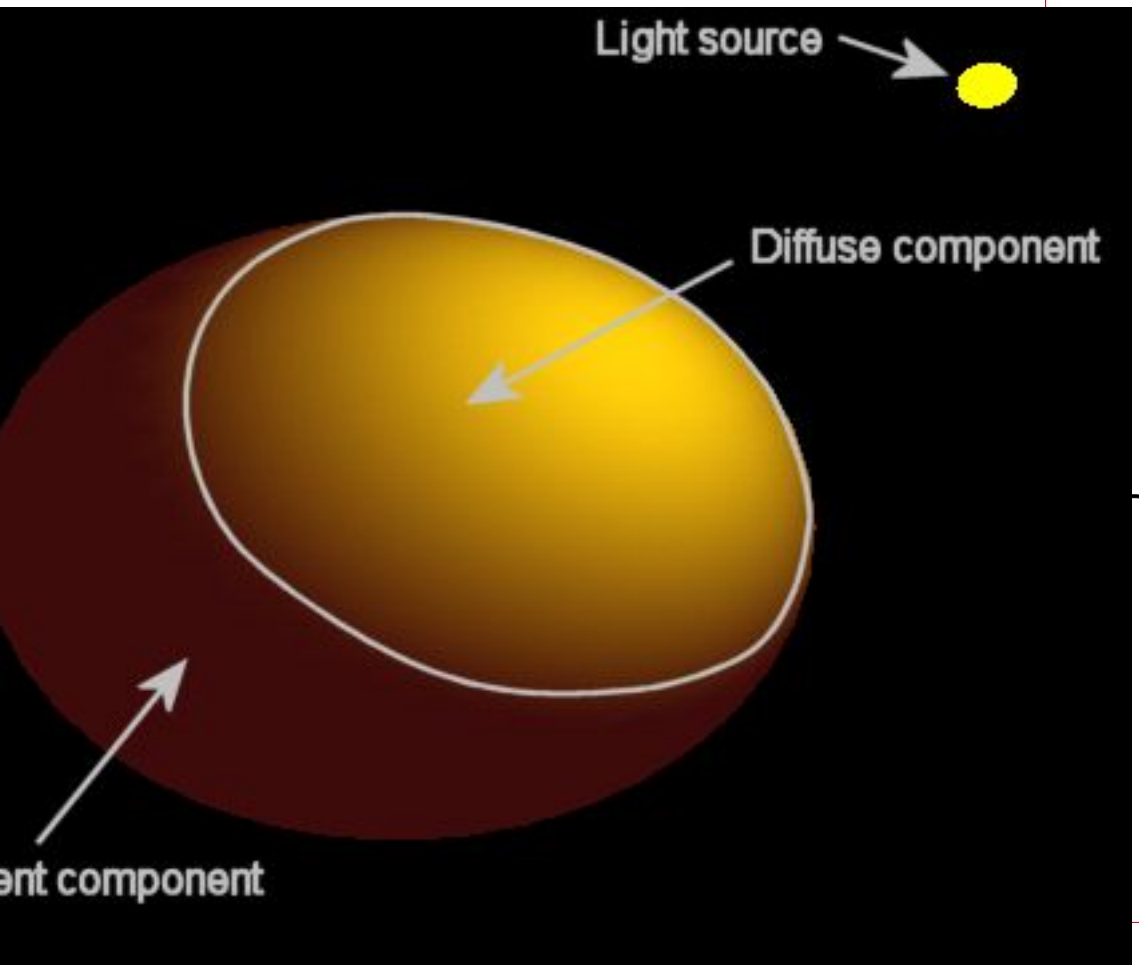
Diffuse

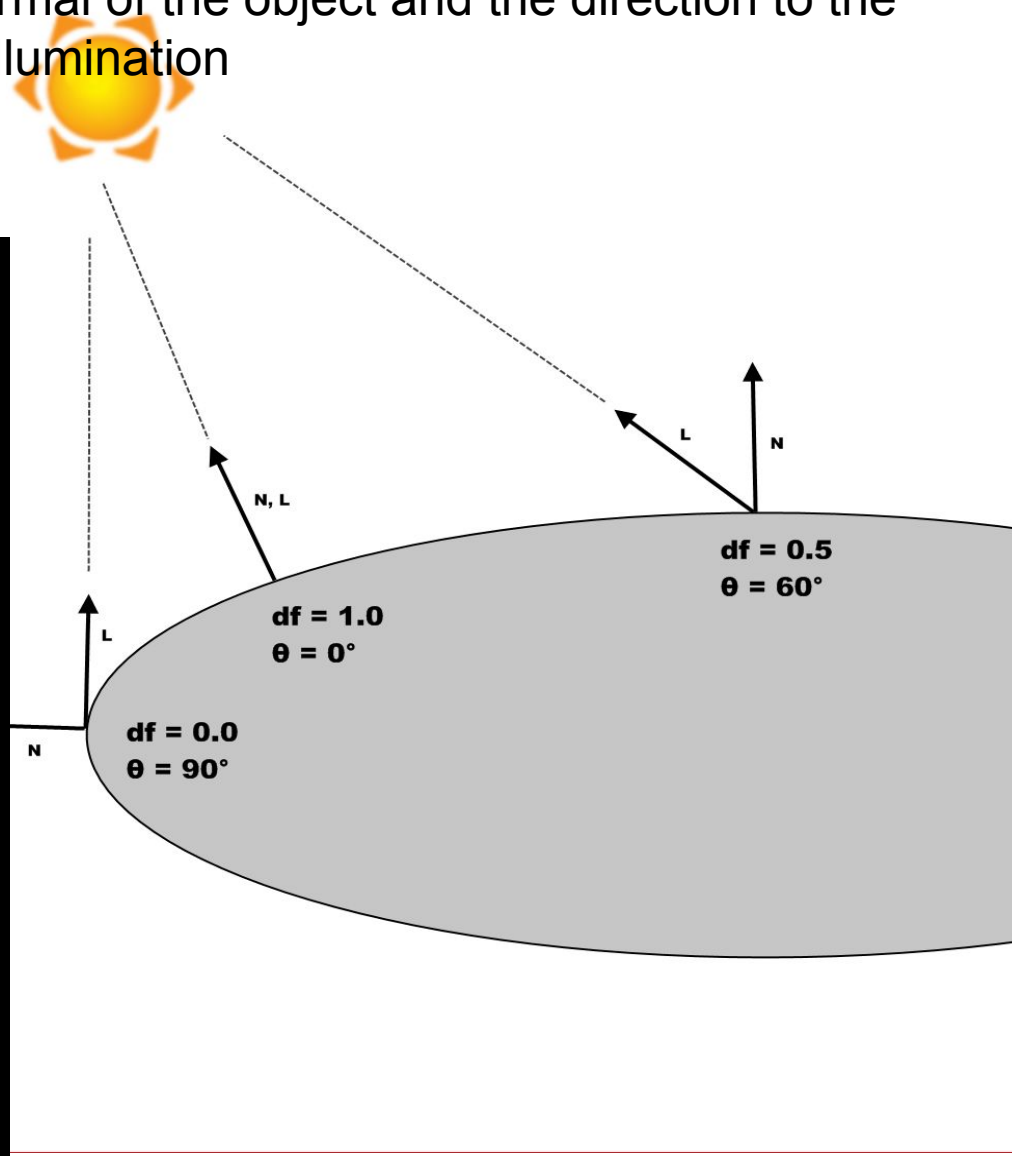
Specular

From http://www.falloutsoftware.com/tutorials/gl/gl8.htm

Some more good resources about lights in OpenGL:
http://jerome.jouvie.free.fr/OpenGl/Lessons/Lesson6.php
http://jerome.jouvie.free.fr/OpenGl/Tutorials/Tutorial12.php - Tutorial15.php
http://www.sjbaker.org/steve/omniv/opengl_lighting.html

In diffuse lighting, the angle between the normal of the object and the direction to the light source determines the intensity of the illumination





From iPhone 3D programming, by Philip Rideout. http://iphone-3d-programming.labs.oreilly.com/ch04.html

Light types in A3D

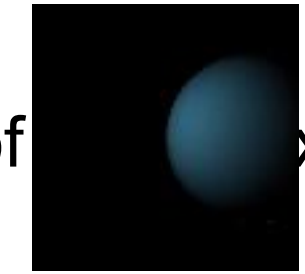
Ambient: Ambient light doesn't come from a specific direction, the rays have light have bounced around so much that objects are evenly lit from all sides. Ambient lights are almost always used in combination with other types of lights.

ambientLight(v1, v2, v3, x, y, z)v1, v2, v3: rgb color of the lightx, y, z position:

Directional: Directional light comes from one direction and is stronger when hitting a surface squarely and weaker if it hits at a a gentle angle. After hitting a surface, a directional lights scatters in all directions.

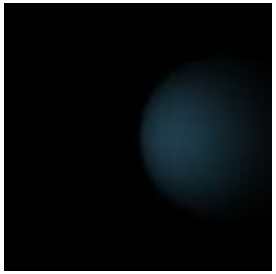
directionalLight(v1, v2, v3, nx, ny, nz)v1, v2, v3: rgb c
the light**nx**, **ny**, and **nz** parameters specify the direction the light is facing.

Point: Point light irradiates from a specific position.

