

for Android \*/

```
222);  
rial-Black-13.v1w");  
25, 40);  
, 25, 60);
```

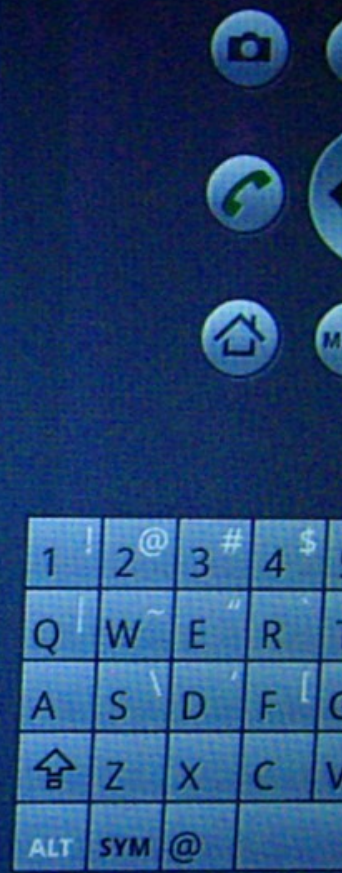
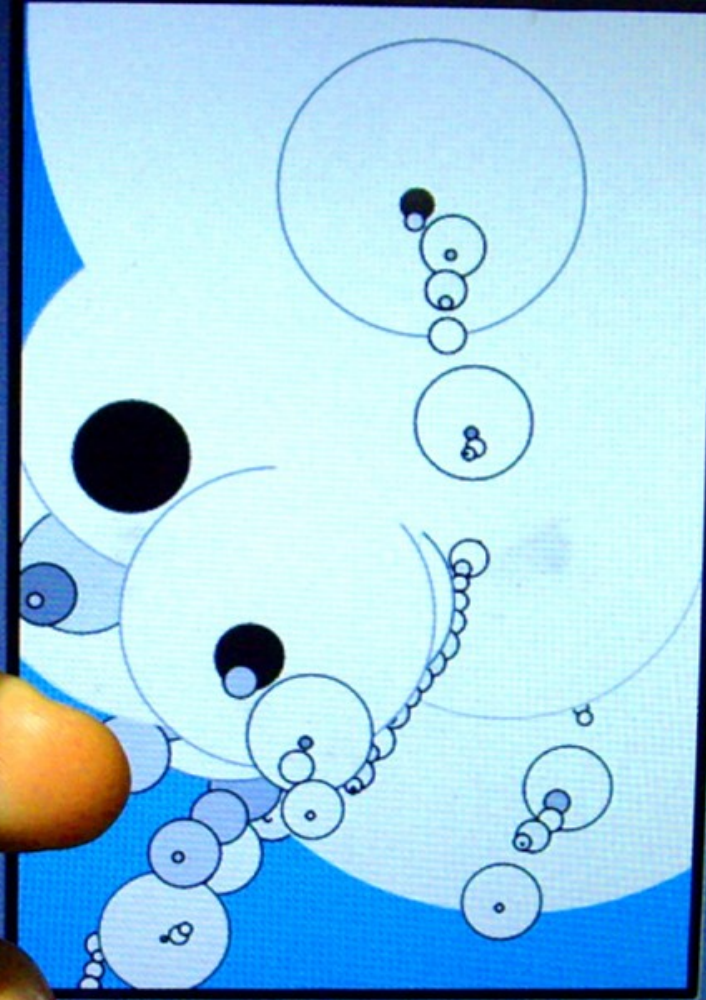
useX, mouseY, pmouse

```
(int x, int y,
```

```
(x-px) + abs
```

```
ed,
```

```
;
```



# Introduction to Processing on Android devices

# Introduction to

**Processing** 

**on Android devices** 

SIGGRAPH Asia 2010, Seoul  
Saturday, December 18<sup>th</sup> (Room E5)

**PART I: Introduction to Processing for Android**  
(14:15 - 16:00, Jihyun Kim)

**PART II: Fast 3D Graphics in Processing for Android**  
(16:15 - 18:00, Andres Colubri)

The key topics of this course consist in **getting started with Processing development on Android devices, introducing the main characteristics of Processing, running simple graphic applications and uploading them to Android devices**, and grasping the possibilities offered by more advanced features such as **OpenGL-accelerated 3D graphics**. A central objective is to provide enough basic material and motivation to the attendees of this course so they can proceed with further explorations of the Android platform and Processing language.

# CONTENTS

## **PART I: Introduction to Processing for Android**

- 1. What is Android?**
- 2. What is Processing?**
- 3. Basic concepts in Android applications**
- 4. First steps with Processing for Android**
- 5. Basic Processing use**
- 6. Extending Processing**

## **PART II: Fast 3D Graphics in Processing for Android**

- 7. OpenGL and Processing**
- 8. Geometrical transformations**
- 9. Camera and perspective**
- 10. Creating 3D objects**
- 11. 3D text**
- 12. Some special topics**
- 13. Models: the PShape3D class**

# **PART I: Introduction to Processing for Android**

# 1. What is Android?



**Android** is an open source operating system designed primarily for mobile devices, 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 maintained by the Open Handset Alliance, which is lead by Google.

More: <http://www.tbray.org/ongoing/When/201x/2010/11/14/What-Android-Is>

## What is the Dalvik VM?



It is a virtual machine to...

- run on a slow CPU
- with relatively little RAM
- on an OS without swap space

**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

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 (Android Package) file.



# Features

<http://developer.android.com/guide/basics/what-is-android.html>

- 1. Application framework:** enabling reuse and replacement of components
- 2. Dalvik virtual machine:** optimized for mobile devices
- 3. Integrated browser:** based on the open source WebKit engine
- 4. Optimized graphics:** powered by a custom 2D graphics library; 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional)
- 5. SQLite** for structured data storage
- 6. Media support** for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- 7. GSM Telephony** (hardware dependent)
- 8. Bluetooth, EDGE, 3G, and WiFi** (hardware dependent)
- 9. Camera, GPS, compass, and accelerometer** (hardware dependent)
- 10. Rich development environment** including a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE



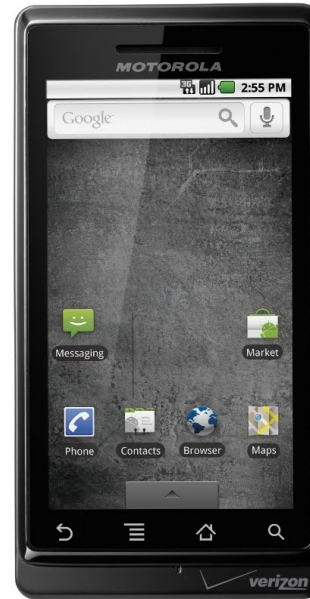
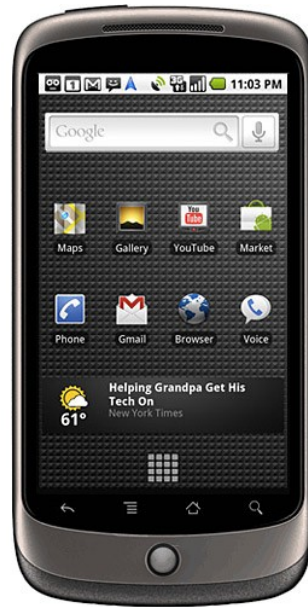
# 1.1 Android devices

The Open Handset Alliance publishes a series of hardware specifications that all the devices for Android must comply with. For Android 2.1, these are:

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

For more details, check the Android Compatibility Program:  
<http://source.android.com/compatibility/overview.html>

# 1.1 Android devices



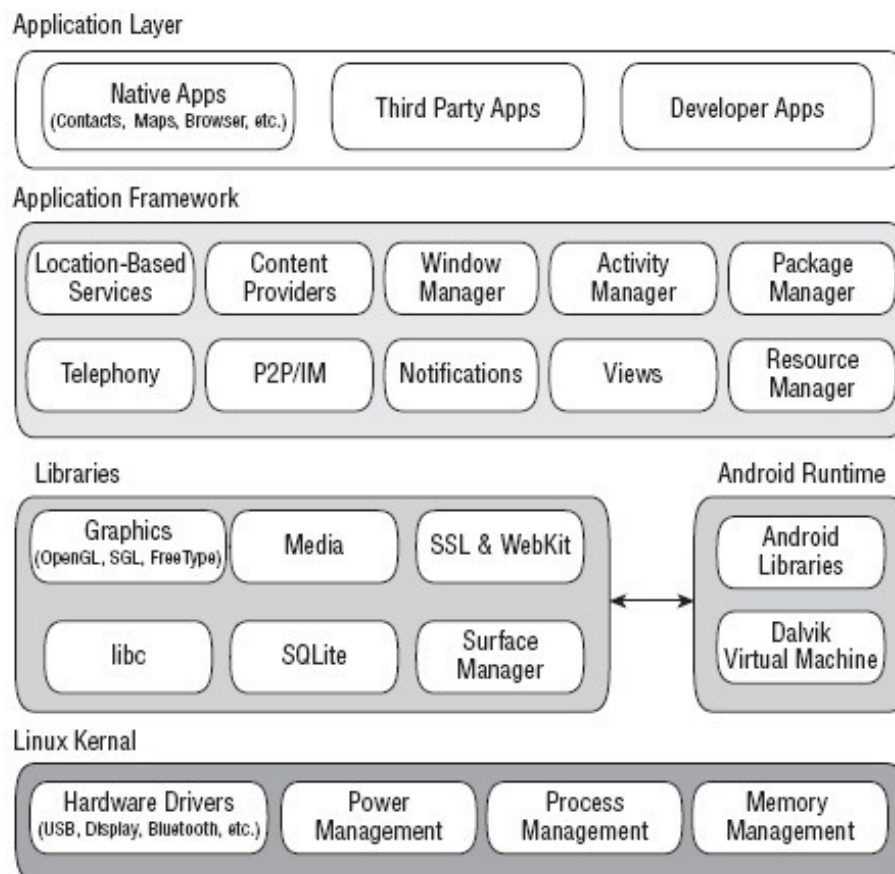
Many hardware manufacturers are producing Android handsets: HTC, Samsung, Motorola, LG, Sony Ericsson... For a comprehensive list, check this website: <http://www.androphones.com/all-android-phones.php>

## 1.2 Development process and Android Market

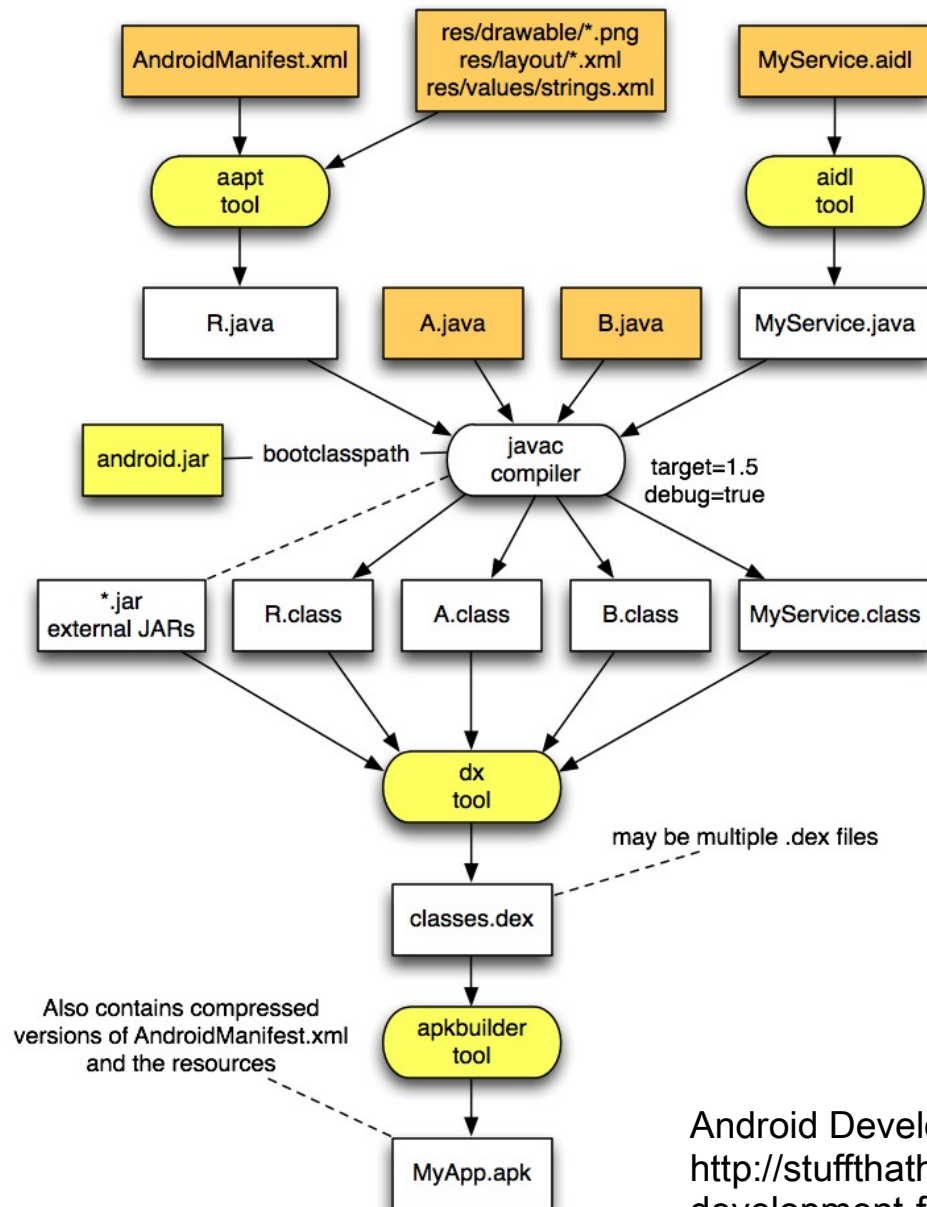
### **Basic Idea:**

1. writing code on host computer
2. using text editor/command line or Eclipse (with ADT)  
( and now Processing).
3. Testing/debugging on Emulator
4. Upload to device
5. ADB allows to debug on device

Android Software Stack



## 1.2 Development process and Android Market



Android Development Flow  
<http://stuffthathappens.com/blog/2008/11/05/android-development-flow/>

## 1.2 Development process and Android Market



CardioTrainer



cloudListPro grocery  
todo list



RunKeeper Free



On the Go



OANDA fxTrade for  
Android



Calorie Counter by  
FatSecret



VoterMap



TIME Mobile



4 Player Reactor



Taylor Swift



ElectionCaster  
(Politics)



Kindle for Android

**Android Market** is an online software store developed by Google for Android devices. An application called "Market" is preinstalled on most Android devices and allows users to browse and download apps published by third-party developers, hosted on Android Market.

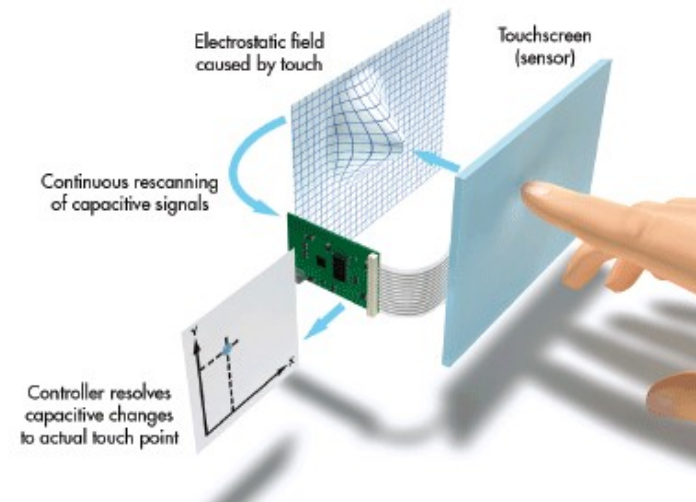
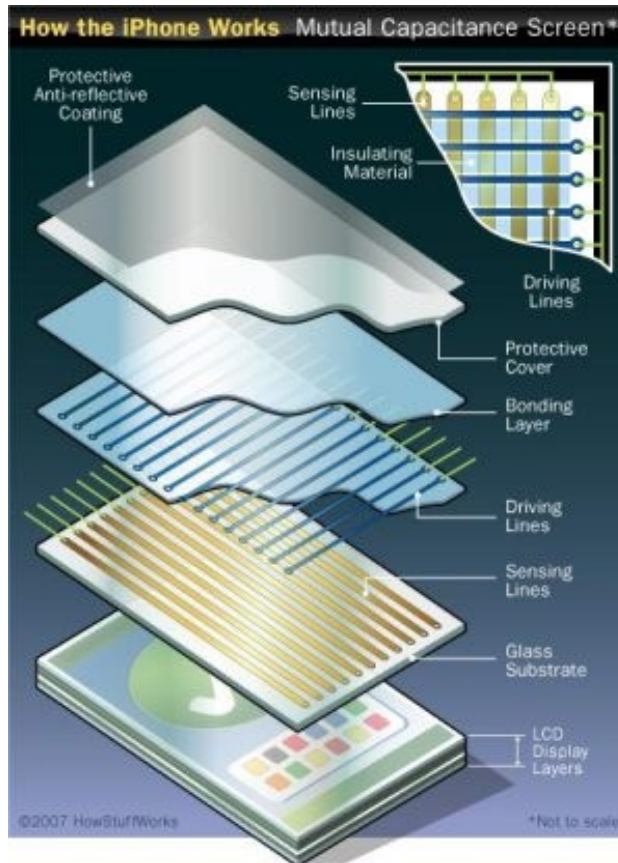
<http://www.android.com/market/>

# 1.2 Development process and Android Market

Percent of Apps on Platform by Normalized Category



# 1.3 Hardware



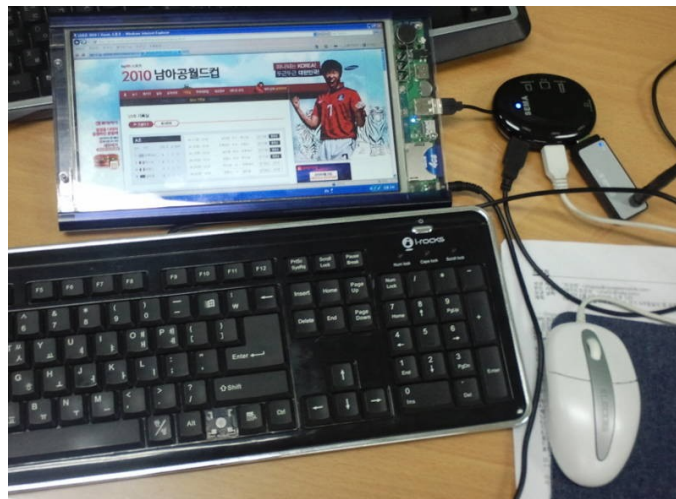
Android devices have a common basic set of hardware features such as **multitouch screen...**

Various sensors such as camera, accelerometer, GPS, magnetometer (compass), etc.





Android can be used on tablets as well.  
This format extends its applications to new  
areas of use and interaction.



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



## 2. 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.

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);
}
```

## 2.1. Main goals of Processing

Simplicity



```
ColorWheel | Processing 1.2.1

ColorWheel

/**
 * Subtractive Color Wheel
 * by Ira Greenberg.
 *
 * The primaries are red, yellow, and blue. The secondaries are green,
 * purple, and orange. The tertiaries are yellow-orange, red-orange,
 * red-purple, blue-purple, blue-green, and yellow-green.
 *
 * Create a shade or tint of the subtractive color wheel using
 * SHADE or TINT parameters.
 *
 * Updated 26 February 2010.
 */