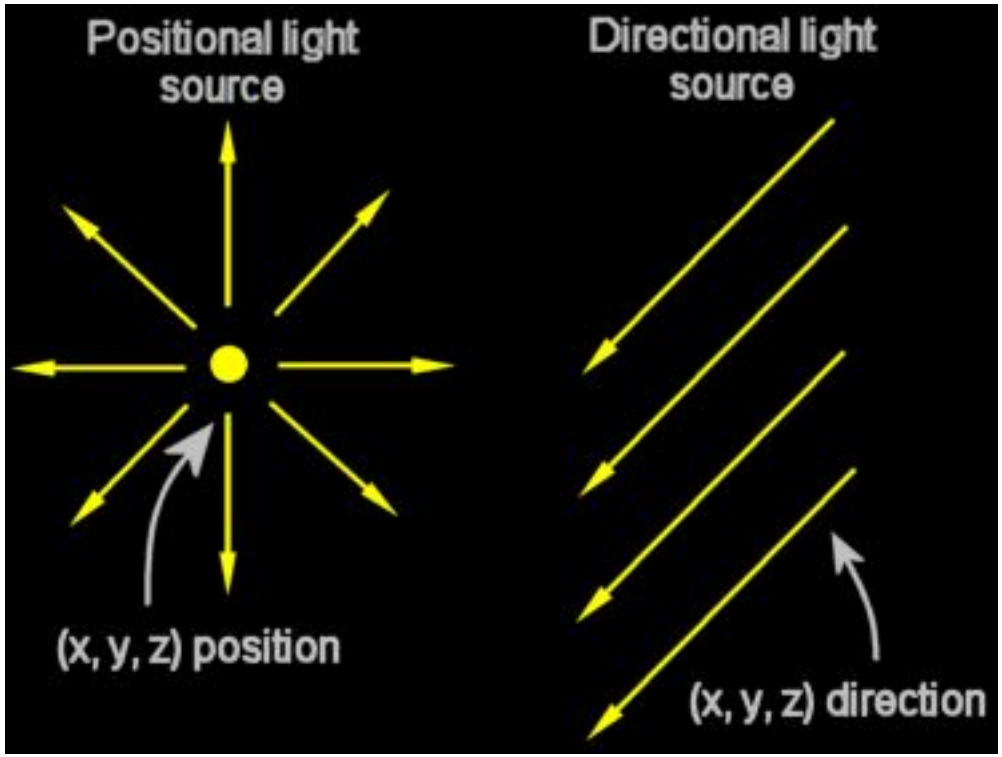


pointLight(v1, v2, v3, x, y, z)v1, v2, v3: rgb color of x, y, z position:

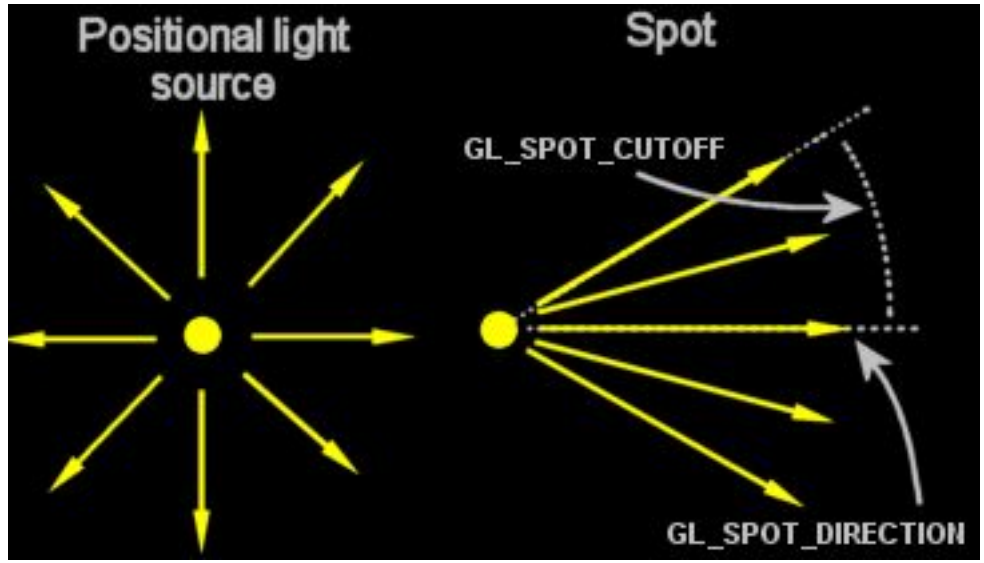
Spot: A spot light emits lights into an emission cone by restricting the emission light source.



spotLight(v1, v2, v3, x, y, z, nx, ny, nz, angle, concentration)v1, v2, v3: rgb color of the lightx, y, z position:**nx**, **ny**, **nz** specify the direction or light angle float: angle the spotlight cone concentration: exponent determining the center bias of the cone

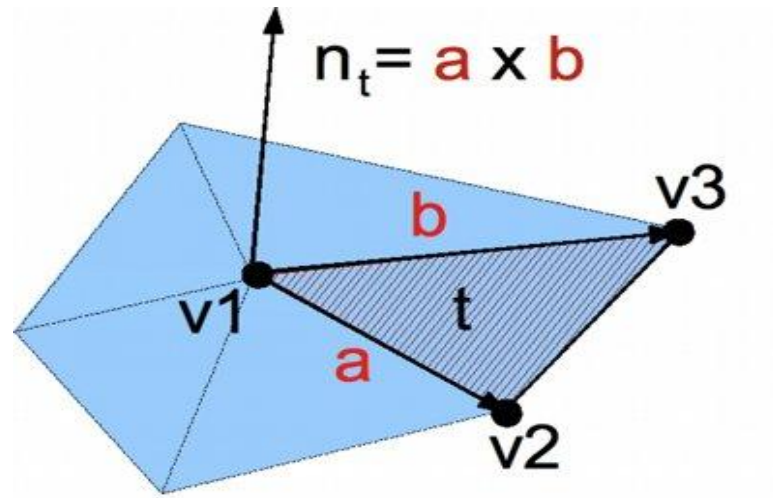


http://jerome.jouvie.free.fr/OpenGl/Lessons/Lesson6.p

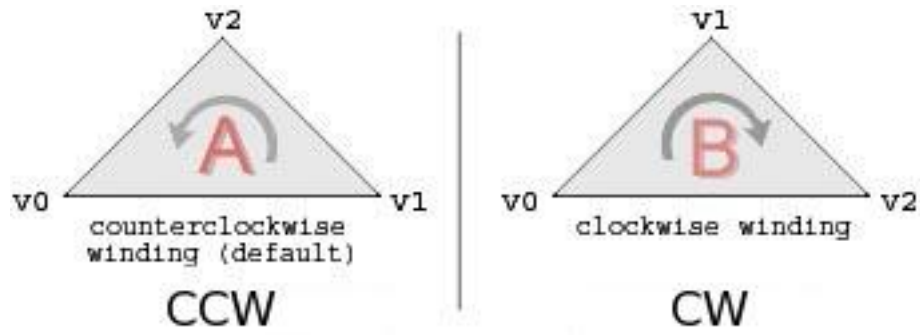


Normals: each vertex needs to have a normal defined so the light calculations can be performed correctly

```
PVector a = PVector.sub(v2, v1);
PVector b = PVector.sub(v3, v1);
PVector n = a.cross(b);
normal(n.x, n.y, n.z);

vertex(v1.x, v1.y, v1.z);
vertex(v2.x, v2.y, v2.z);
vertex(v3.x, v3.y, v3.z);
```



Polygon winding: The ordering of the vertices that define a face determine which side is inside and which one is outside. Processing uses CCW ordering of the vertices, and the normals we provide to it must be consistent with this.



Text in A3D

Text in A3D works exactly the same as in A2D.
We load/create fonts with loadFont/createFont
We set current font with textFont
We write text using the text() function

```
PFont fontA;
void setup()  {
  size(240, 400, A3D);
  background(102);
  String[] fonts = PFont.list();
  fontA = createFont(fonts[0], 32);
  textFont(fontA, 32);
}

void draw() {
  fill(0);
  text("An", 10, 60);
  fill(51);
  text("droid", 10, 95);
  fill(204);
  text("in", 10, 130);
  fill(255);
  text("A3D", 10, 165);
}
```

3D Text

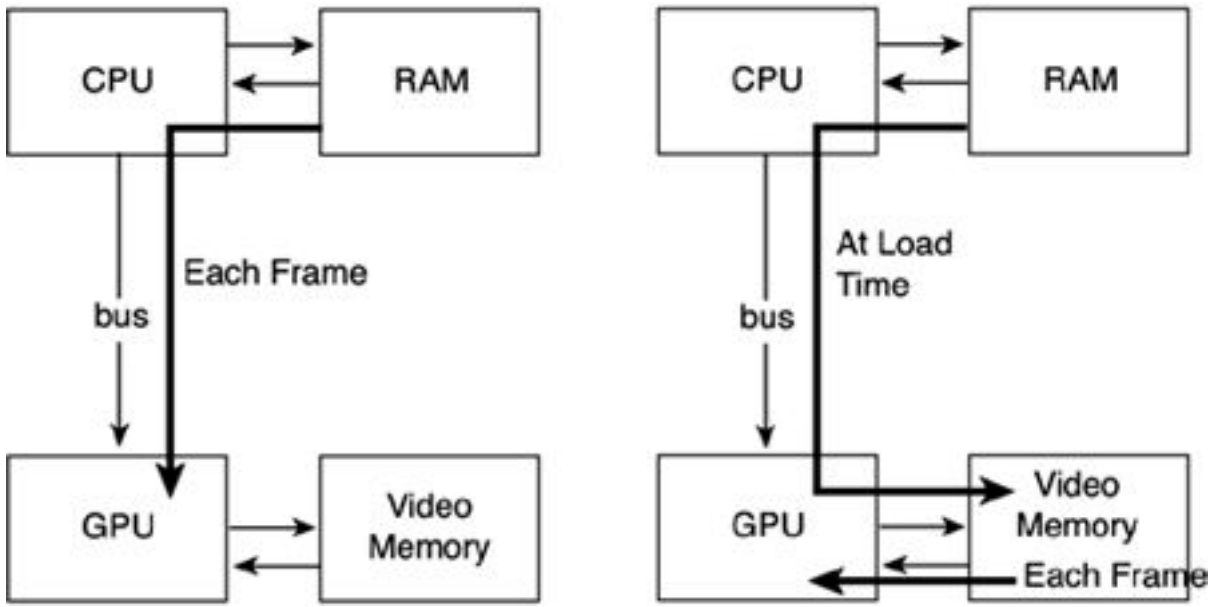
The main addition in A3D is that text can be manipulated in three dimensions. Each string of text we print to the screen with text() is contained in a rectangle that we can rotate, translate, scale, etc.
The rendering of text is also very efficient because is accelerated by the GPU (A3D internally uses OpenGL textures to store the font characters)

```
fill(0);
pushMatrix();
translate(rPos,10+25);
char k;
for(int i = 0;i < buff.length(); i++) {
  k = buff.charAt(i);
  translate(-textWidth(k),0);
  rotateY(-textWidth(k)/70.0);
  rotateX(textWidth(k)/70.0);
  scale(1.1);
  text(k,0,0);
}
popMatrix();
```

```
PFont font;
char[] sentence = { 'S', 'p', 'A' , 'O', '5', 'Q',
                    'S', 'p', 'A' , 'O', '5', 'Q',
                    'S', 'p', 'A' , 'O', '5', 'Q',
                    'S', 'p', 'A' , 'O', '5', 'Q' };

void setup() {
  size(240, 400, P3D);
  font = loadFont("Ziggurat-HTF-Black-32.vlw");
  textFont(font, 32);
}

void draw() {
  background(0);

  translate(width/2, height/2, 0);

  for (int i = 0; i < 24; i++) {
    rotateY(TWO_PI / 24 + frameCount * PI / 5000);
    pushMatrix();
    translate(100, 0, 0);
    //box(10, 50, 10);
    text(sentence[i], 0, 0);
    popMatrix();
  }
}
```