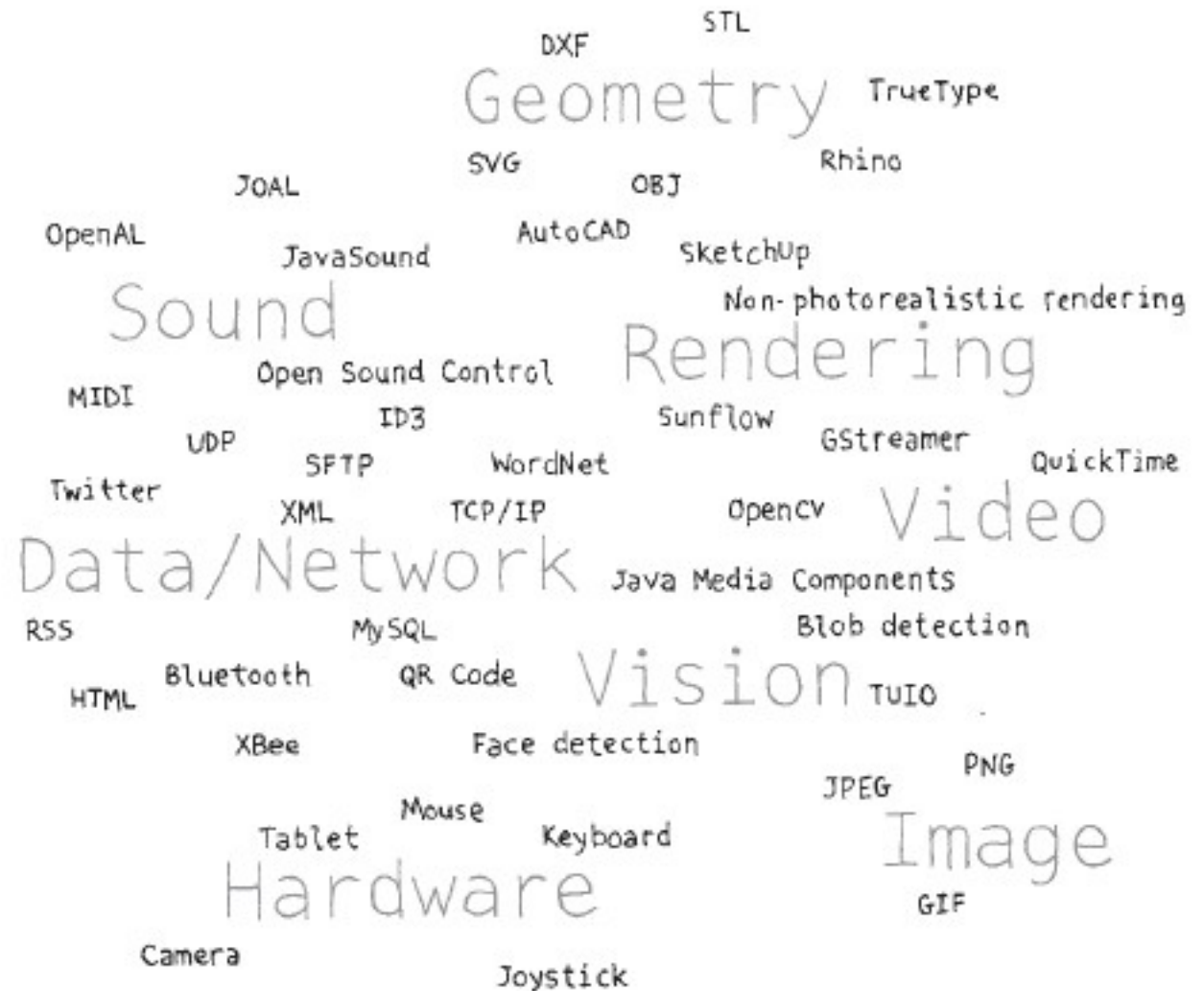
int segs = 12;
int steps = 6;
float rotAdjust = TWO_PI / segs / 2;
float radius;
float segWidth;
float interval = TWO_PI / segs;

void setup() {
  size(200, 200);
  background(127);
  smooth();
  ellipseNode(RADIUS);
}
```

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

## 2.1. Main goals of Processing

Flexibility

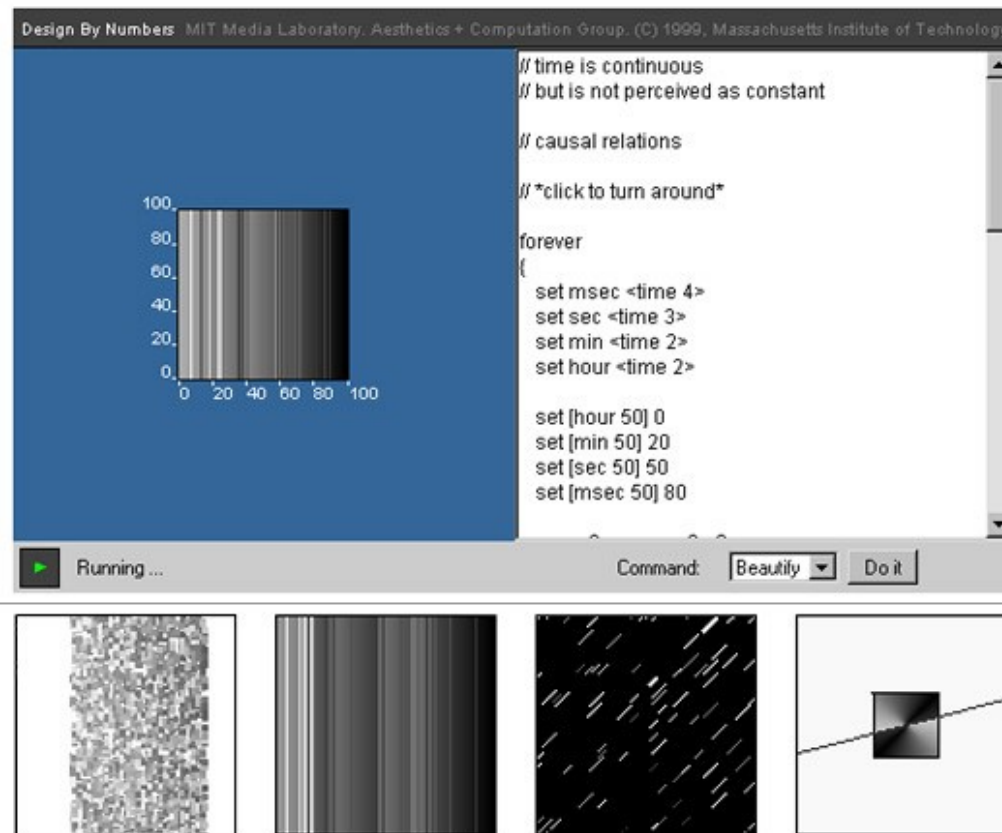


**Many types of information** can flow in and out of Processing.

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



## 2.2 Use in education and Artistic/Design Production



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).

## 2.2 Use in education and Artistic/Design Production

The screenshot displays the OpenProcessing website interface. At the top left, the 'OpenProcessing' logo is visible, along with a user status 'not logged in yet' and a 'login/register' link. Navigation links for 'home', 'browse', 'classrooms & collections', 'books', 'blog', and 'search' are located at the top. A small image of a person shouting with the text 'This guy really rocks :)' is shown in the top right, with a caption 'Upload, share and get feedback on your designs! Powered by InFuAds'.

The main content is divided into two sections: 'Classroomsbeta' and 'Collectionsbeta'. Each section contains a list of items, each with a thumbnail image, a title, author, and a count of sketches and users.

**Classroomsbeta**

- Beykent University - Computer Programming for Interaction** by Bager Akbay, 57 sketches by 29 users
- Gestalten mit Code (FH Mainz, summer 2009)** by Florian Jenett, 104 sketches by 11 users
- The Nature of Code** by Daniel Shiffman, 12 sketches by 3 users
- Generating Visuals, MAD '08 at UPF** by Ricard Marner Piñón, 73 sketches by 16 users
- MSc Adaptive Architecture & Computation at UCL** by Alasdair Turner, 23 sketches by 3 users
- Electronic Media Design program at Langara College in Vancouver** by Jer Thorp, 181 sketches by 23 users

**Collectionsbeta**

- Trees Generation** by Sinan Ascioglu, 43 sketches by 36 users
- Simply Object-Oriented** by Sinan Ascioglu, 18 sketches by 16 users
- Games** by Sinan Ascioglu, 59 sketches by 45 users
- default title** by Thiago Bueno Silva, 0 sketches by 0 users
- Physics** by Sinan, 15 sketches by 9 users
- SoftwareLike Programs** by Riley Galloway, 29 sketches by 13 users

At the bottom of the screenshot, there are several more thumbnails of sketches, including one titled 'Practice Sketch - Ultimate Circle' and another with 'MAD VIT' text.

sketches from classes on OpenProcessing.  
<http://openprocessing.org/collections/>



## 2.2 Use in education and Artistic/Design Production

Page: 12 \ 11 \ 10 \ 9 \ 8 \ 7 \ 6 \ 5 \ 4 \ 3 \ 2 \ 1



[Understanding Shakespeare](#)  
by Stephan Thiel

Introducing a new form of reading drama to help understand Shakespeare's works in new and insightful way. Using Processing, a number of word visualizations are created to highlight relationships throughout the play.

Links: [Stephan Thiel](#)



[One Perfect Cube](#)  
by Florian Jenett

Three synchronized clocks that form a cube image every twelve hours for exactly one second.

Links: [FlorianJenett.de](#)



[Feltron 2009 Annual Report](#)  
by Nicholas Felton

Each day in 2009, Felton asked every person with whom he had a meaningful encounter to submit a record of this meeting through an online survey...

Links: [2009 Report and Processing](#), [Feltron.com](#)



[Computing Kaizen](#)  
by GSAPP Hasegawa/Collins Studio

Toy software produced by an advanced graduate studio at the Columbia University GSAPP. Centered around the design of a technology incubator in the Akihabara district of Tokyo, this studio explored evolutionary structures that anticipate change and internalize complex relationships.

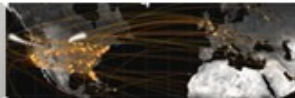
Links: [Proxy](#), [GSAPP](#)



[Fine Collection of Curious Sound Objects](#)  
by Georg Reil, Kathy Scheuring

Storytelling combined with physical computing. Six mundane objects incorporate sensors that trigger feedback to the user as part of a fictional history of the objects.

Links: [geschoir.de](#), [theplacestofindme.de](#)



[Just Landed](#)  
by Jer Thorp

Combines Processing, Twitter, and MetaCarta to map the phrase "Just landed in..." onto the globe and renders as video.

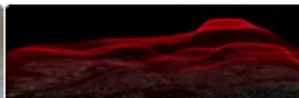
Links: [bjpm.com](#)



[John Henry von Neumann](#)  
by Chandler B. McWilliams

Performance that recasts the archetypal conflict between human and machine labor as a competition to complete an algorithmic drawing in an eight-hour workday.

Links: [brysonian.com](#)



[In the Air](#)  
by Victor Viña, Nerea Calvillo

Visualization project to reveal the invisible agents of Madrid's air (gases, particles, pollen, etc.), to see how they relate to the city.

Links: [Victor Viña](#), [Nerea Calvillo](#)



[Silica-Esc](#)  
by Vladimir Todorovic

Generative movie that portrays a possible computing platform for the future. The story takes place in Singapore, where a decision about the new device is about to be made.

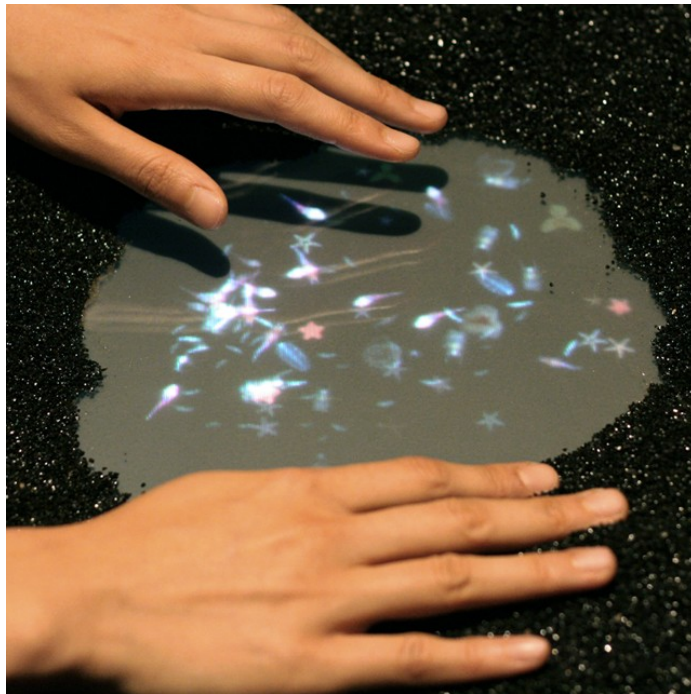
Links: [tadar.net](#), [tadar Flickr](#)

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

works based on Processing at this website:  
<http://processing.org/exhibition/>

## 2.2 Use in education and Artistic/Design Production

### Interactivity



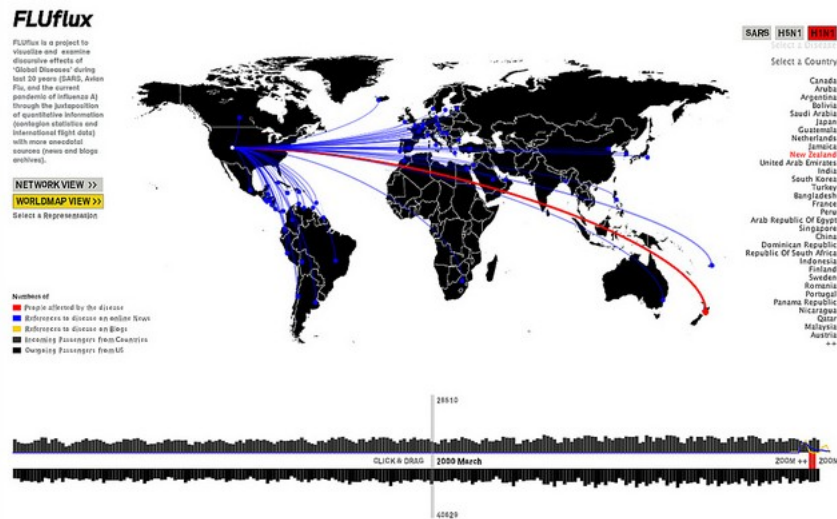
**<Oasis>**, 2008: Yunsil Heo, Hyunwoo Bang  
[http://everyware.kr/portfolio/contents/09\\_oasis/](http://everyware.kr/portfolio/contents/09_oasis/)



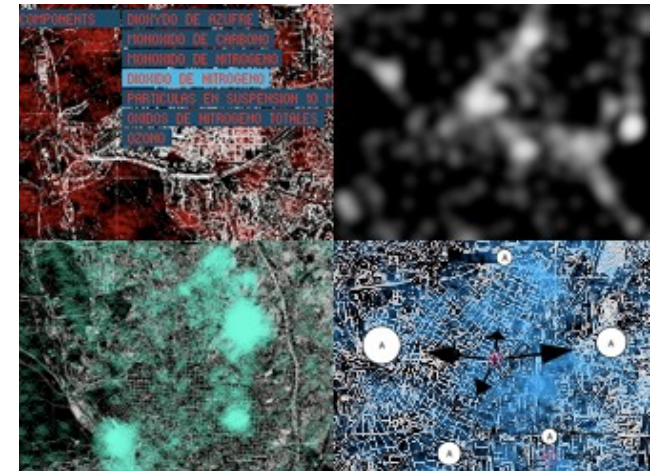
**<Shadow Monsters>**, 2005, Philip Worthington  
<http://worthersoriginal.com/>

## 2.2 Use in education and Artistic/Design Production

### Data Visualization



**<FLUflux>, 2009**, Andres Colubri, Jihyun Kim  
<http://threeecologies.com/fluflux/>



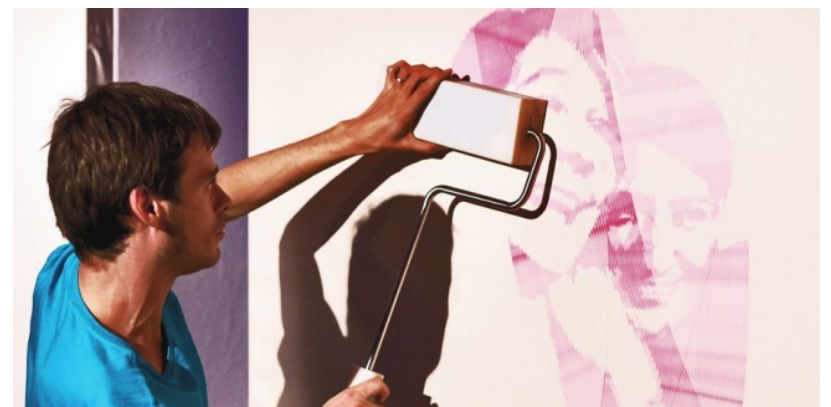
**<In the Air>, 2008**, Víctor Viña, Nerea Calvillo  
<http://www.intheair.es/>

## 2.2 Use in education and Artistic/Design Production

### Physical Computing



**<Openings>**, 2008, Andrea Boeck, Jihyun Kim, and Justin Liu  
<http://we-make-money-not-art.com/>



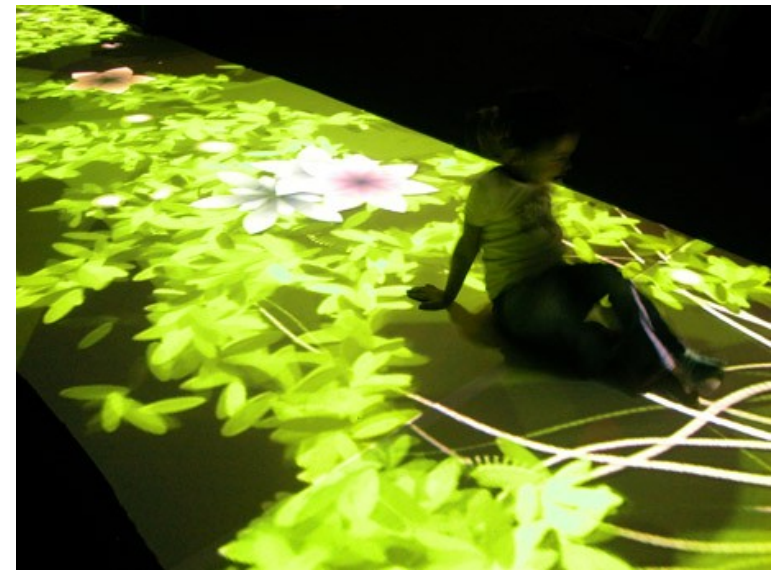
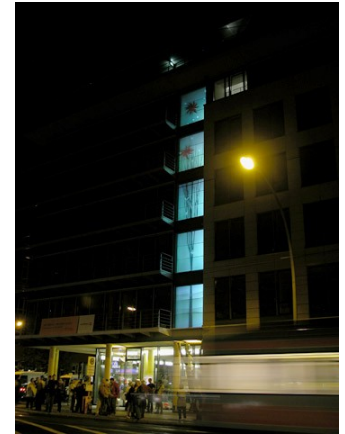
**<Light Roller>**, 2006, Random International  
<http://www.random-international.com/>

## 2.2 Use in education and Artistic/Design Production

Real-time graphics and video



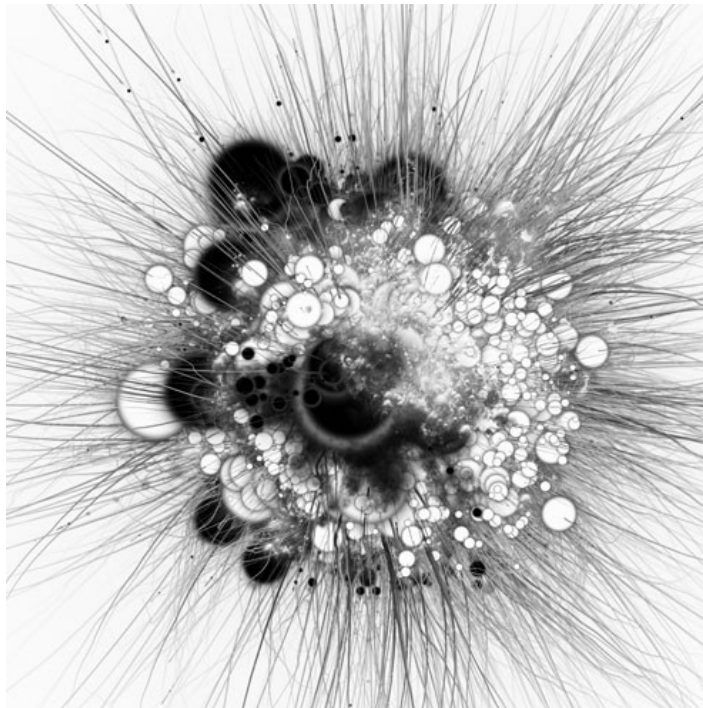
**< Latent State >**, 2009, Andres Colubri\_  
live cinema performance



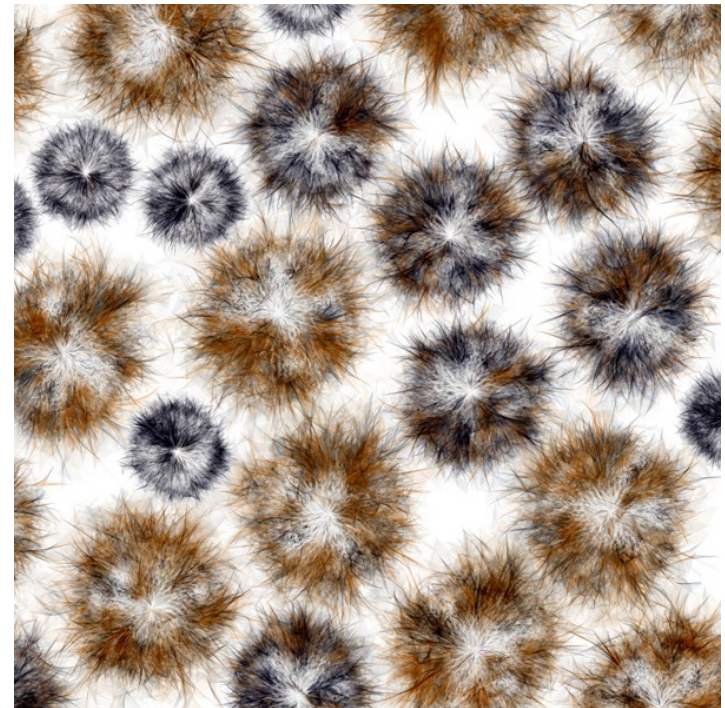
**<Vattnfall Media Façade>** Art+Com  
<http://www.artcom.de/>

## 2.2 Use in education and Artistic/Design Production

### 2.2.5 Generative art

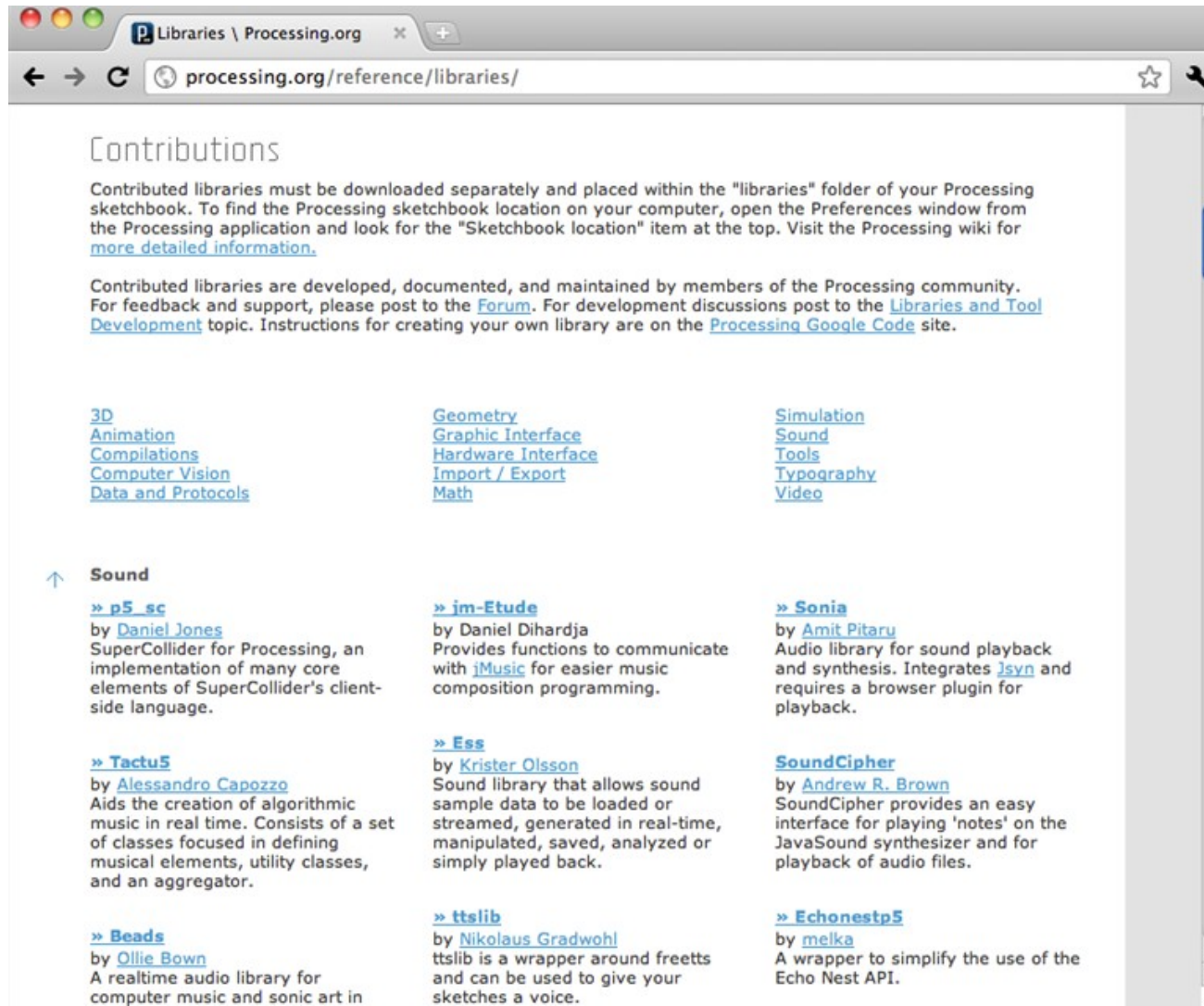


**<Magnetic Ink, pt. 3>** Robert Hodgin  
<http://www.flight404.com/blog/?p=103>



**<Process 6>** Casey Reas  
<http://reas.com/category.php?section=works>

## 2.3 Overview of Processing Libraries



Processing allows more than 100 external libraries contributed by the community.

# 3. Basic Concepts in Android applications

An Android application consists out of the following parts:

- 1. Activity** - A screen in the Android application
- 2. Services** - Background activities without UI
- 3. Content Provider** - provides data to applications, Android contains a SQL-Lite DB which can serve as data provider
- 4. Broadcast Receiver** - receives system messages, can be used to react to changed conditions in the system
- 5. Intents** - allow the application to request and/or provide services . For example the application call ask via an intent for a contact application. Application register itself via an IntentFilter. Intends are a powerful concept as they allow to create loosely coupled applications.

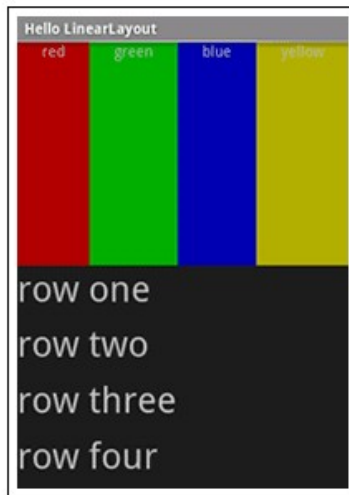
<https://sites.google.com/site/androidappcourse/>



# 3.1 Views, Activities, Intents, and the manifest file

## Views

Linear Layout



Relative Layout

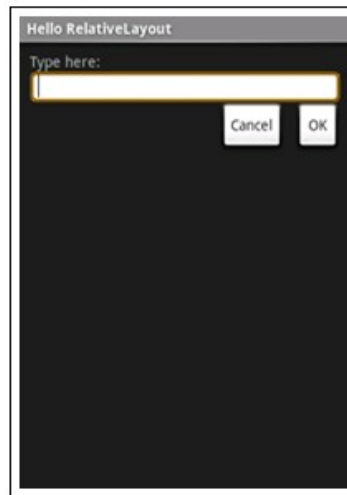
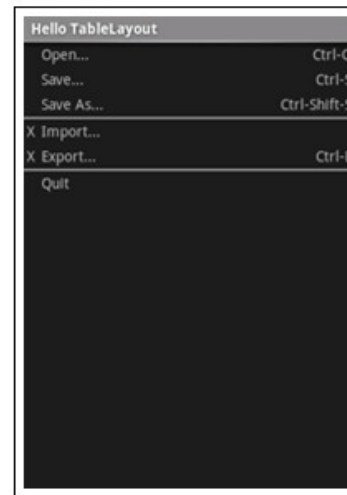
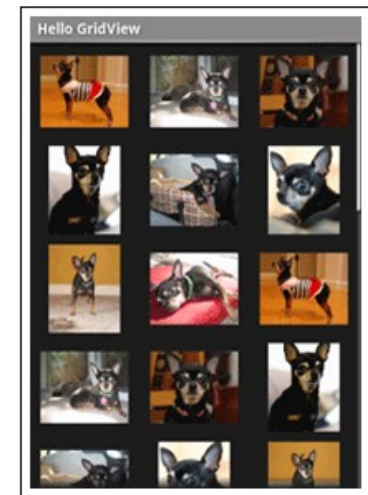


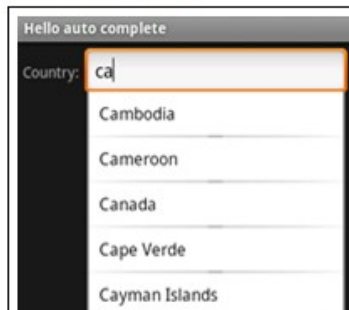
Table Layout



Grid View



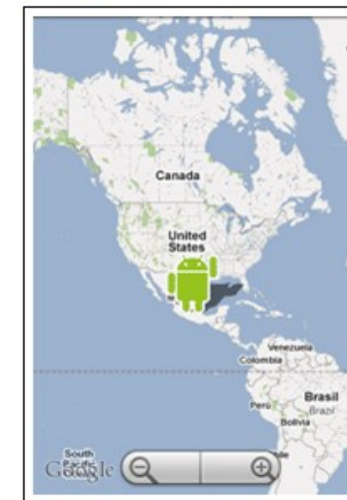
Auto Complete



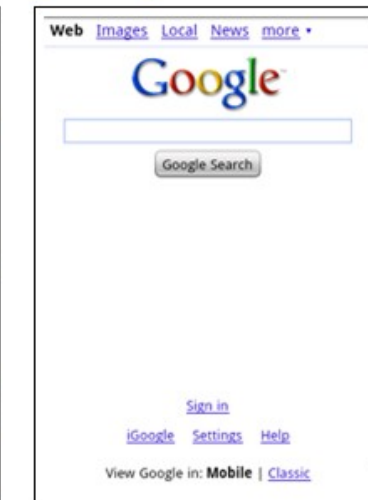
Gallery



Google Map View



Web View

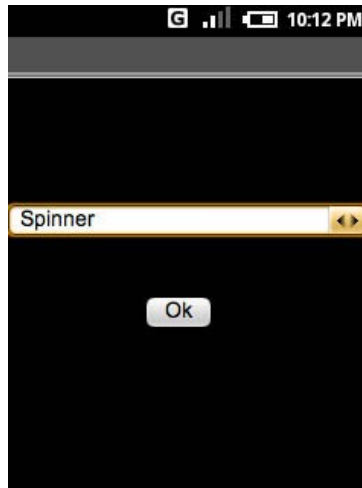


The visual content of the window is provided by a hierarchy of views — objects derived from the base **View** class. Each view controls a particular rectangular space within the window.

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

## 3.1 Views, Activities, Intents, and the manifest file

### GUI design



```
<?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 id="@+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/courses/cs160-sp08/wiki/index.php/Getting\\_Started\\_with\\_Android](http://vis.berkeley.edu/courses/cs160-sp08/wiki/index.php/Getting_Started_with_Android)

Android uses an **XML based** markup language to define user interface layouts, in a way that's similar to UIML. XML is used to create flexible interfaces which can then be modified and wired up in the Java code. Mozilla's XUL, Windows Presentation Foundation XAML, and Macromedia Flex's MXML (and to some extent even SVG) all operate similarly. Each node in the XML tree corresponds to a screen object or layout container that will appear in the rendered interface..

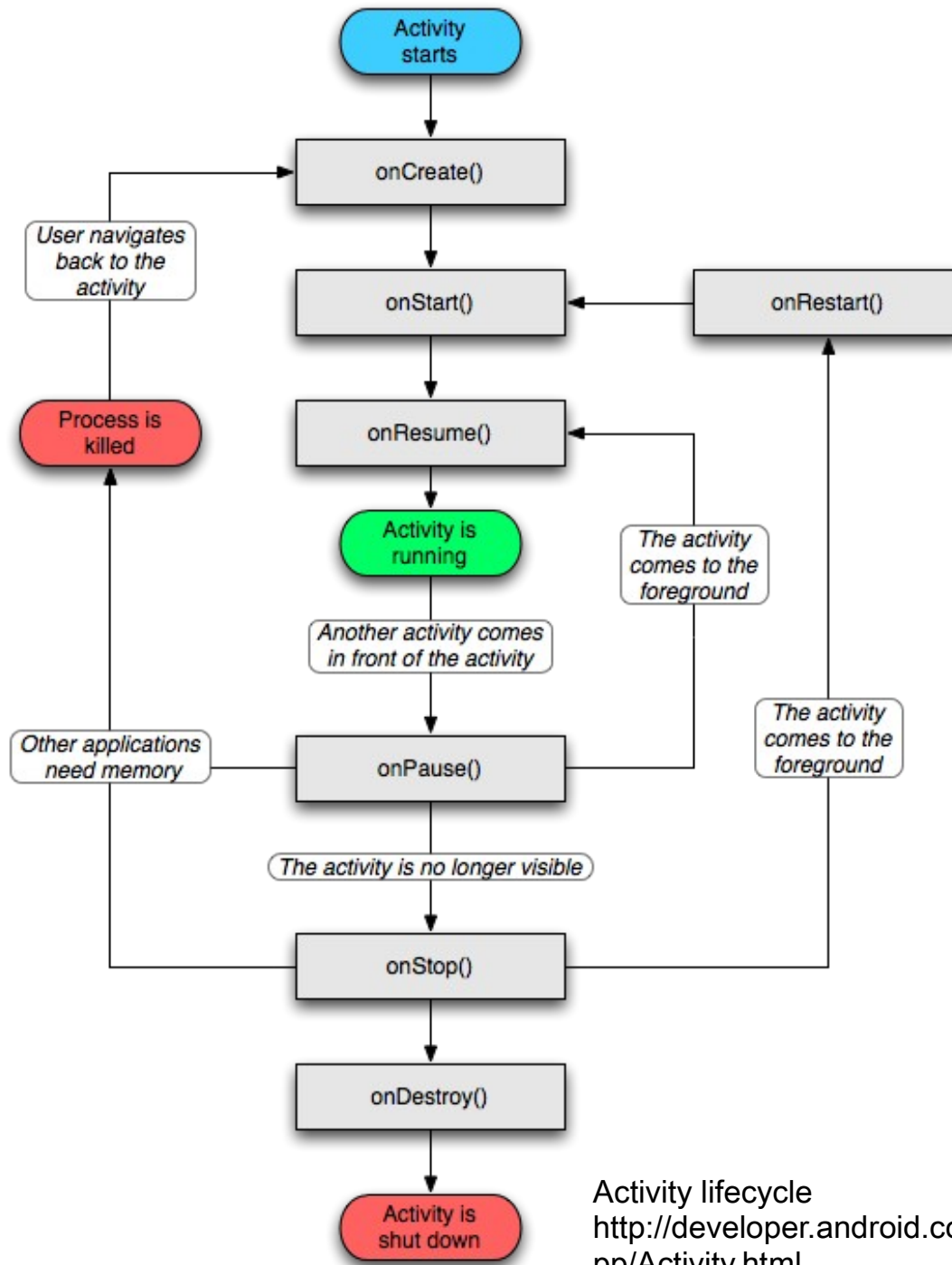
## 3.1 Views, Activities, Intents, and the manifest file

### 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. 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.

<http://developer.android.com/guide/topics/fundamentals.html>



Activity lifecycle  
<http://developer.android.com/reference/android/app/Activity.html>

## 3.1 Views, Activities, Intents, and the manifest file

### Service



<http://blog.gbinghan.com/2010/08/android-basics-quick-start.html>

**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.

<http://developer.android.com/guide/topics/fundamentals.html>

## 3.1 Views, Activities, Intents, and the manifest file

### 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 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 to act on, among other things.

<http://developer.android.com/guide/topics/fundamentals.html>

## 3.1 Views, Activities, Intents, and the manifest file

### Manifest file

```
<?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>
```

An Android application is described in the file "**AndroidManifest.xml**". This file 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.

## 3.2 Android Software Development Kit (SDK)

The **Android SDK** includes a comprehensive set of development tools. 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).



## 3.3 Use of Eclipse for Android development



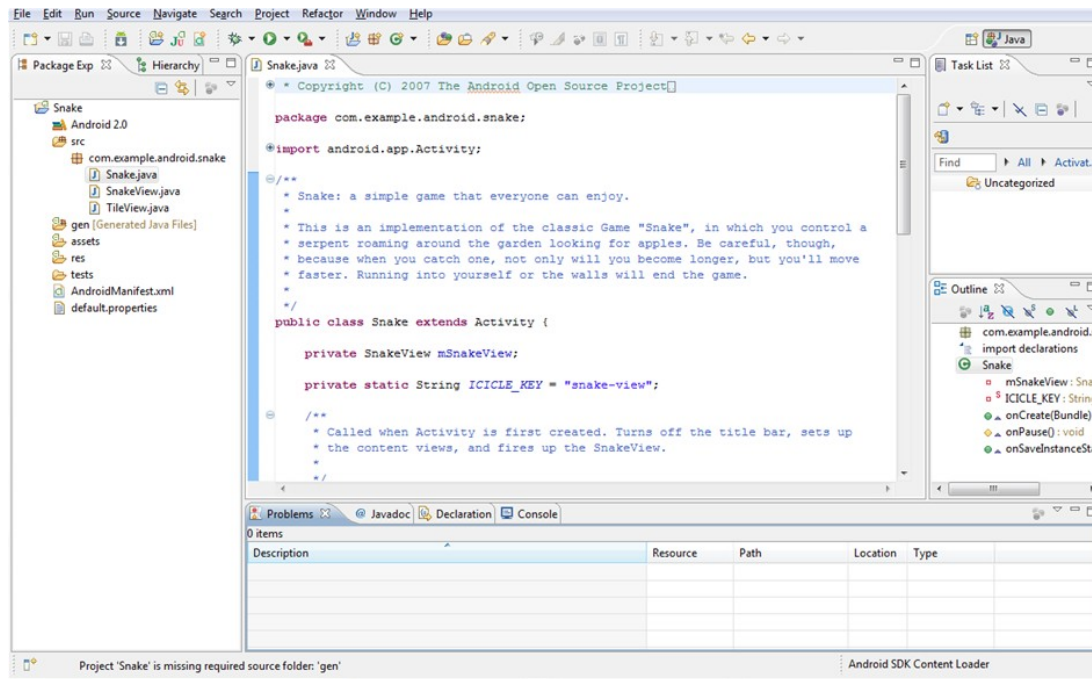
**Eclipse** can be used as the IDE for Android development. To do so, you need to install the SDK and then the: ADT (Android Development Tools)

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 Android SDK tools, and even export signed (or unsigned) APKs in order to distribute your application.

## 3.3 Use of Eclipse for Android development

### 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**.

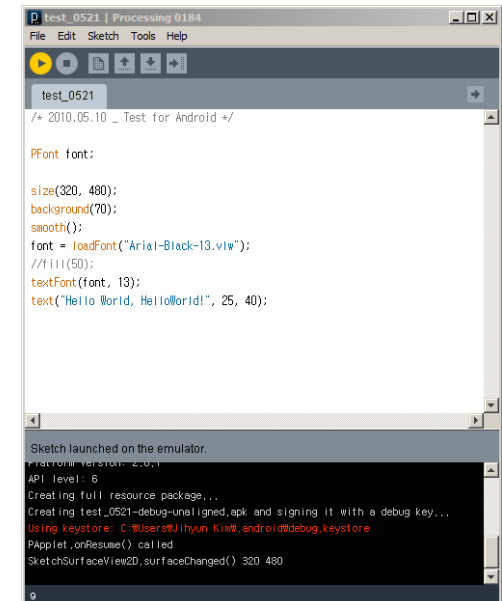
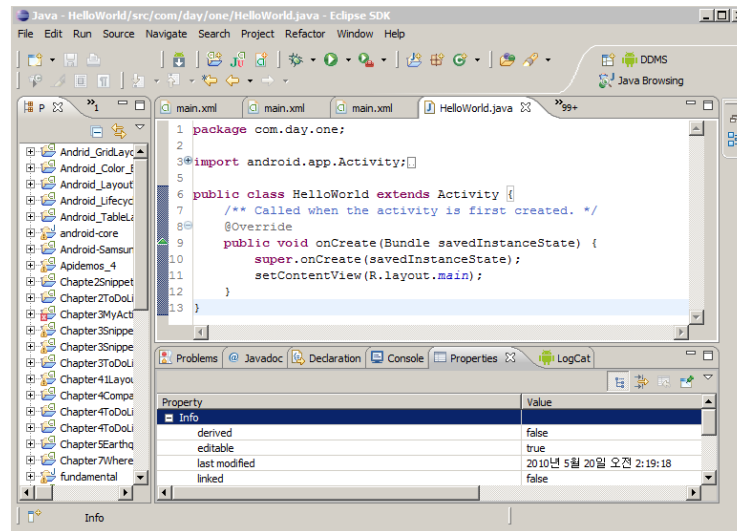
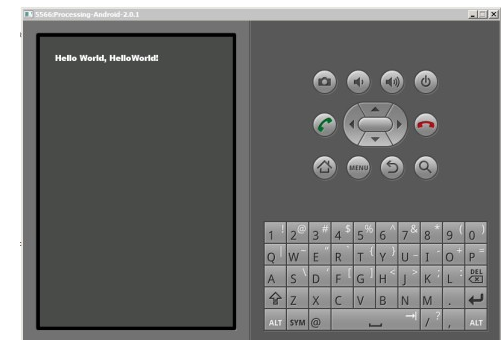
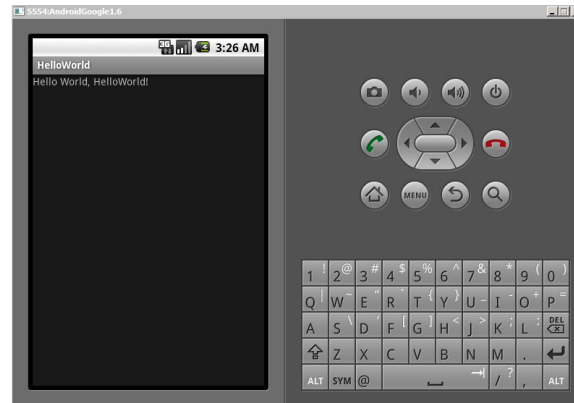


## Solution: **Processing for Android!**

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!"

# 4. First steps with Processing for Android



**Eclipse**

**Processing**

## 4.0 Current status of Processing for Android



At the time of this writing, the latest pre-release of Processing with Android support is 0192, which you can download from this page:

<http://code.google.com/p/processing/downloads/list>

Since this is a pre-release, it should be considered as “alpha” status software which contains bugs and unfinished features. However, it is functional enough to get started with Android development, even for 3D.

**Important note:** Processing for Android currently needs the Android SDK to be manually installed beforehand, the plan is that next releases of Processing will bundle the SDK.

Since this feature is still not implemented, we will describe in the next slides how to install the SDK on Windows, OSX, and Linux

# 4.1 Installation of 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.

Detailed instructions

<http://developer.android.com/sdk/installing.html>

Here are the Java SE downloads in detail.

Java Platform, Standard Edition	
<b>JDK 6 Update 20 (JDK or JRE)</b> This release contains critical security updates to the Java runtime. Please update now to take advantage of these enhancements. » <a href="#">Learn more</a>	<a href="#">Download JDK</a> <a href="#">Docs</a>
<b>What Java Do I Need?</b> You must have a copy of the JRE (Java Runtime Environment) on your system to run Java applications and applets. To develop Java applications and applets, you need the JDK (Java Development Kit), which includes the JRE. <b>NOTE:</b> The Firefox 3.6 browser requires Java SE 6 Update 10 or later. Otherwise, Java-based web applications <u>will not work</u> .	<a href="#">Download JRE</a> <a href="#">Docs</a>

<http://java.sun.com/javase/downloads/index.jsp>

## Pre-Releases

- 0184 | 2010 04 14 [Windows \(without Java\)](#) | [Mac OS X](#) | [Linux x86](#) | *syntax & android fixes*
- 0182 | 2010 03 29 [Windows \(without Java\)](#) | [Mac OS X](#) | [Linux x86](#) | *autoformat, syntax, pdf fixes*
- 0181 | 2010 03 19 [Windows \(without Java\)](#) | [Mac OS X](#) | [Linux x86](#) | *bug fixes for 0180*
- 0180 | 2010 03 15 [Windows \(without Java\)](#) | [Mac OS X](#) | [Linux x86](#) | *pre-release supporting Java 5 syntax*

<http://processing.org/download/>

Android developers

Home **SDK** Dev Guide Reference Resources Videos Blog

Android SDK Starter Package

Download  
Installing the SDK

Downloadable SDK Components

Adding SDK Components

- Android 2.1 Platform *new!*
- Android 1.6 Platform
- Android 1.5 Platform

Older Platforms

- SDK Tools, r5 *new!*
- USB Driver for Windows, r3

ADT Plugin for Eclipse

- ADT 0.9.6 *new!*

Native Development Tools

- Android NDK, r3 *new!*

More Information

### Download the Android SDK

Welcome Developers! If you are new to the Android SDK, please read the [Quick Start](#), below, for an overview of downloading a new SDK package.

If you are already using the Android SDK and would like to update to the latest tools or platforms, please use it downloading a new SDK package.