Models: Introduction to VBOs

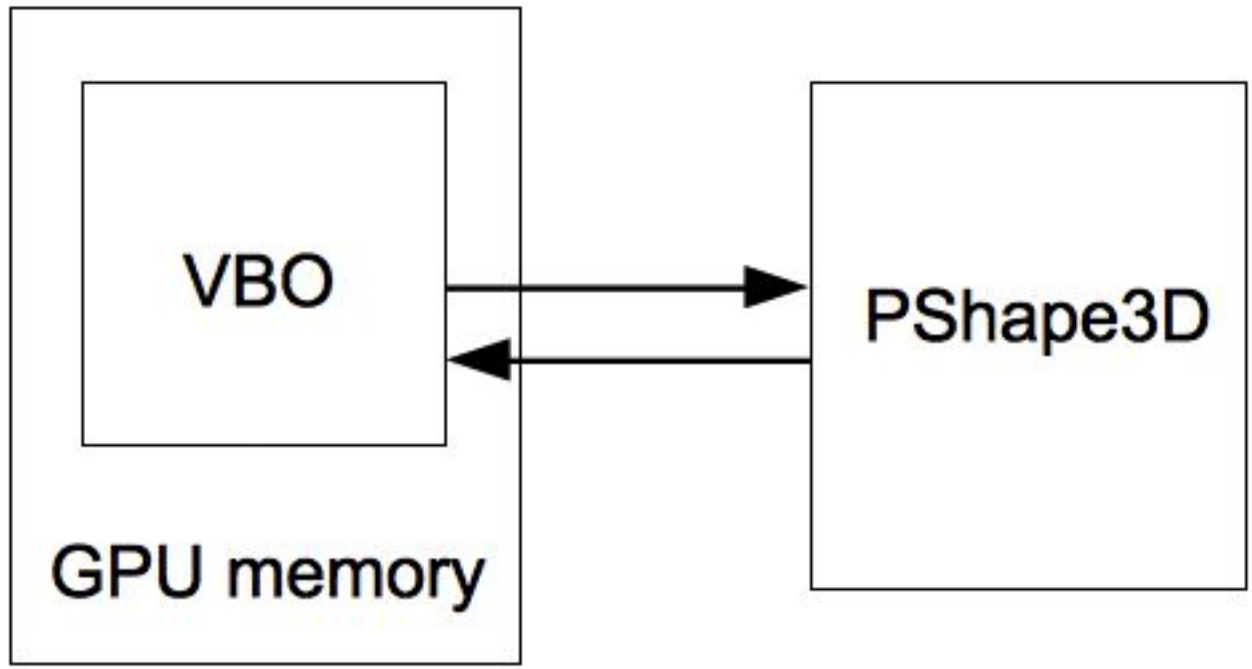
*Normally, the data that defines a 3D object (vertices, colors, normals, texture coordinates) are sent to the GPU at every frame.
* The current GPUs in mobile devices have limited bandwidth, so data transfers can be slow.
* If the geometry doesn't change (often) we can use Vertex Buffer Objects.
* A Vertex Buffer Object is a piece of GPU memory where we can upload the data defining an object (vertices, colors, etc.)
* The upload (slow) occurs only once, and once the VBO is stored in GPU memory, we can draw it without uploading it again.
* This is similar to the concept of Textures (upload once, use multiple times).



http://www.songho.ca/opengl/gl_vbo.html

The PShape3D class in A3D encapsulates VBOs

*The class Pshape3D in A3D encapsulates a VBO and provides a simple way to create and handle VBO data, including updates, data fetches, texturing, loading from OBJ files, etc.
* A PShape3D has to be created with the total number of vertices know beforehand. Resizing is possible, but slow.
* How vertices are interpreted depends on the geometry type specified at creation (POINT, TRIANGLES, etc), in a similar way to beginShape()/endShape()
* Vertices in a PShape3D can be divided in groups, to facilitate normal and color assignment, texturing and creation of complex shapes with multiple geometry types (line, triangles, etc).



Manual creation of Pshape3D models

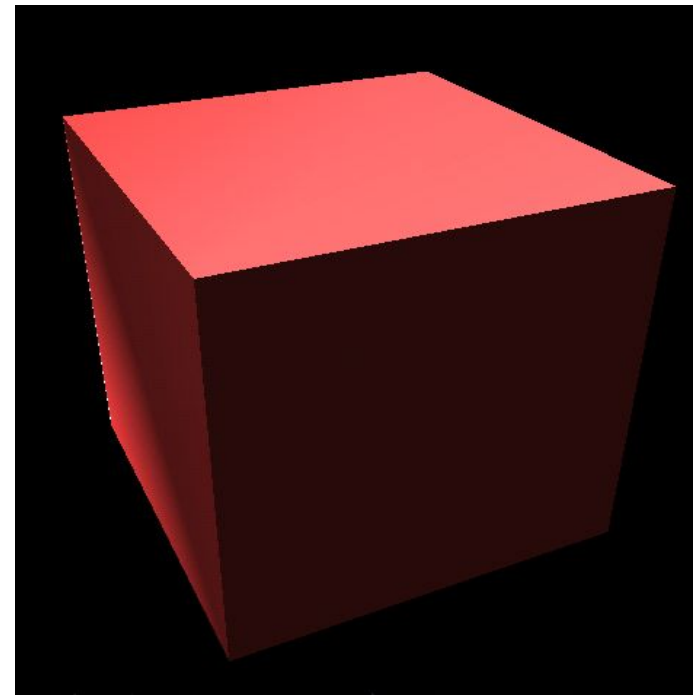
A PShape3D can be created by specifying each vertex and associated data (normal, color, etc) manually.
Remember that the normal specification must be consisten with the CCW vertex ordering.

Creation

```
cube = createShape(36, TRIANGLES);

cube.beginUpdate(VERTICES);
cube.setVertex(0, -100, +100, -100);
cube.setVertex(1, -100, -100, -100);
...
cube.endUpdate();

cube.beginUpdate(COLORS);
cube.setVertex(0, color(200, 50, 50, 150));
cube.setVertex(1, color(200, 50, 50, 150));
...
cube.endUpdate();

cube.beginUpdate(NORMALS);
cube.setVertex(0, 0, 0, -1);
cube.setVertex(1, 0, 0, -1);
...
cube.endUpdate();
```

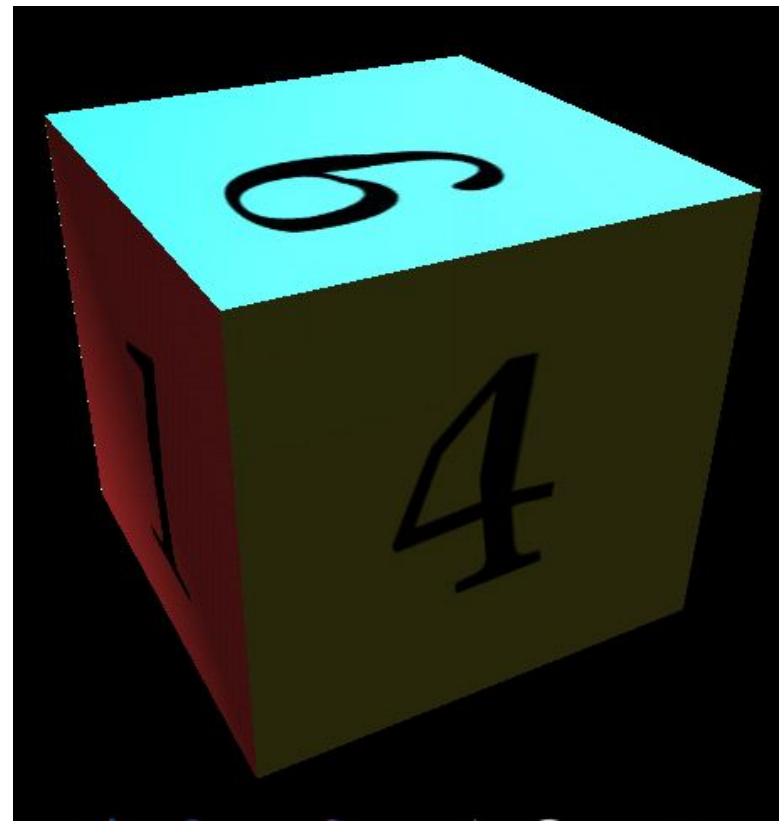
Drawing

```
translate(width/2, height/2, 0);
shape(cube);
```



A PShape3D can be textured with one ore more images.
The vertices can be organized in groups, which allows to assign a different texture to each group of vertices.
Groups also facilitate the assignment of colors and normals.