Platform	Package	Size	MD5 Checksum
Windows	<a href="#">android-sdk_r05-windows.zip</a>	23449838 bytes	cc2c51a24e2f976e0fa652e182ef5840
Mac OS X (intel)	<a href="#">android-sdk_r05-mac_os_x.zip</a>	19871714 bytes	6fced0e1c36624c926551637eb3308
Linux (386)	<a href="#">android-sdk_r05-linux_386.tar</a>	16208523 bytes	1d695d6a31310406f5d49092a1bd9850

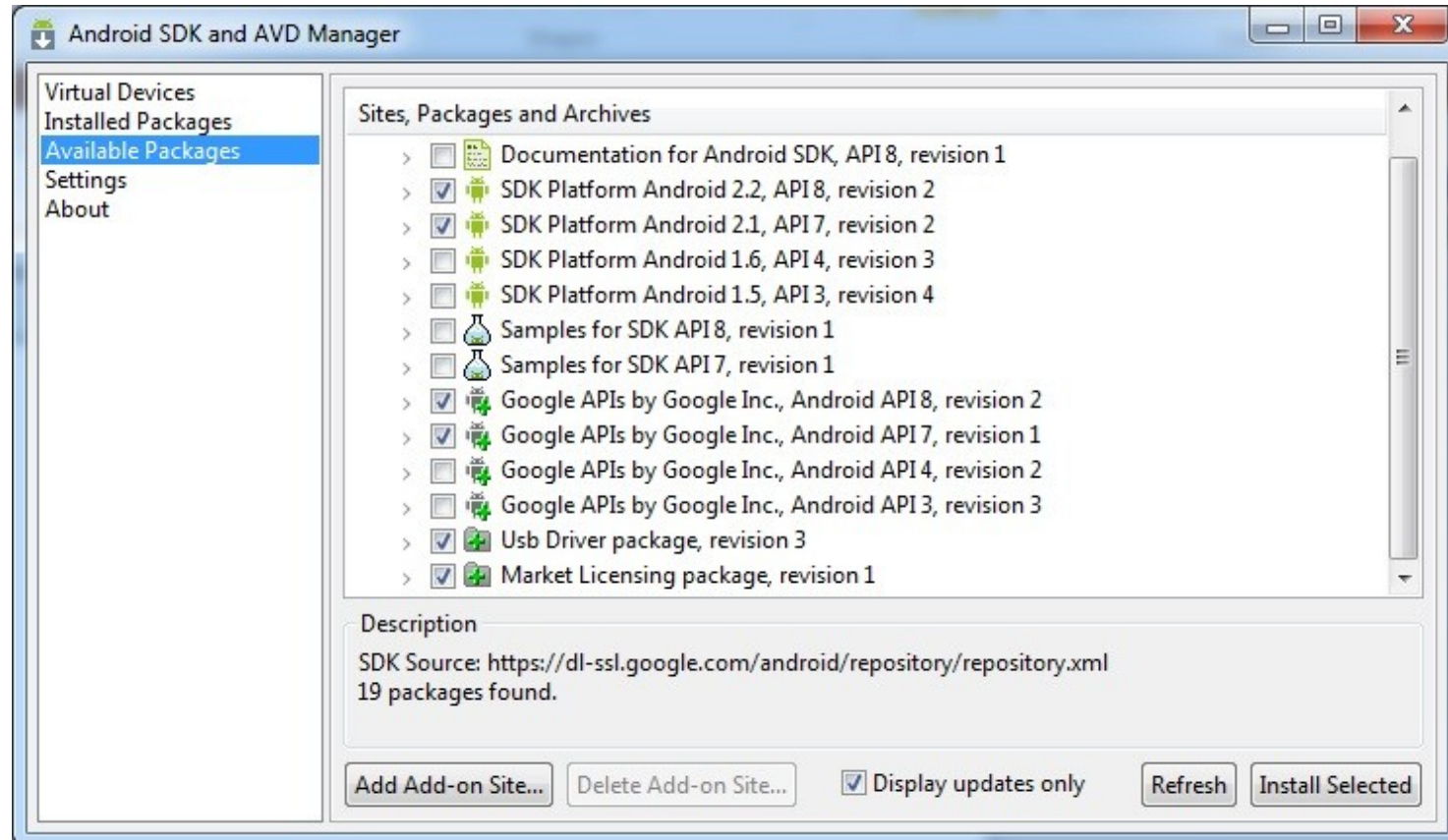
<http://developer.android.com/>

## 4.1 Installation of the Android SDK and Processing

### Windows

1. Make sure that you install the **JDK (Java Development Kit) first...**
2. **Download latest package** from <http://developer.android.com/sdk/index.html>
3. **Unzip package** at the place of your preference...  
C:\Users\andres\Coding\android-sdk-windows
4. Add the environmental variables **PATH and ANDROID\_SDK**
5. Also, if the path to the bin folder of the JDK is not in the PATH, add it as well.
6. The last step in setting up your SDK is running the android.exe application included in the tools folder of the SDK. This will launch the Android SDK and ADV Manager, which allows to download and install the rest of the required components of the SDK.

## 4.1 Installation of the Android SDK and Processing



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!)



## 4.1 Installation of the Android SDK and Processing

### **Additional step on Windows: USB driver installation**

This procedure is very important in Windows to be able to connect the Android devices to Processing.

The USB driver that comes with the Android SDK 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 following the steps indicated here:

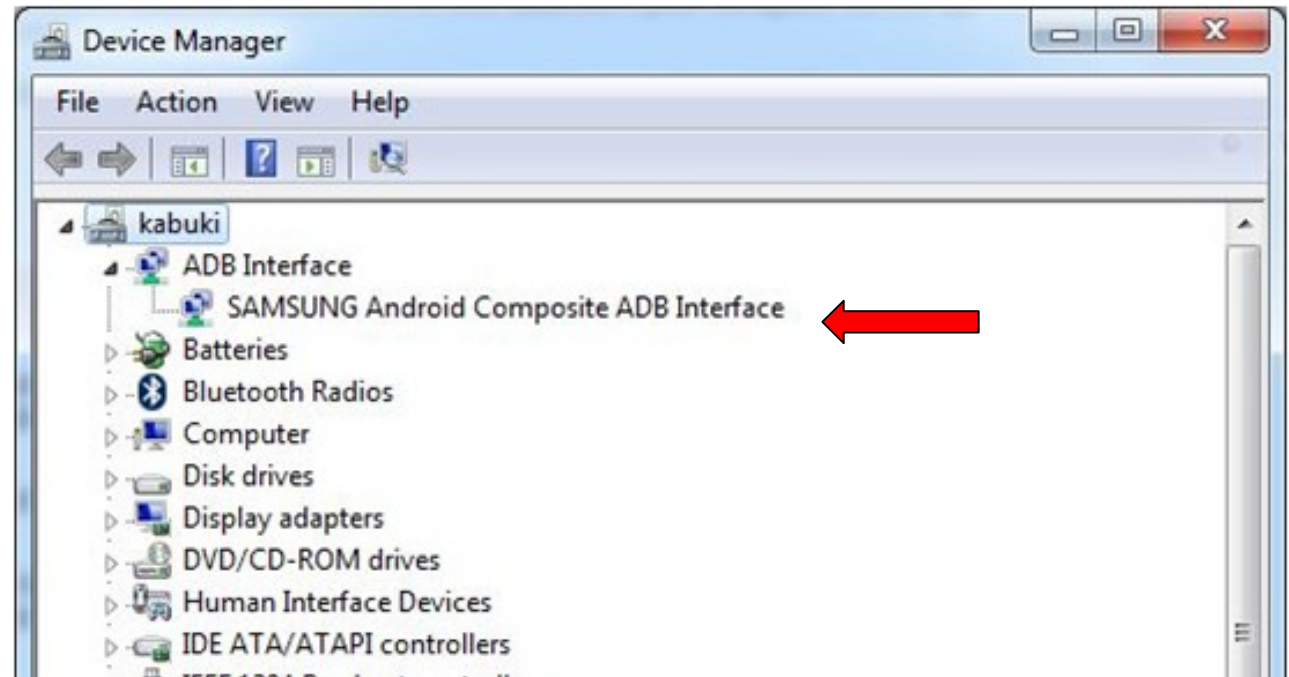
<http://developer.android.com/sdk/win-usb.html>

## 4.1 Installation of the Android SDK and Processing

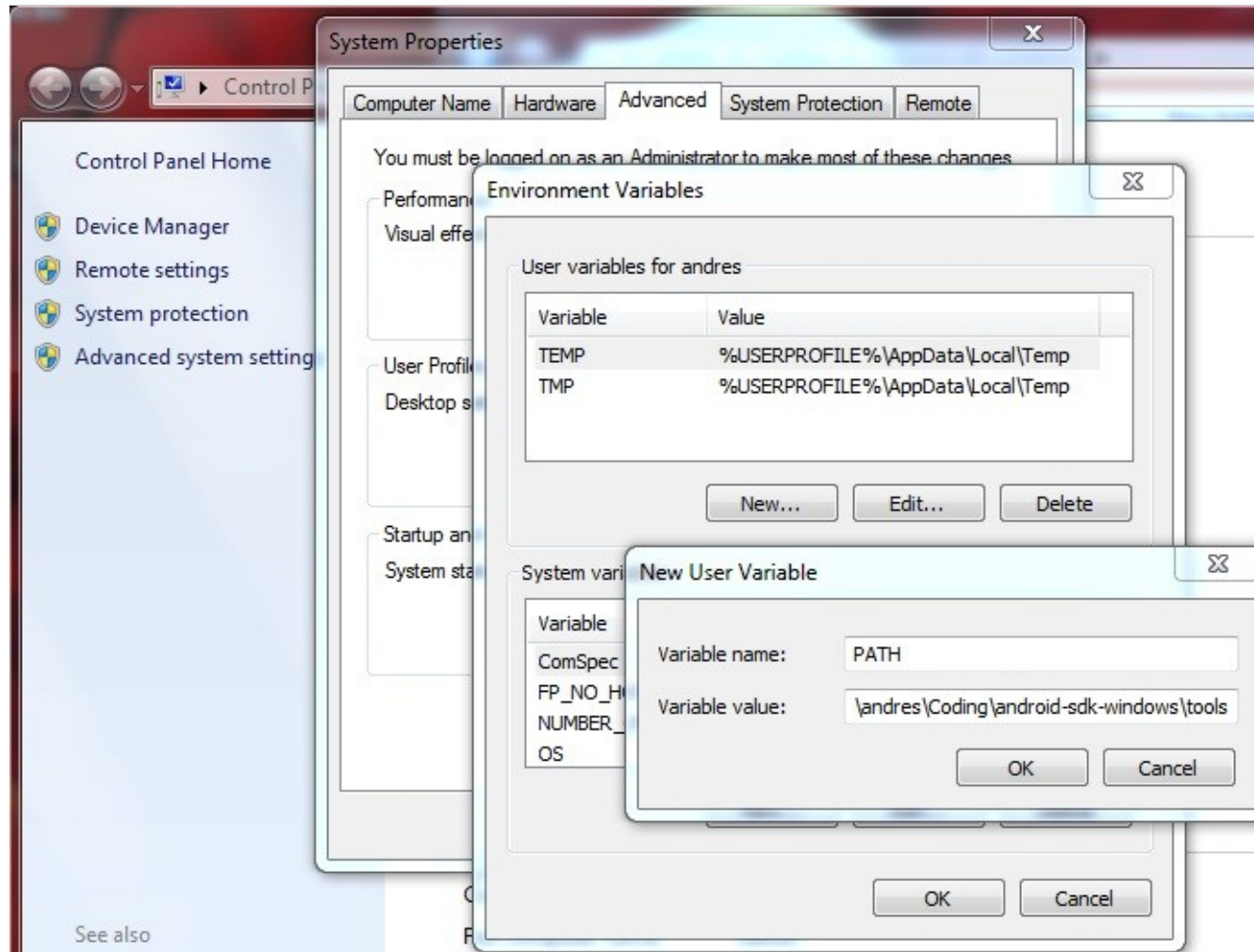
Other phones might require the USB drivers from the **manufacturer of the device**. For instance, the ones for the **Galaxy S** are available here:

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

After installation, the phone should appear in the Device Manager as follows:



## 4.1 Installation of the Android SDK and Processing



Setting environmental variables in Windows

## 4.1 Installation of the Android SDK and Processing

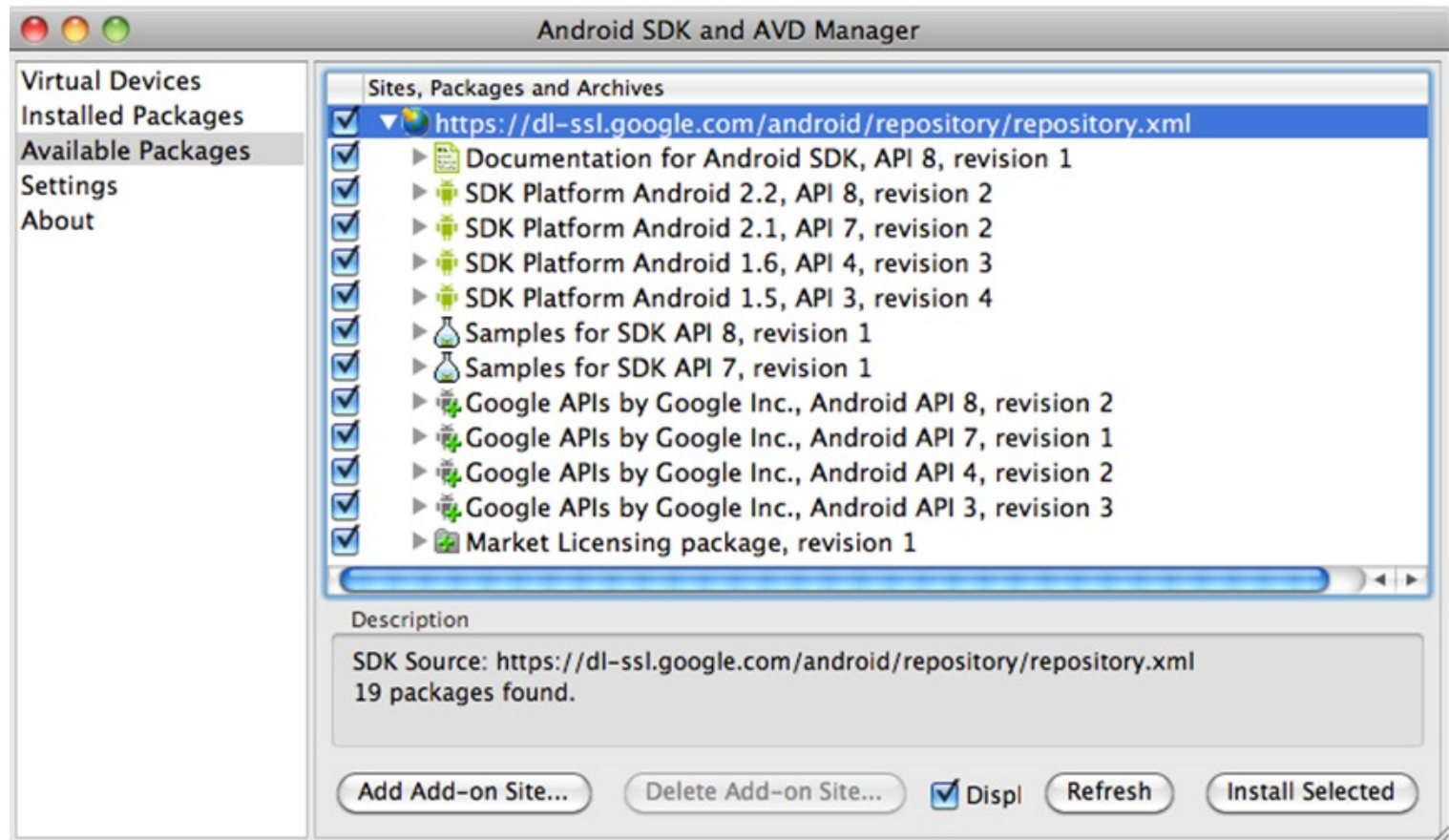
### Mac OSX

1. Download latest package from <http://developer.android.com/sdk/index.html>
2. Unzip package at the place of your preference, for example `/Users/joe/Android/android-sdk-mac_x86`
3. On a Mac OS X, look in your home directory for `.bash_profile` and proceed same as for Linux. You can create the `.bash_profile` if you haven't already set one up on your machine and add

```
export PATH=${PATH}:<your_sdk_dir>/tools
export ANDROID_SDK=${PATH}:<your_sdk_dir>
```

4. The last step consists in setting up your SDK is running the *Android SDK and AVD Manager* . You can start it from the terminal by typing *android*

## 4.1 Installation of the Android SDK and Processing



Make sure to install:

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

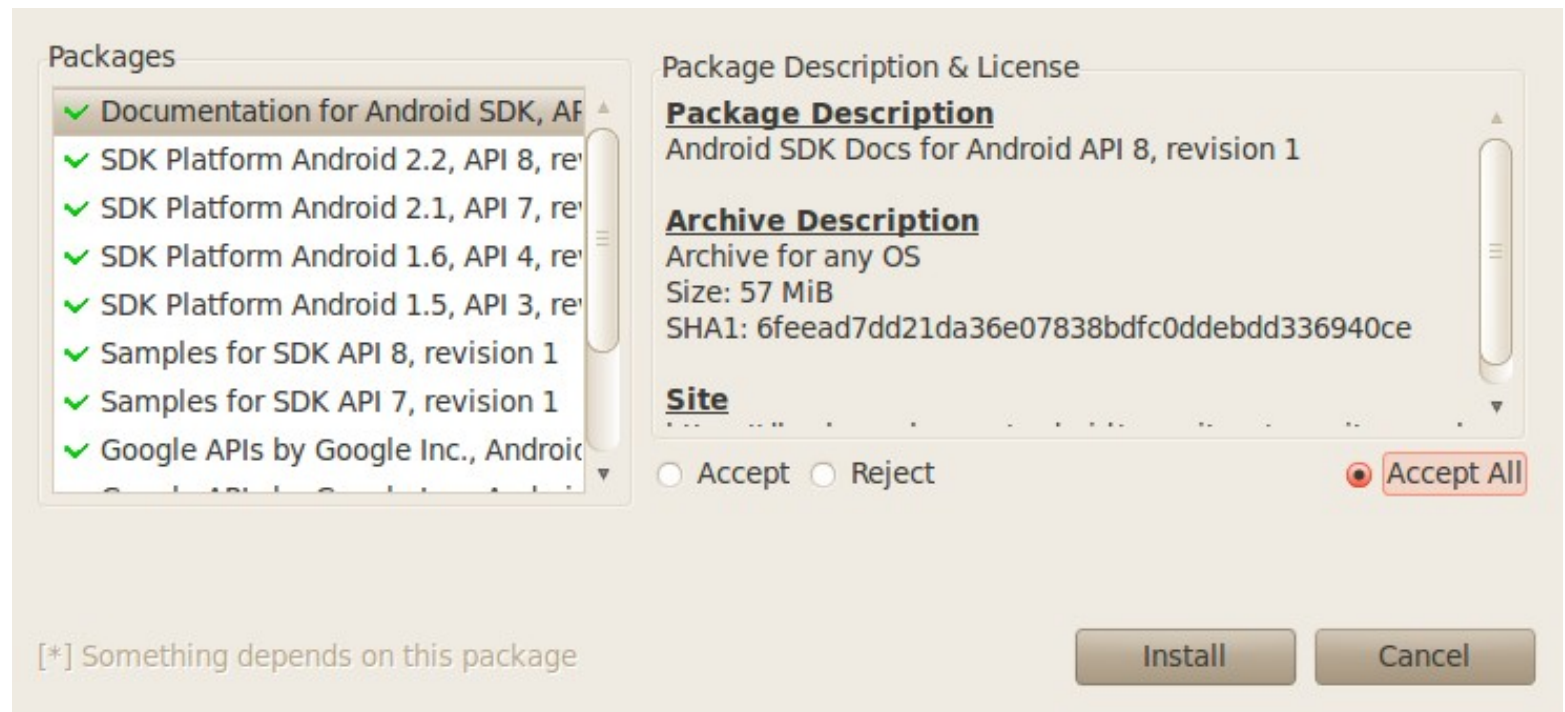
## 4.1 Installation of the Android SDK and Processing

### Linux

1. Make sure that you install the sun-java JDK first. On Ubuntu:  
`sudo apt-get install sun-java6-jdk`
2. Download latest linux package from <http://developer.android.com/sdk/index.html>
3. 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 some additional packages:  
<http://stackoverflow.com/questions/2710499/android-sdk-on-a-64-bit-linux-machine>
4. Add the environmental variables PATH and ANDROID\_SDK
5. 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:  

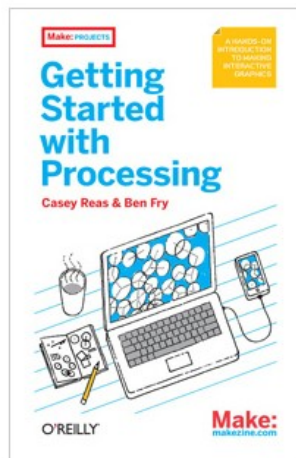
```
export PATH=${PATH}:<your_sdk_dir>/tools
export ANDROID_SDK=<your_sdk_dir>
```
6. The last step consists in setting up your SDK is running the *Android SDK and AVD Manager* . You can start it from the terminal by typing *android*

## 4.1 Installation of the Android SDK and Processing



# 4.1 Installation of the Android SDK and Processing

Download  
Processing



We're excited to announce the release of [Getting Started with Processing](#). This informal, short book is an introduction to Processing and interactive computer graphics. [Please have a closer look.](#)

A range of books for different skill levels are featured on the [Books page](#).

- » [Download Processing](#)
- » [Explore the Exhibition](#)
- » [Play with Examples](#)
- » [Browse Tutorials](#)

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.

- » Free to download and open source
- » Interactive programs using 2D, 3D or PDF output
- » OpenGL integration for accelerated 3D
- » For GNU/Linux, Mac OS X, and Windows
- » Projects run online or as double-clickable applications
- » Over 100 libraries extend the software into sound, video, computer vision, and more...