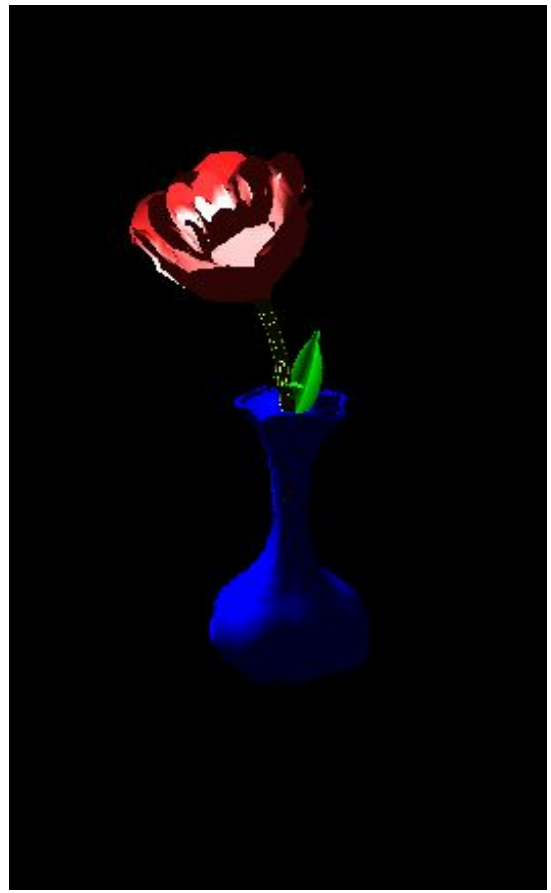
```
cube.beginUpdate(VERTICES);
cube.setGroup(0);
cube.setVertex(0, -100, +100, -100);
cube.setVertex(1, -100, -100, -100);
...
cube.setGroup(1);
cube.setVertex(6, +100, -100, +100);
cube.setVertex(7, +100, +100, +100);
...
cube.endUpdate();

cube.setGroupColor(0, color(200, 50, 50, 150));
cube.setGroupColor(1, color(200, 50, 50, 150));
...

cube.setGroupNormal(0, 0, 0, -1);
cube.setGroupNormal(1, +1, 0, 0);
...
cube.setGroupTexture(0, face0);
cube.setGroupTexture(1, face1);
...
```



OBJ loading

The OBJ format is a text-based data format to store 3D geometries. There is an associated MTL format for materials definitions.
It is supported by many tools for 3D modelling (Blender, Google Sketchup, Maya, 3D Studio). For more onfo: http://en.wikipedia.org/wiki/Obj
A3D supports loading OBJ files into PShape3D objects with the loadShape() function.
Depending the extension of the file passed to loadShape (.svg or .obj) Processin will attempt to interpret the file as either SVG or OBJ.
The current styles active at the time of loading the shape are used to generate the geometry.

```
PShape object;
float rotX;
float rotY;

void setup() {
  size(480, 800, A3D);
 noStroke();
 object = loadShape("rose+vase.obj");
}

void draw() {
 background(0);
 ambient(250, 250, 250);
 pointLight(255, 255, 255, 0, 0, 200);

 translate(width/2, height/2, 400);
 rotateX(rotY);
 rotateY(rotX);

 shape(object);
}
```



Copying SVG shapes into PShape3D

Once we load an SVG file into a PShapeSVG object, we can copy into a PShape3D for increased performance:

```
PShape bot;
PShape3D bot3D;

public void setup() {
  size(480, 800, A3D);
  bot = loadShape("bot.svg");
  bot3D = createShape(bot);
}

public void draw() {
  background(255);

  shape(bot3D, mouseX, mouseY, 100, 100);
}
```