To see more of what people are doing with Processing, check out these sites:

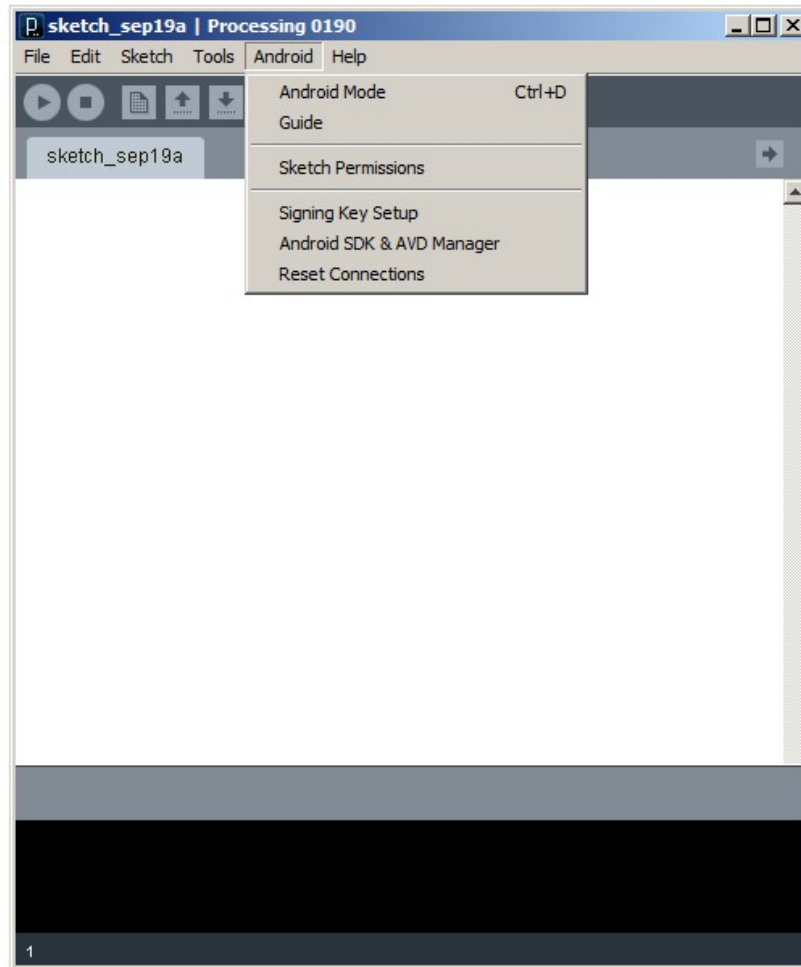
- » [Processing Wiki](#)
- » [Processing Discussion Forum](#)
- » [OpenProcessing](#)
- » [CreativeApplications.Net](#)
- » [O'Reilly Answers](#)
- » [Vimeo](#)
- » [del.icio.us](#)

Processing website:  
<http://processing.org/download/>

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

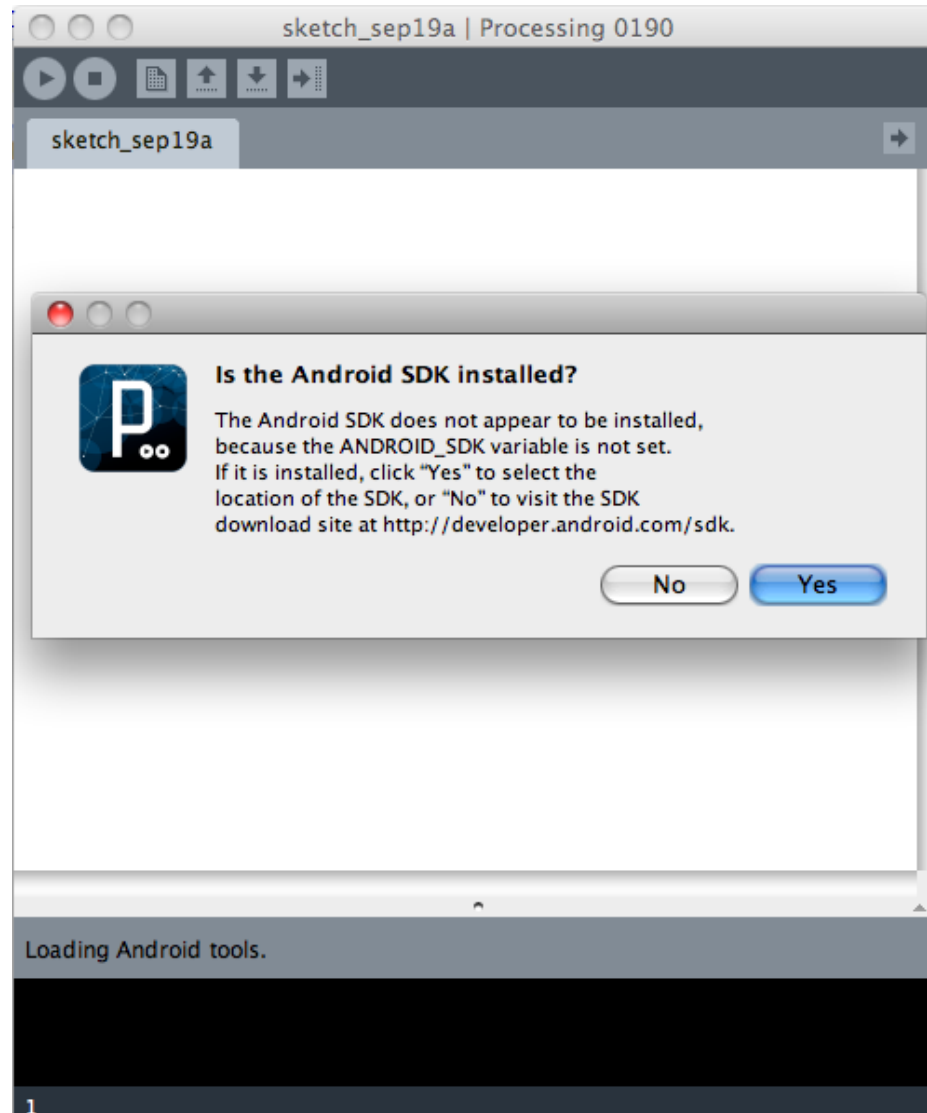


## 4.2 Android Mode in Processing



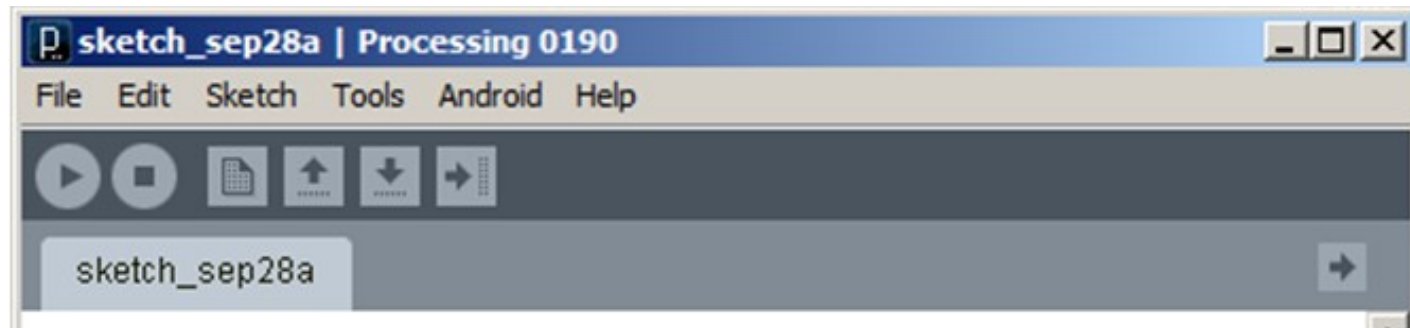
To run our code on the Android emulator or on the Android device, we need to set enable the **Android mode** in the PDE.

## 4.2 Android Mode in Processing



If missing the ANDROID\_SDK variable, just select the folder when asked

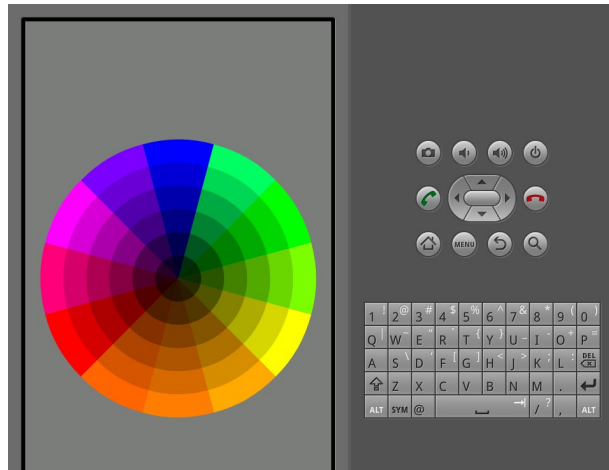
## 4.2 Android Mode in Processing



- **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.

## 4.3 Running Sketches on the emulator and the device

Android  
Emulator



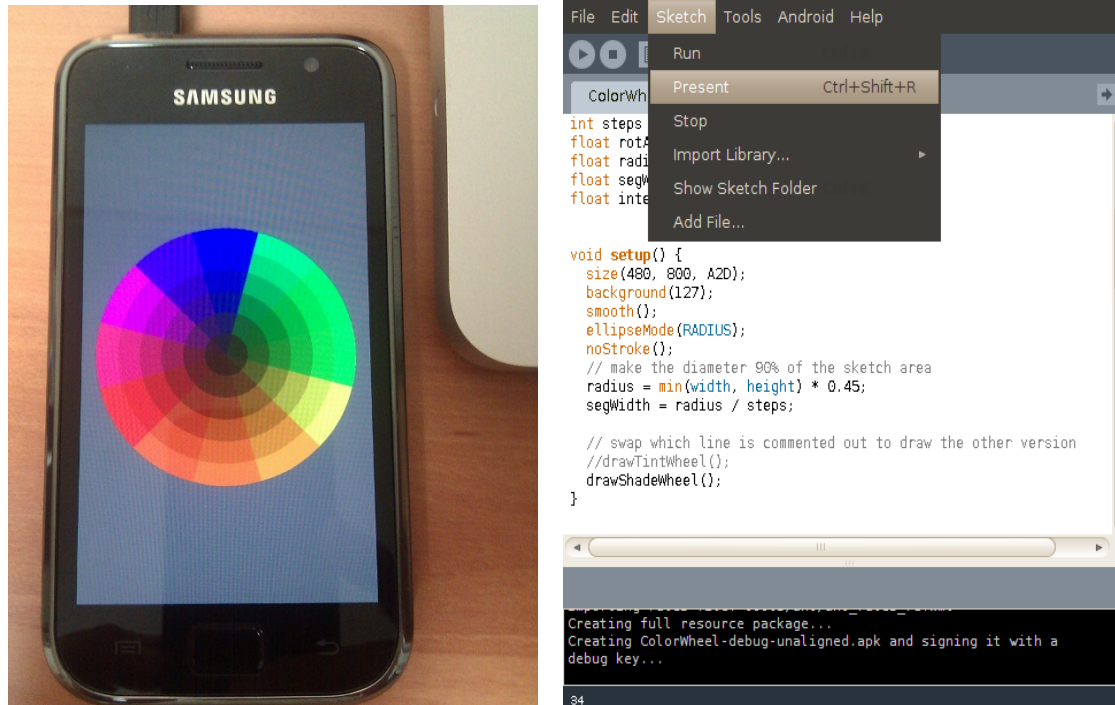
```
File Edit Sketch Tools Android Help
[Play] [Stop] [Screenshot] [Refresh] [Run]
ColorWheel 5
*/
int segs = 12;
int steps = 6;
float rotAdjust = TWO_PI / segs / 2;
float radius;
float segWidth;
float interval = TWO_PI / segs;

void setup() {
  size(480, 800, A2D);
  background(127);
  smooth();
  ellipseMode(RADIUS);
  noStroke();
  // make the diameter 90% of the sketch area
  radius = min(width, height) * 0.45;
  segWidth = radius / steps;

  // swap which line is commented out to draw the other version
  //drawTintWheel();
  drawSketchWheel();
}

Sketch launched on the emulator.
Creating full resource package...
Creating ColorWheel-debug-unaligned.apk and signing it with a
debug key...
```

## 4.3 Running Sketches on the emulator and the device



The device has to be properly connected to the USB port, and running at least **Android 2.1**.

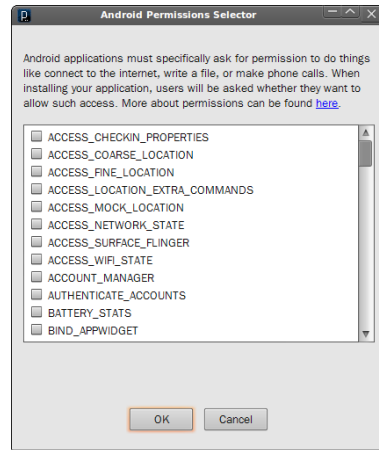
On Linux, you might need to run first the following commands from the terminal:

```
adb-killserver  
cd <sdk location>/tools  
sudo ./adb start-server
```

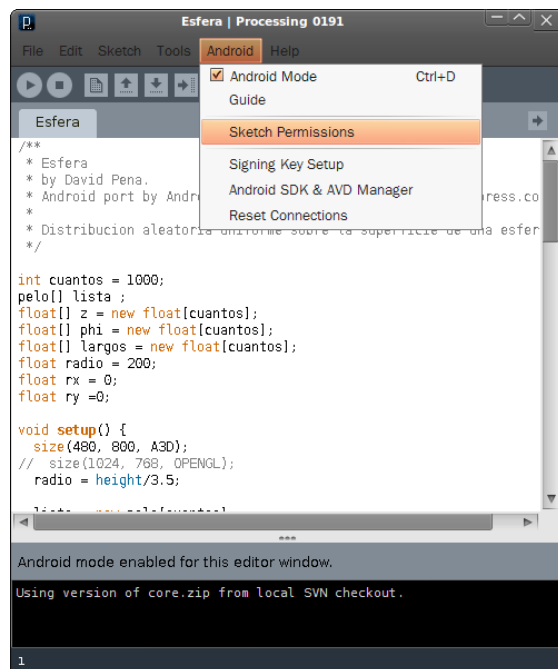
Type `adb devices` to get the list of currently connected devices.

## 4.3 Running Sketches on the emulator and the device

### 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:

<http://developer.android.com/guide/topics/security/security.html#permissions>  
<http://developer.android.com/reference/android/Manifest.permission.html>

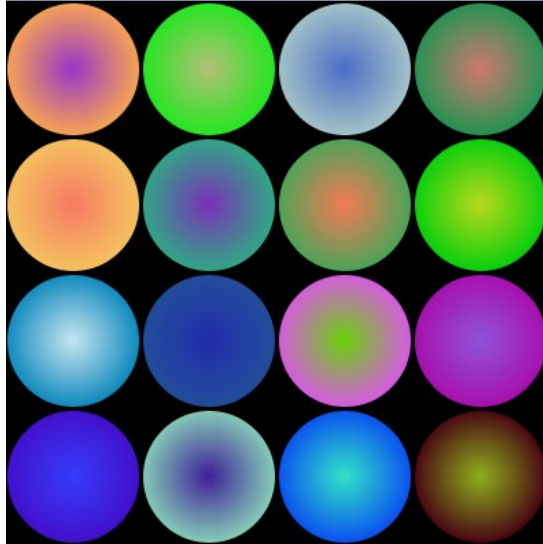
# 5. Basic Processing use

## 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>

## 5.1 Drawing of 2D shapes and use of color



<http://processing.org/learning/basics/radialgradient.html>

### Functions and Parameters

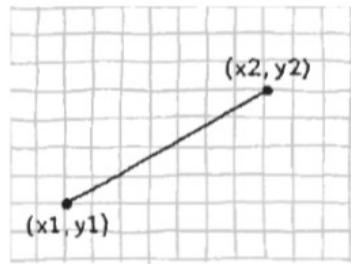
`size()`, `point()`, `line()`, `Triangle()`,  
`quad()`, `rect()`, `ellipse()`, `arc()`, `vertex()`

### Color

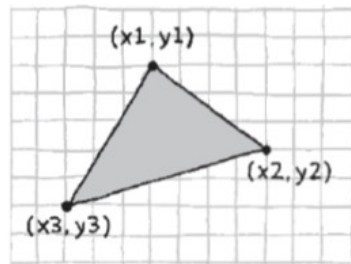
`Color()`, `Colormode()`, `fill()`, `stroke()`, `Background()`



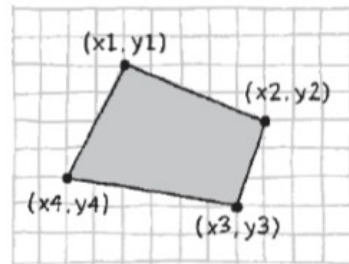
# 5.1 Drawing of 2D shapes and use of color



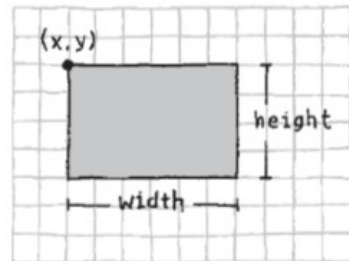
`line(x1, y1, x2, y2)`



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



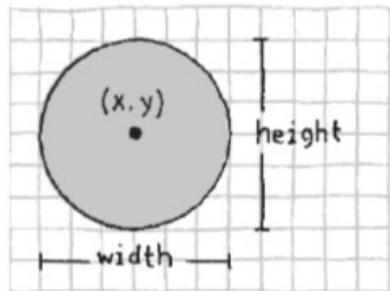
`quad(x1, y1, x2, y2, x3, y3, x4, y4)`



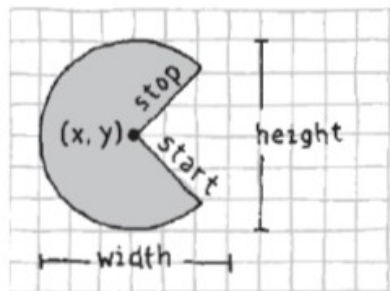
`rect(x, y, width, height)`

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

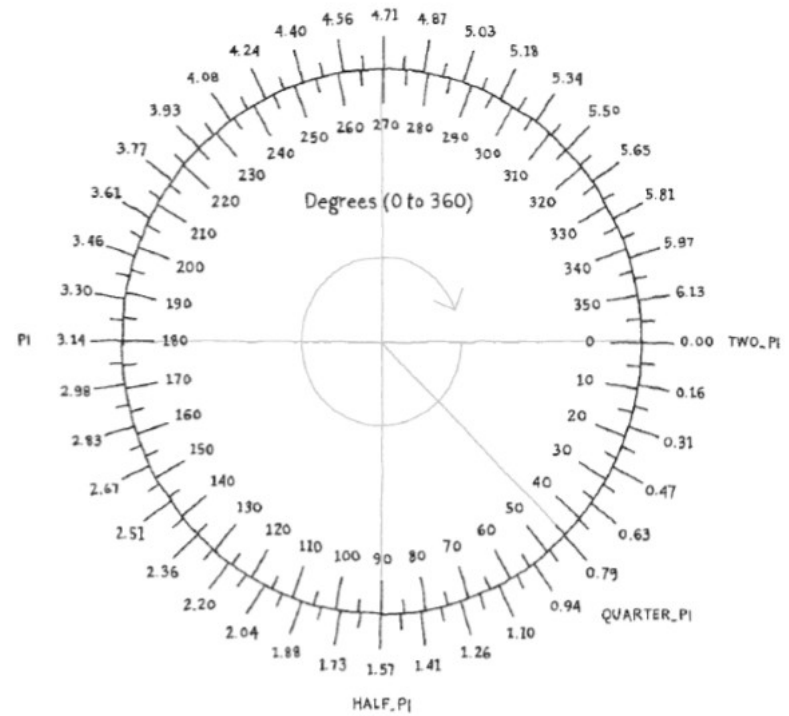
# 5.1 Drawing of 2D shapes and use of color



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



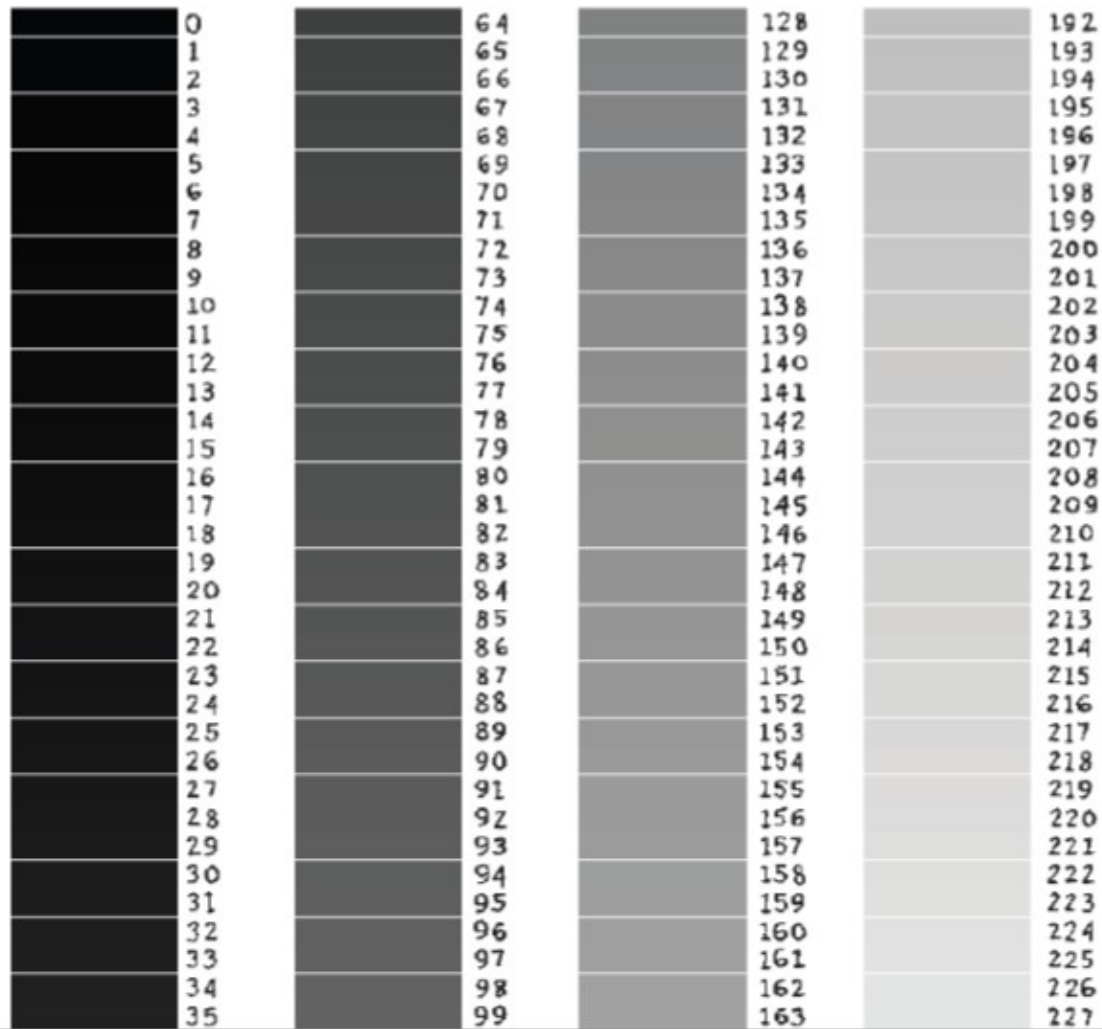
`arc(x, y, width, height, start, stop)`



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

## 5.1 Drawing of 2D shapes and use of color

Grayscale

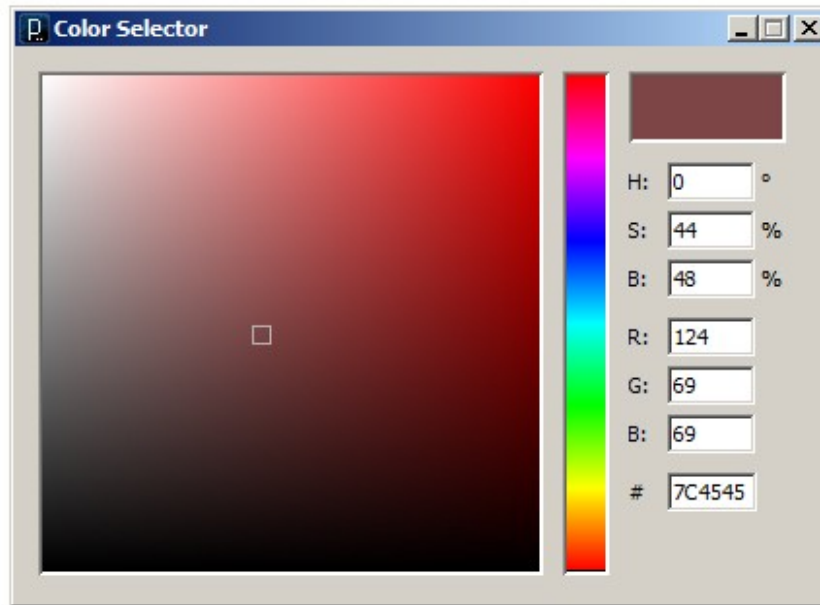


**0(black) – 255(white)**

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

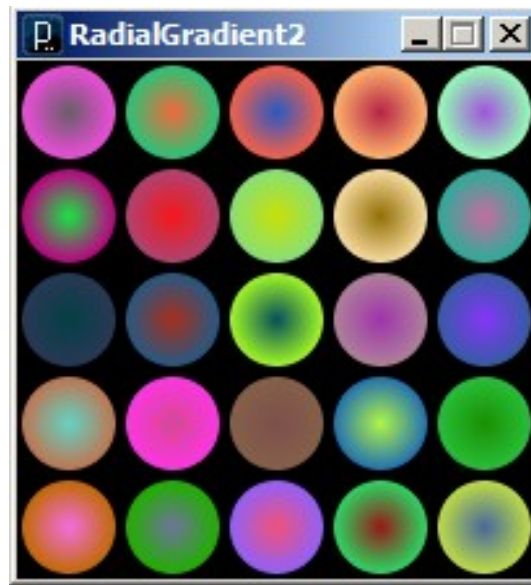
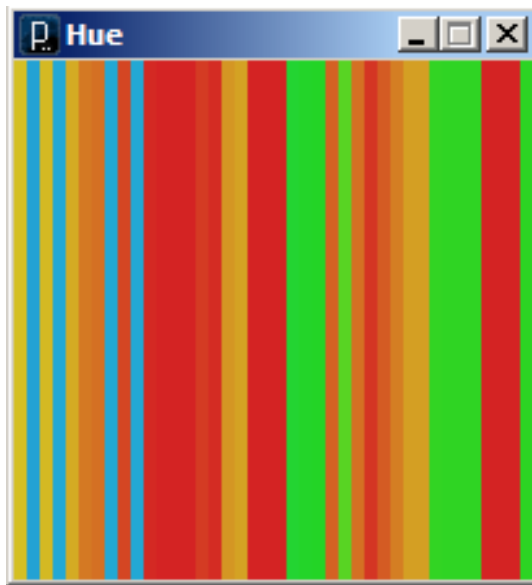
## 5.1 Drawing of 2D shapes and use of color

Color



```
color(gray)  
color(gray, alpha)  
color(value1, value2, value3)  
color(value1, value2, value3, alpha) color(hex)  
color(hex, alpha)
```

## 5.1 Drawing of 2D shapes and use of color



<Examples from processing>

```
fill(gray), fill(gray, alpha), fill(value1, value2, value3),  
fill(value1, value2, value3, alpha) fill(color),  
fill(color, alpha), fill(hex) fill(hex, alpha)
```

## 5.2 Animation and motion

### **setup()**

Called once when the program is started. Used to define initial environment properties such as screen size, background color, loading images, etc. before the **draw()** begins executing. Variables declared within **setup()** are not accessible within other functions, including **draw()**. There can only be one **setup()** function for each program and it should not be called again after its initial execution.

<http://processing.org/reference/>

```
void setup() {  
  println("Setup: Start");  
}  
void draw() {  
  println("I'm running");  
}
```

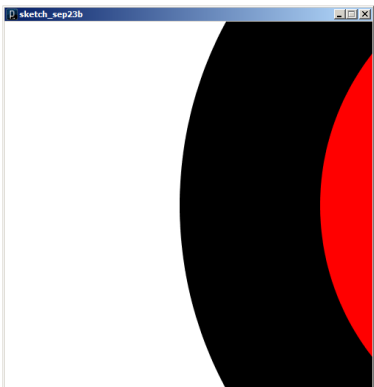
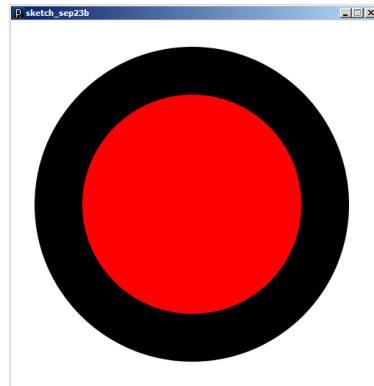
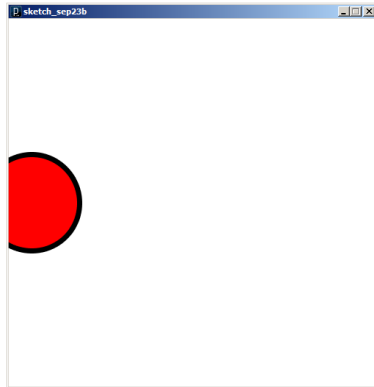
*When this code is run, the following is written to the Console:*

```
Setup: Start  
I'm running  
I'm running  
I'm running  
...
```

### **draw()**

Called directly after **setup()** and continuously executes the lines of code contained inside its block until the program is stopped or **noLoop()** is called.

## 5.2 Animation and motion

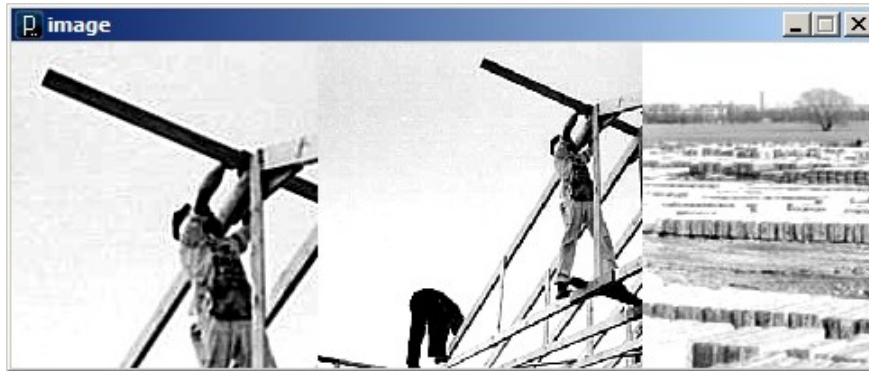


```
int x=0;

void setup() {
  size(500,500);
  stroke(0);
  smooth();
  frameRate(150); //speed
}

void draw() {
  x= x+1;
  if (x > 1500) {
    x=0;
  }
  background(255);
  strokeWeight(x/4);
  fill(255,0,0);
  ellipse(x, height/2, x+100,
x+100);
}
```

## 5.3 Media (Images and Fonts)



```
PImage img1;  
PImage img2;
```

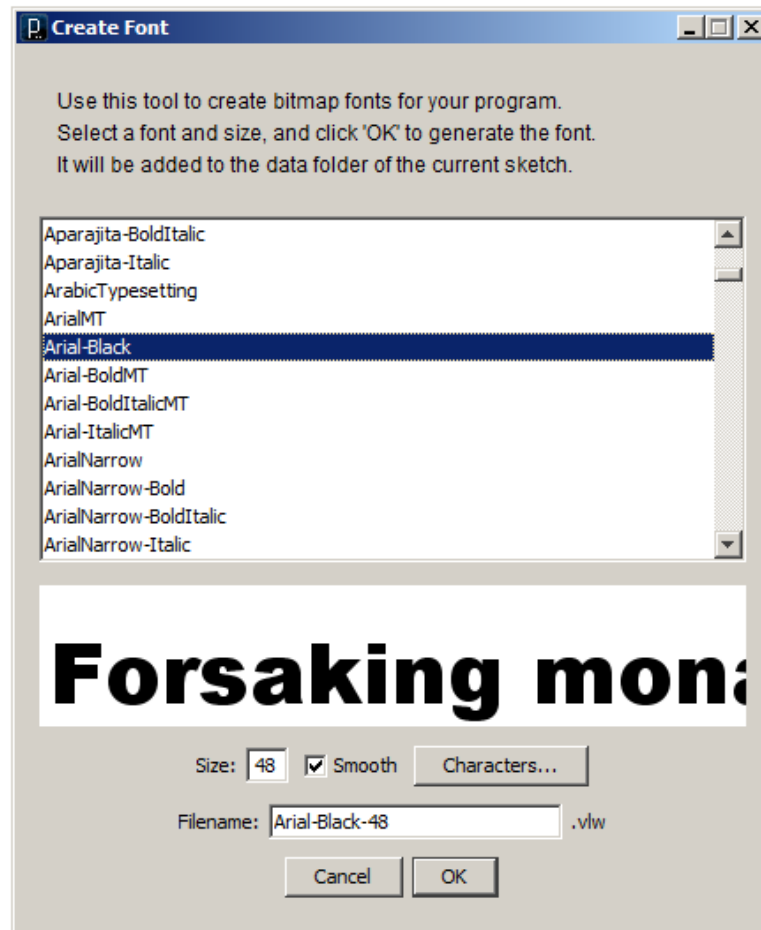
```
void setup() {  
  size(480, 180);  
  img1 = loadImage("image1.jpg");  
  img2 = loadImage("image2.JPG");  
}
```

```
void draw() {  
  background(0);  
  image(img1, -150, 0);  
  image(img1, 170, 0, 270, 180);  
  image(img2, 350, 0, 500, 180);  
}
```

```
PImage()  
PImage(width, height)  
PImage(width, height,  
format)  
PImage(img)
```



## 5.3 Media (Images and Fonts)



Sets the current font. The font must be loaded with **loadFont()** before it can be used. This font will be used in all subsequent calls to the **text()** function. If no **size** parameter is input, the font will appear at its original size (the size it was created at with the "Create Font..." tool) until it is changed with **textSize()**.

## 5.3 Media (Images and Fonts)



PFont  
loadFont()  
textFont()  
text(data, x, y) text(data, x, y, z)  
text(stringdata, x, y, width, height)  
text(stringdata, x, y, width, height, z)

```
size(200,100);  
background(0);
```

```
PFont font;  
// The font must be located in the sketch's  
// "data" directory to load successfully  
font = loadFont("Arial-Black-48.vlw");  
textFont(font, 47);  
text("Hello!", 10, height/2);  
textSize(14);  
text("Hello!", 10, 70);
```

## 5.4 Interaction (keyboard, touchscreen, multitouch handling)

Keyboard  
key  
keyCode  
keyPressed()  
keyReleased()

### keyPressed()

The boolean system variable `keyPressed` is true if any key is pressed and false if no keys are pressed.

```
void draw() {  
  if (keyPressed == true)  
  {  
    fill(0);  
  } else {  
    fill(255);  
  }  
  rect(25, 25, 50, 50);  
}
```

### key

The system variable `key` always contains the value of the most recent key on the keyboard that was used (either pressed or released).

```
//Click on the window to give it  
//focus and press the 'B' key
```

```
void draw() {  
  if(keyPressed) {  
    if (key == 'b' || key == 'B') {  
      fill(0);  
    }  
  } else {  
    fill(255);  
  }  
  rect(25, 25, 50, 50);  
}
```

From <http://processing.org/reference/keyPressed.html>

## 5.4 Interaction (keyboard, touchscreen, multitouch handling)

For compatibility, mouseX and mouseY work similar to the original API, and always contain the most recent mouse position. The following methods:

### Touchscreen

mousePressed()  
mouseReleased()  
mouseMoved()  
mouseDragged()

are also available.

In addition, the motionX, motionY, pmotionX, pmotionY, and motionPressure events can be used to track relative motion (the native way Android handles input). Unlike mouseX/mouseY, those variables are also float values.

For more information about handling input, check the wiki page:  
<http://wiki.processing.org/w/Android>

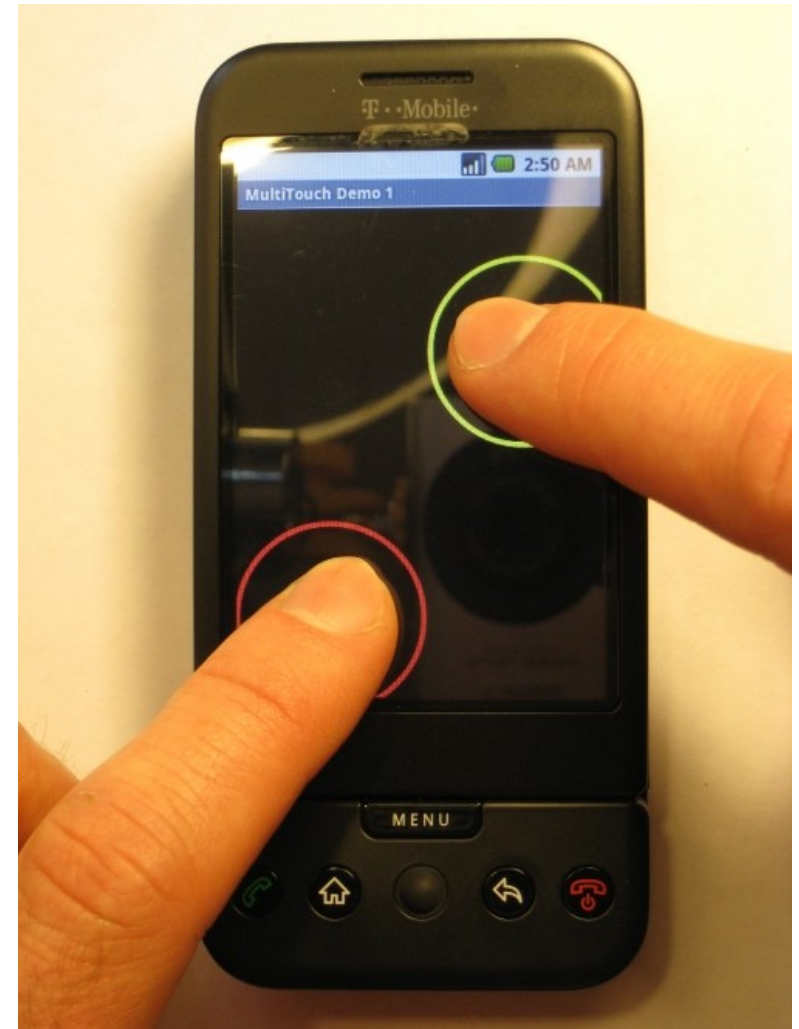
## 5.4 Interaction (keyboard, touchscreen, multitouch handling)