Shape recording

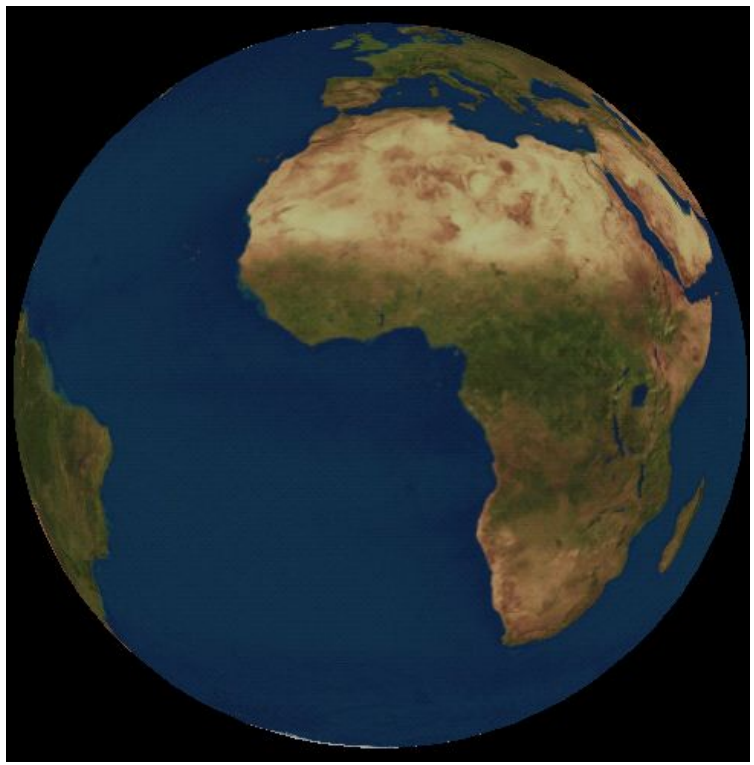
Shape recording into a PShape3D object is a feature that can greatly increase the performance of sketches that use complex geometries.
The basic idea of shape recording is to save the result of standard Processing drawing calls into a PShape3D.
There are two was of using shape recording: one is saving a single shape with beginShapeRecorder/endShapeRecorder, and the second allows saving multiple shapes with beginShapesRecorder/endShapesRecorder

```
PShape recShape;

void setup() {
  size(480, 800, A3D);

  beginShapeRecorder(QUADS);
  vertex(50, 50);
  vertex(width/2, 50);
  vertex(width/2, height/2);
  vertex(50, height/2);
  recShape = endShapeRecorder();
  ...
}

void draw() {
  ...
  shape(recShape3D);
  ...
}
```

```
objects recShape;

void setup() {
  size(480, 800, A3D);
  beginShapesRecorder();
  box(1, 1, 1);
  rect(0, 0, 1, 1);
  ...
  objects = endShapesRecorder();
  ...
}

void draw() {
  ...
  shape(objects);
  ...
}
```

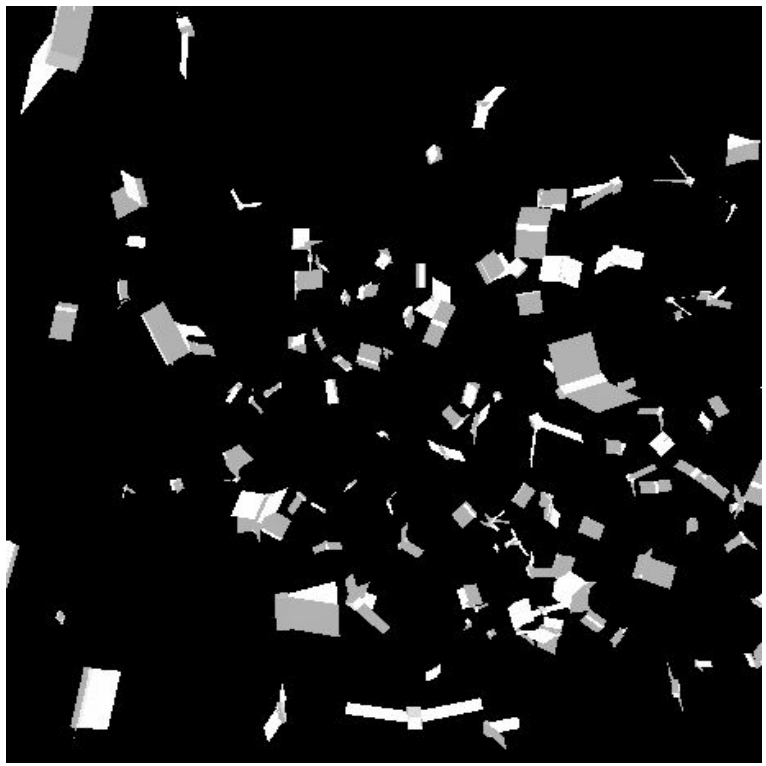
The performance gains of using shape recording are quite substantial.
It usually increases the rendering framerate by 100% or more.

Textured sphere



Without shape recording: 19fps
With shape recording: 55fps

Birds flock



Without shape recording: 7fps
With shape recording: 18fps

Particle systems

PShape3D allows to create models with the POINT_SPRITES geometry type.
With this type, each vertex is treated as a textured point sprite.
At least one texture must be attached to the model, in order to texture the sprites.
More than sprite texture can be attached, by dividing the vertices in groups.
The position and color of the vertices can be updated in the draw loop in order to simulate motion (we have to create the shape as `DYNAMIC)`.

```
particles = createShape(1000, POINT_SPRITES, DYNAMIC);
particles.beginUpdate(VERTICES);
for (int i =0; i < particles.getNumVertices(); i++) {
  float x = random(-30, 30);
  float y = random(-30, 30);
  float z = random(-30, 30);
  particles.setVertex(i, x, y, z);
}
particles.endUpdate();
sprite = loadImage("particle.png");
particles.setTexture(sprite);
```

Creation/initialization

```
particles.beginUpdate(VERTICES);
for (int i =0; i < particles.getNumVertices(); i++) {
    float[] p = particles.getVertex(i);
    p[0] += random(-1, 1);
    p[1] += random(-1, 1);
    p[2] += random(-1, 1);
    particles.setVertex(i, p);
  }
particles.endUpdate();
```

Dynamic update

Wrapping up...

Lets look an example where we do:
1) offscreen rendering to texture
2) particle systems (rendered to offscreen surface)
3) dynamic texturing of a 3D shape using the result of the offscreen drawing