### Multitouch handling

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

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

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



## Code snippet showing how to use the `surfaceTouchEvent()` 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);
}
```

# 6. Extending Processing

Processing for Android is also extensible through user-contributed libraries. Still a few at the moment of this writing, hopefully many more will be created in coming months:

1) Ketai: library to handle sensors and camera:  
[processingandroid.org/Ketai](http://processingandroid.org/Ketai)  
<http://code.google.com/p/ketai/>

2) APWidgets: library to use native Android UI widgets in a Processing sketch  
<http://code.google.com/p/apwidgets/>

3) OscP5: library to send/receive OSC messages. Even though the package was created for PC/Mac Processing, it runs on Android as well.

**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.

More online information about OpenGL ES:

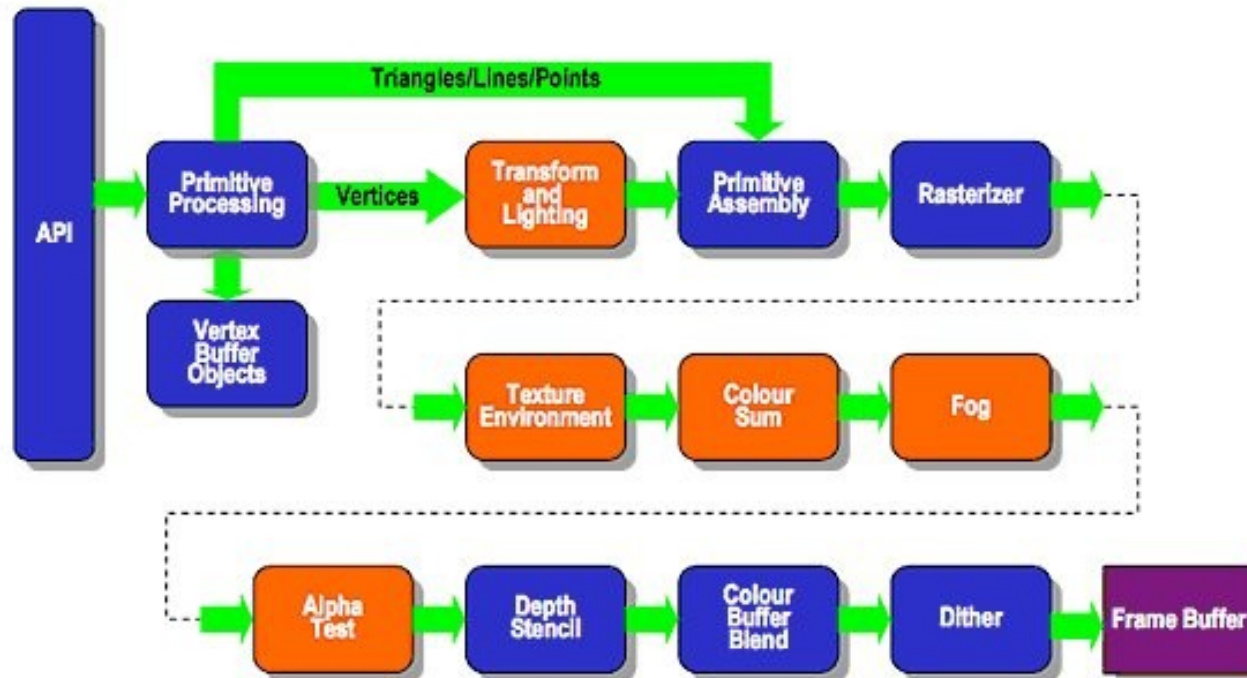
<http://developer.android.com/guide/topics/graphics/opengl.html>

<http://www.khronos.org/opengles/>



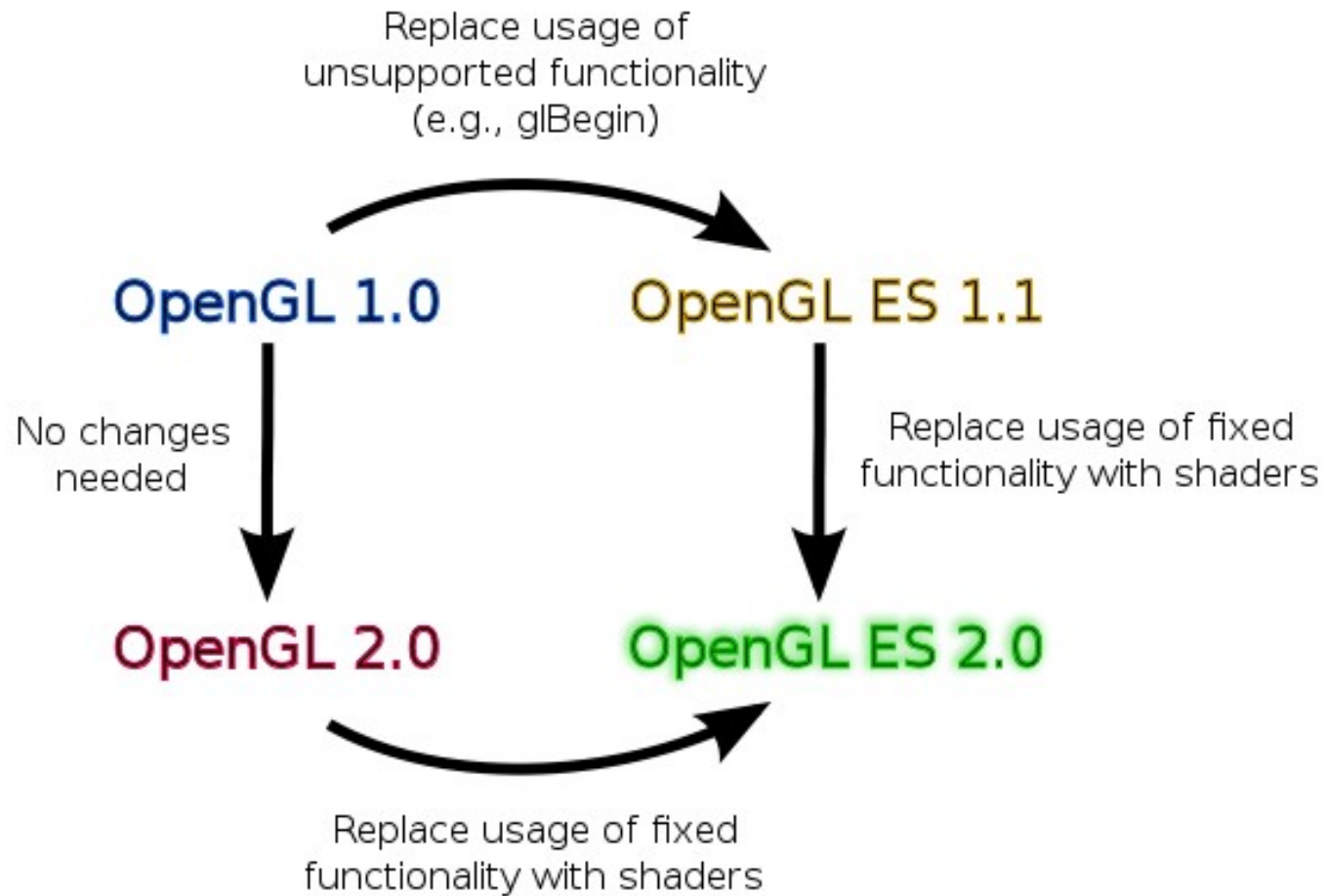
OpenGL ES is the standard 3D API the majority of the mobile platforms (Nokia, iPhone, etc.)

## The graphics pipeline on the GPU



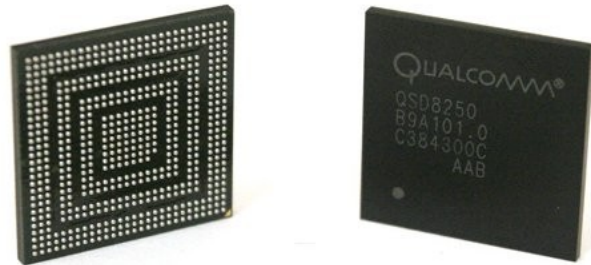
The graphics pipeline is the sequence of steps in the GPU from the data (coordinates, textures, etc) provided through the OpenGL ES API to the final image on the screen (or Frame Buffer)

## Relationship between OpenGL and OpenGL ES

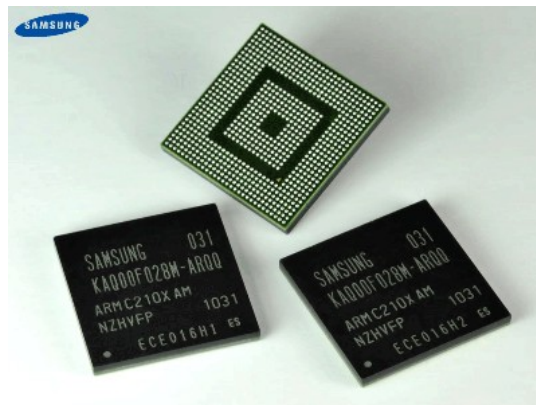


Source: <http://wiki.maemo.org/OpenGL-ES>

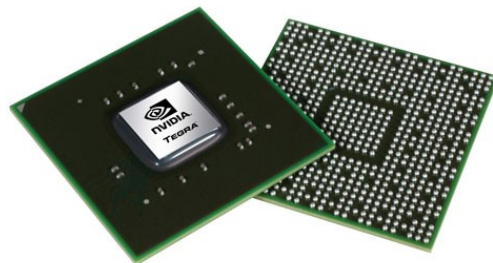
## Mobile GPUs



Qualcomm Adreno



PowerVR SGX



NVidia Tegra

## Comparison of current mobile GPUs

	Medium performance	High performance
Qualcomm Adreno	200 (Nexus 1, HTC Evo, Desire)	205 (Desire HD)
PowerVR SGX	SGX 530 (Droid, Droid 2, DroidX)	SGX 540 (Galaxy S, Vibrant, Captivate)
Nvidia Tegra		Tegra 250 (LG Optimus 2, many upcoming tablets)

Useful websites for benchmark information about mobile GPUs:

<http://smartphonebenchmarks.com/>

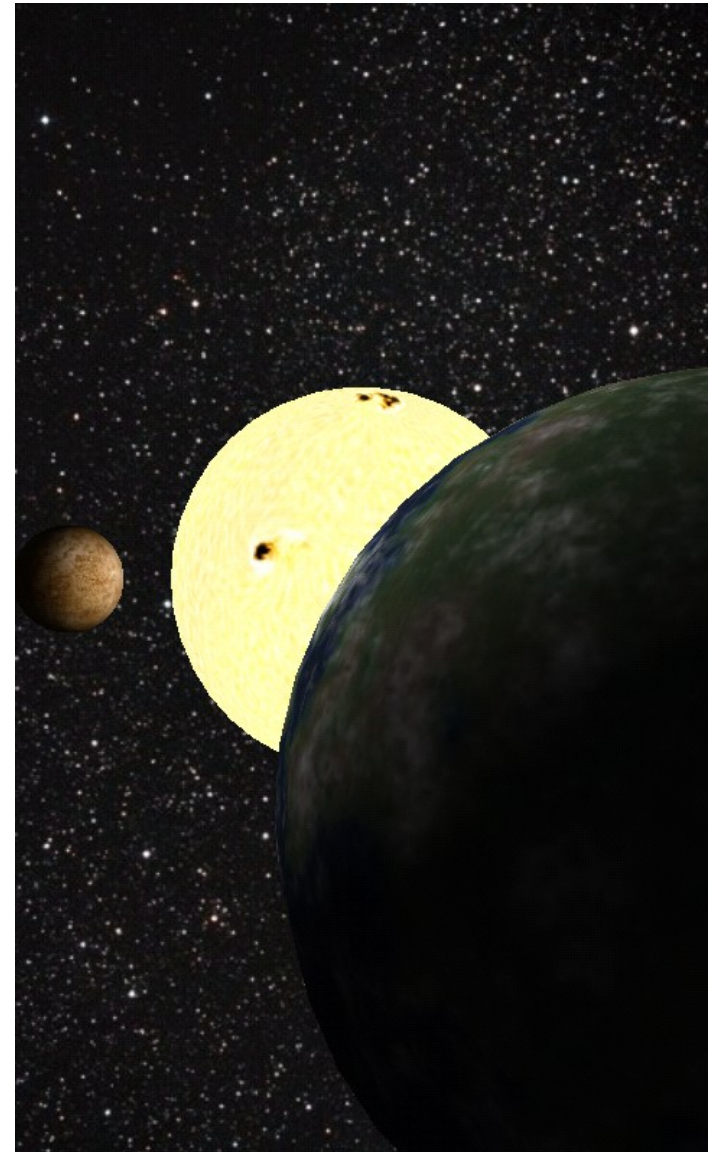
[http://www.glbenchmark.com/latest\\_results.jsp?](http://www.glbenchmark.com/latest_results.jsp?)

## Hardware requirements for 3D in Processing for Android

1. In principle, any GPU that supports OpenGL ES 1.1
1. GPUs such as the Adreno 200 or PowerVR SVG 540/530 are recommended.
1. Older GPUs found on devices such as the G1 might work, but the performance is limited.
1. 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 Android 3D (A3D) renderer in Processing



## Processing Renderers

1. In Processing for Android there is no need to use OpenGL ES directly (although it is possible).
1. The drawing API in Processing uses OpenGL internally when selecting the A3D (Android 3D) renderer.
1. The renderer in Processing is the module that executes all the drawing commands.
1. During the first part of this workshop we used the A2D renderer, which only supports 2D drawing.
1. 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`
6. If no renderer is specified, then A2D is used by default.

## What A3D is not:

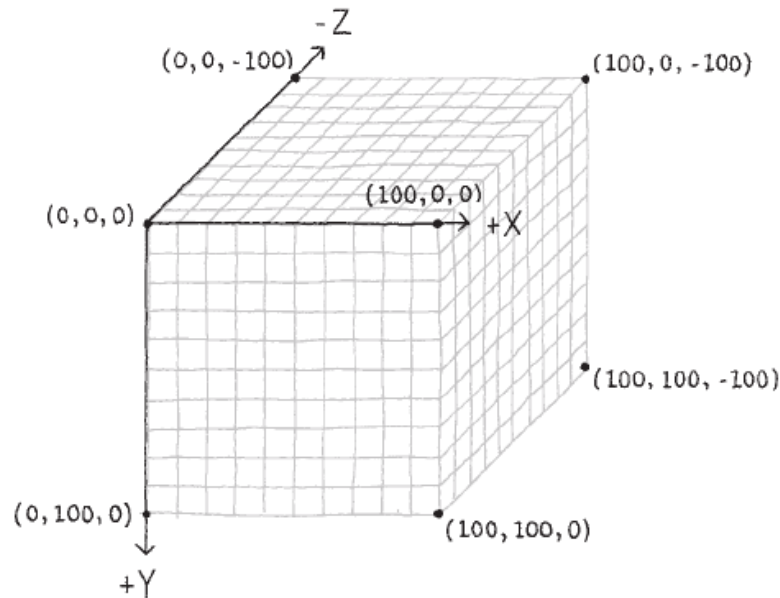
- 1) A game engine (although it can be used to create games)
- 2) A scene-graph manager (but one it could be built on top of it)

Let's just say that A3D is a minimal renderer for 3D graphics that follows the emphasis on simplicity and ease of use of Processing. Although is “minimal”, it offers some advanced functionality such as offscreen rendering, particle systems, etc.

## **What A3D currently offers:**

- 1) A simple API for constructing 3D shapes, which extends Processing's 2D mode
- 2) Camera, perspective, basic lighting, geometrical transformations
- 3) Pixel operations, texture blending and multitexturing
- 4) Offscreen rendering based on FBOs
- 5) 3D models based on VBOs
- 6) OpenGL-accelerated fonts
- 7) Model recording (direct mode drawing + VBOs)
- 8) Sprite-based particle systems

# 8. Geometrical transformations



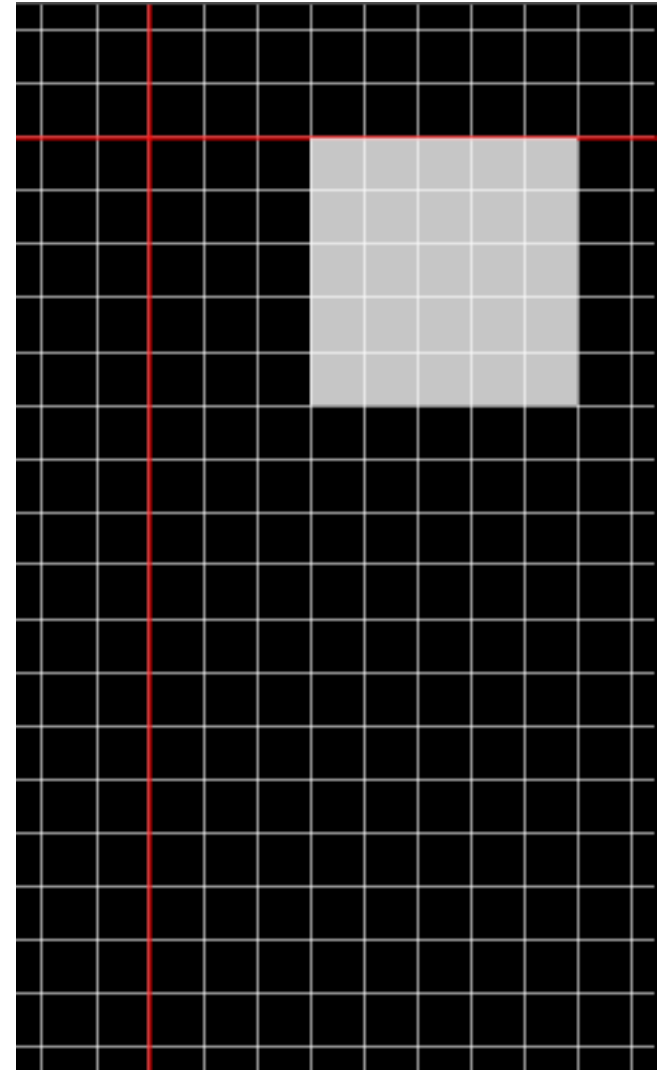
Casey Reas and Ben Fry.  
Getting Started with Processing.  
O'Really Media, 2010

1. The coordinate system in Processing is defined with the 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.

## 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);  
}
```



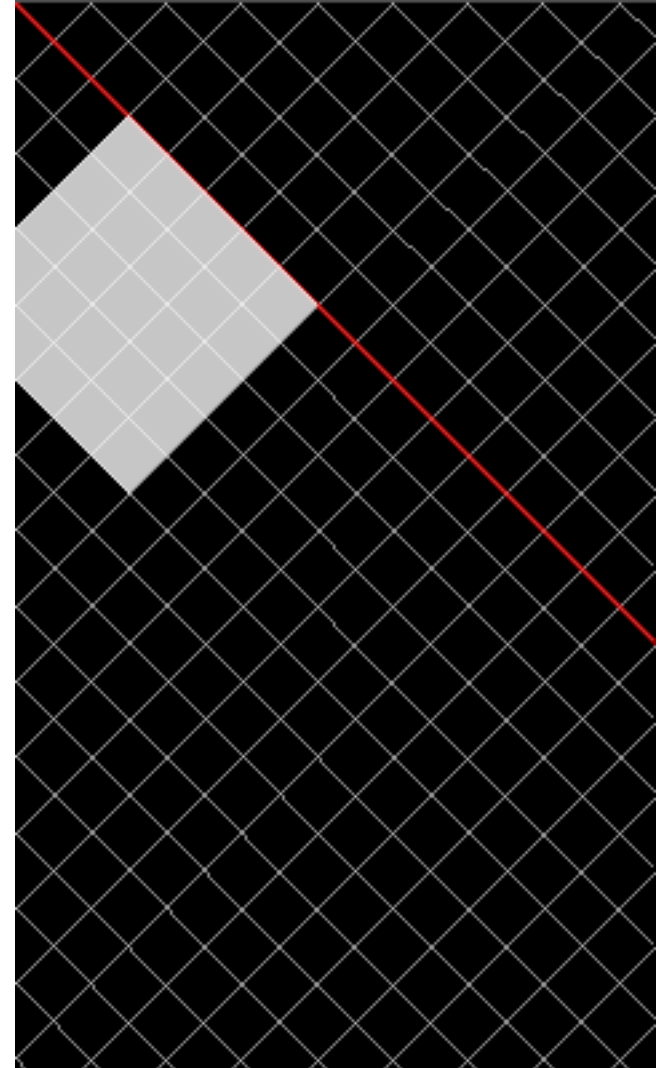
## Rotations

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(angle, vx, vy, vz)
```

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

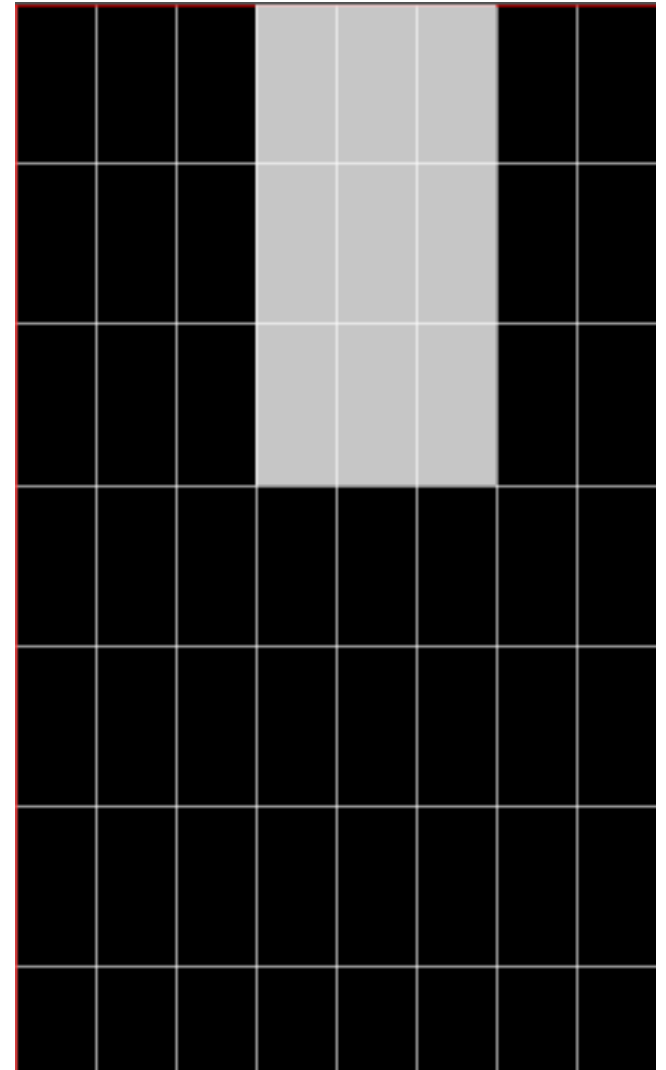
void draw() {
  background(0);
  rotateZ(PI / 4);
  noStroke();
  fill(255, 200);
  rect(60, 0, 100, 100);
}
```



## Scaling

Scaling can be uniform (same scale factor on each axis) or not, since the `scale(sx, sy, sz)` function 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);  
}
```

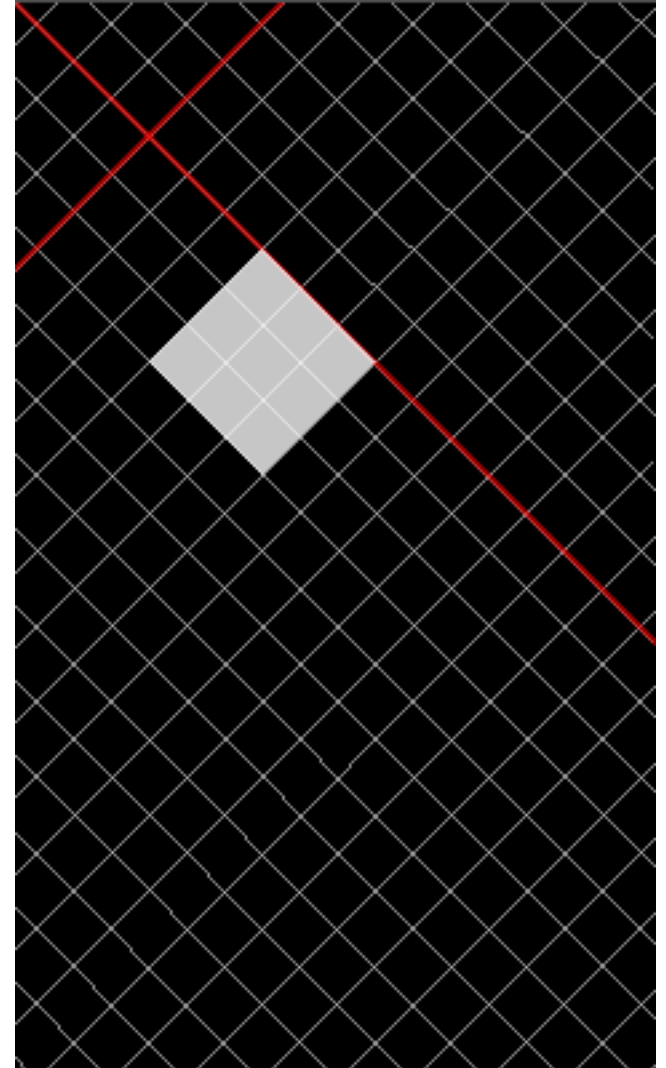




## Just a couple of important points about geometrical transformations...

1. By combining `translate()` with `rotate()`, the rotations can be applied around any desired point.
2. The order of the transformations is important

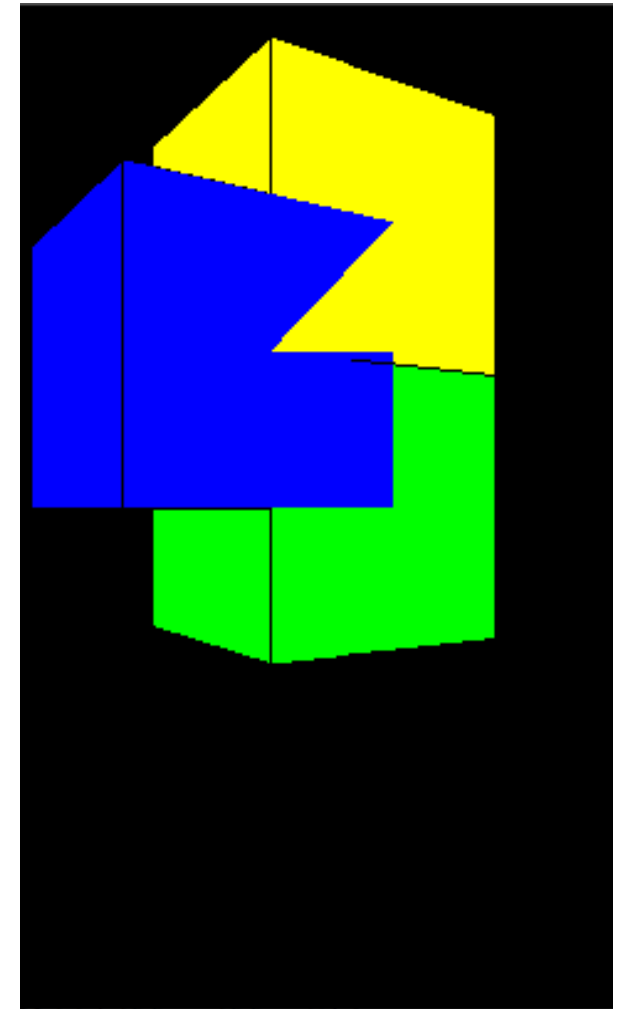
```
void setup() {  
  size(240, 400, A3D);  
  stroke(255, 150);  
}  
  
void draw() {  
  background(0);  
  translate(50, 50, 0);  
  rotateZ(PI / 4);  
  noStroke();  
  fill(255, 200);  
  rect(60, 0, 60, 60);  
}
```



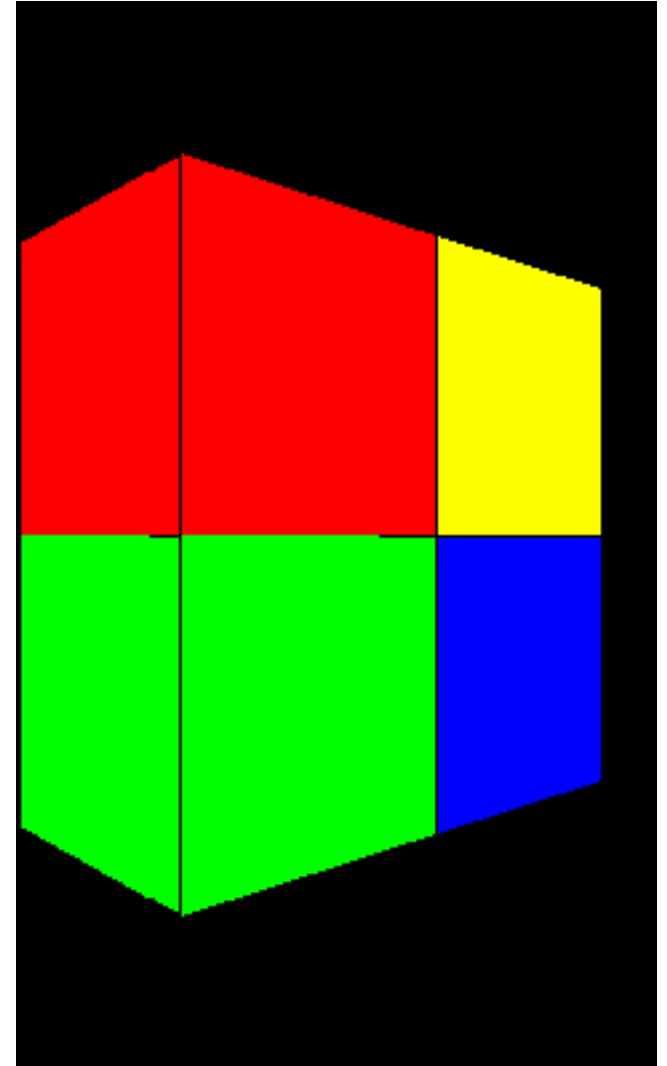
## The transformation stack

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

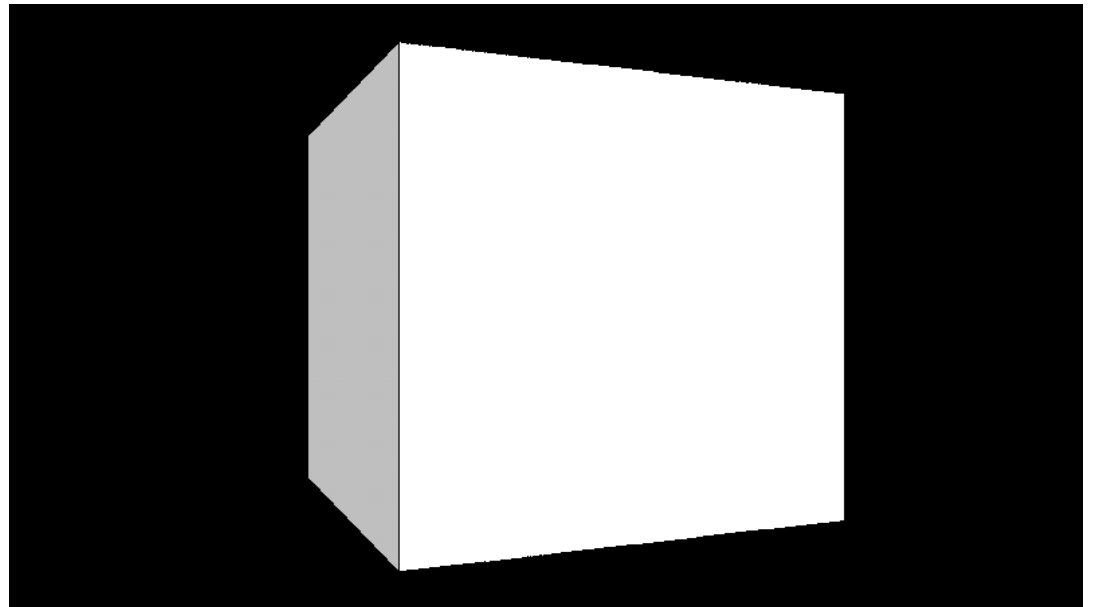
```
void setup(){
  size(240, 400, A3D);
}
void draw(){
  background(0);
  translate(width/2, height/2);
  rotateY(frameCount*PI/60);
  translate(-50, -50);
  fill(255, 0, 0);
  box(100, 100, 100);
  translate(50, -50);
  fill(255, 255, 0);
  box(100, 100, 100);
  translate(-50, 50);
  fill(0, 0, 255);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



```
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);
  popMatrix();
}
```

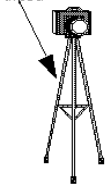
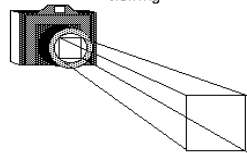
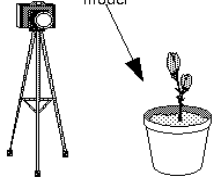
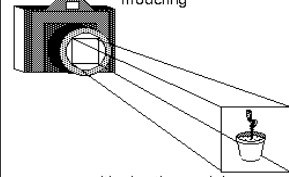

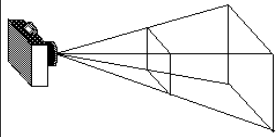




Lets quickly code up a 3D “hello world” example with A3D...



# 9. Camera and perspective

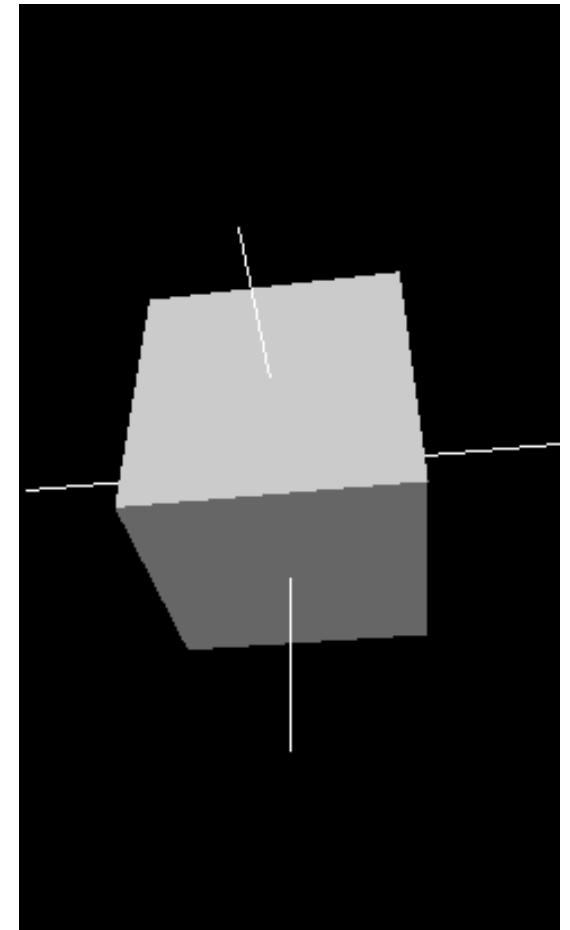
1. Configuring the view of the scene in A3D requires setting the camera location and the viewing volume.
2. This can be compared with setting a physical camera in order to take a picture:

With a Camera	With a Computer	
 <p>tripod</p>	 <p>viewing</p> <p>positioning the viewing volume in the world</p>	<pre>camera(eyeX, eyeY, eyeZ,       centerX, centerY, centerZ,       upX, upY, upZ)</pre>
 <p>model</p>	 <p>modeling</p> <p>positioning the models in the world</p>	
 <p>lens</p>	 <p>projection</p> <p>determining shape of viewing volume</p>	<pre>perspective(fov, aspect, zNear, zFar) ortho(left, right, bottom, top, near, far)</pre>
 <p>photograph</p>	 <p>viewport</p>	<p>(image from the OpenGL Red Book, first edition)</p>

## Camera placement

1. 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)`
2. 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`.

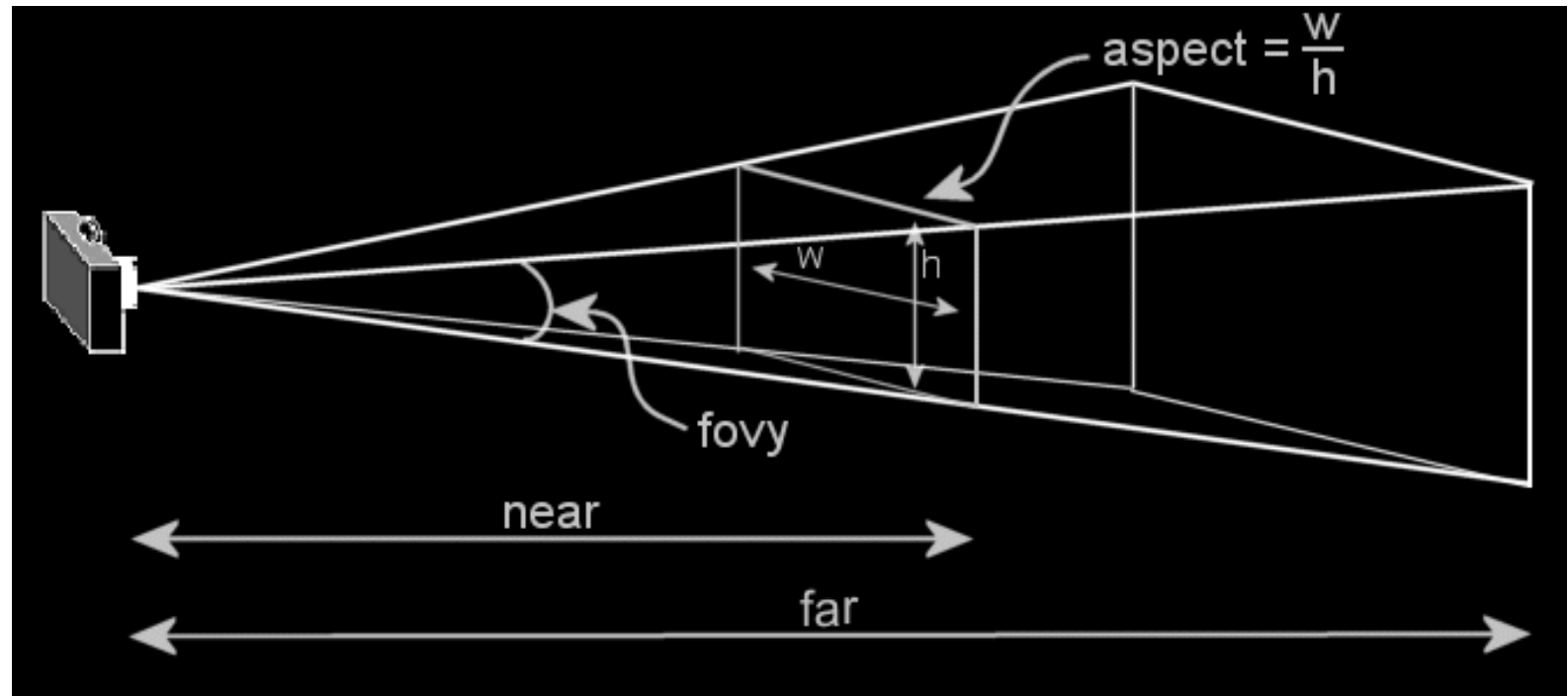
```
void setup() {  
  size(240, 400, A3D);  
  fill(204);  
}  
  
void draw() {  
  lights();  
  background(0);  
  camera(30.0, mouseY, 220.0,  
        0.0, 0.0, 0.0,  
        0.0, 1.0, 0.0);  
  noStroke();  
  box(90);  
  stroke(255);  
  line(-100, 0, 0, 100, 0, 0);  
  line(0, -100, 0, 0, 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/Lessons/Lesson1.php>

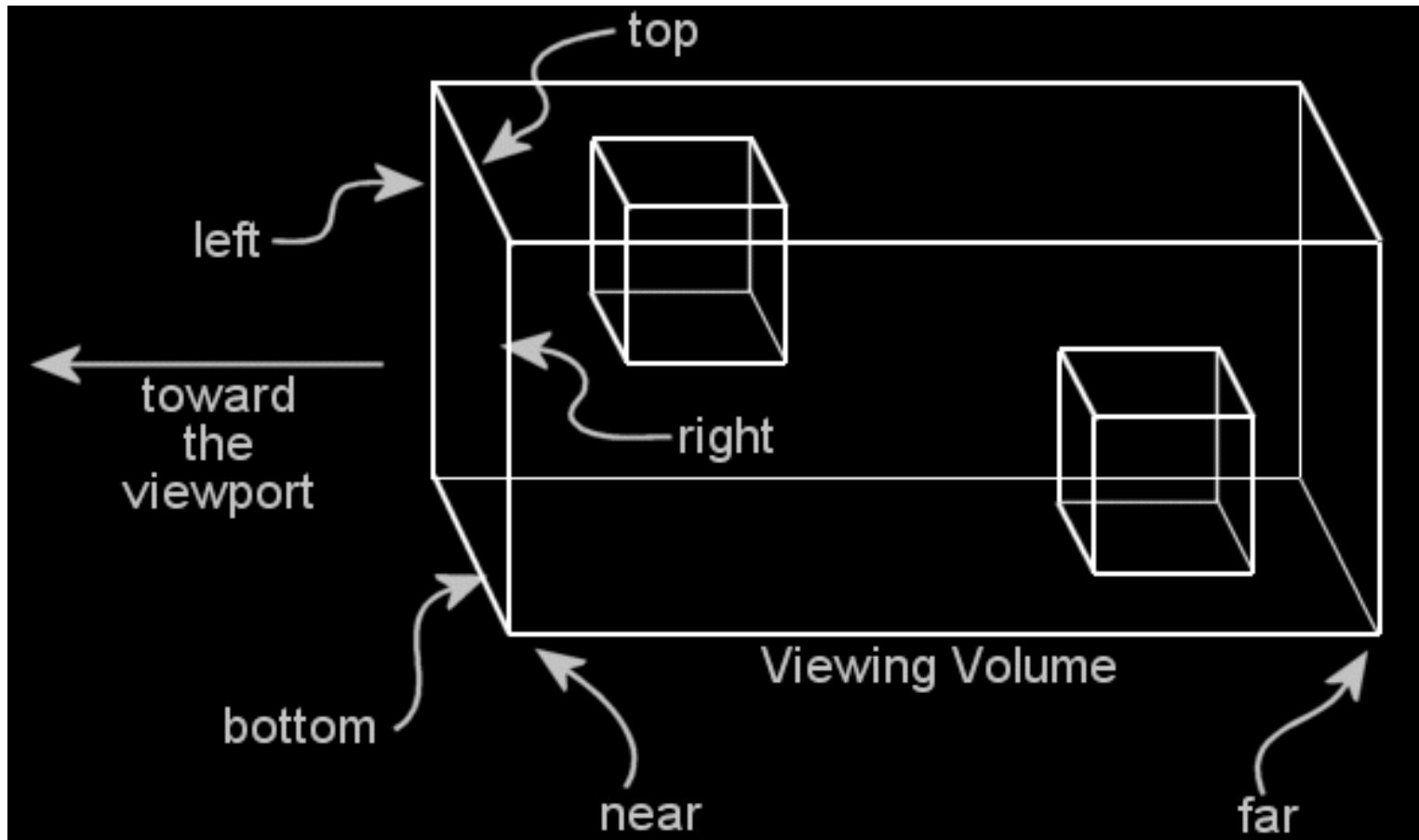


`perspective(fov, aspect, zNear, zFar)`

`perspective(PI/3.0, width/height, cameraZ/10.0, cameraZ*10.0)` where cameraZ is  $((\text{height}/2.0) / \tan(\text{PI} * 60.0 / 360.0))$  (default values)

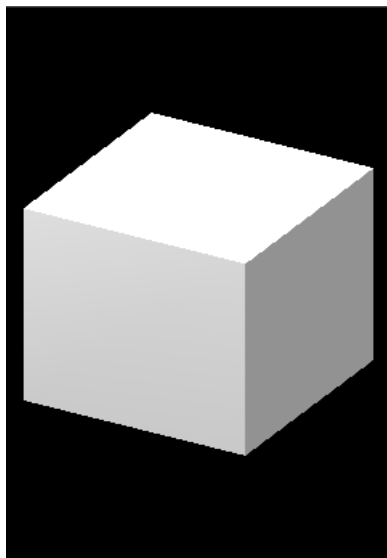
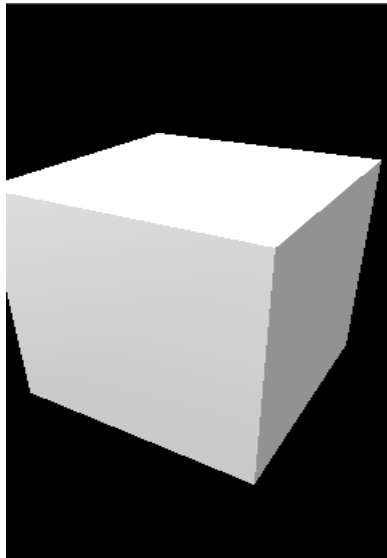
## Orthographic view

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) (default)
```





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

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

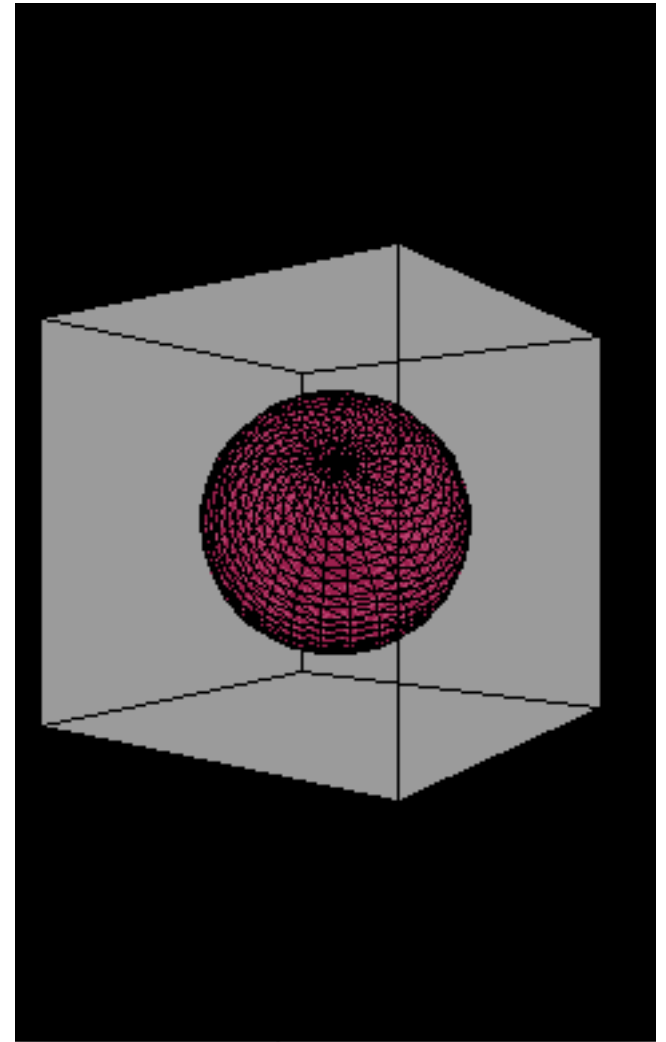
  if(mousePressed) {
    float fov = PI/3.0;
    float cameraZ = (height/2.0) / tan(PI * fov / 360.0);
    perspective(fov, float(width)/float(height),
               cameraZ/2.0, cameraZ*2.0);
  } else {
    ortho(-width/2, width/2, -height/2, height/2, -10, 10);
  }

  translate(width/2, height/2, 0);
  rotateX(-PI/6);
  rotateY(PI/3);
  box(160);
}
```

# 10. Creating 3D objects

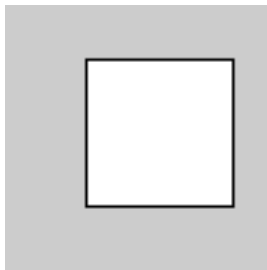
A3D provides some functions for drawing predefined 3D primitives:  
sphere(r), box(w, h, d)

```
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();  
}
```



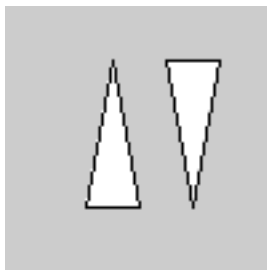
## beginShape()/endShape()

1. 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)
2. This functionality is already present in A2D, with the difference that in A3D we can specify vertices with z coordinates.



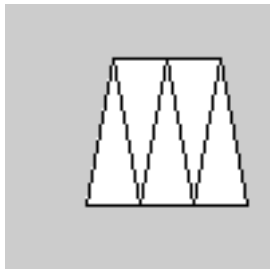
```
beginShape();  
vertex(30, 20, 0);  
vertex(85, 20, 0);  
vertex(85, 75, 0);  
vertex(30, 75, 0);  
endShape(CLOSE);
```

**Closed polygon**



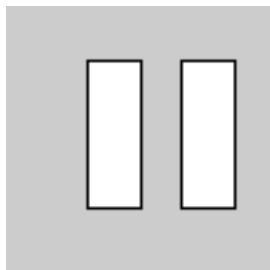
```
beginShape(TRIANGLES);  
vertex(30, 75, 0);  
vertex(40, 20, 0);  
vertex(50, 75, 0);  
vertex(60, 20, 0);  
vertex(70, 75, 0);  
vertex(80, 20, 0);  
endShape();
```

**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



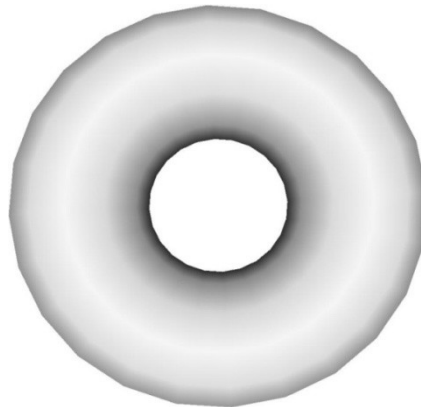
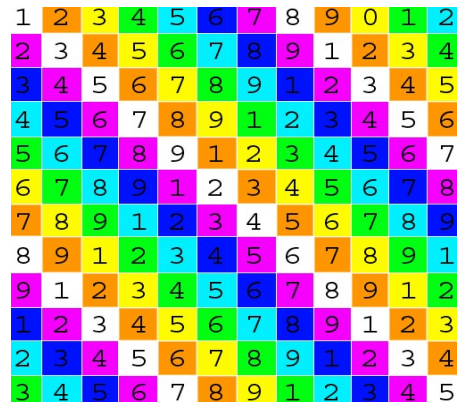
```
beginShape (QUADS);  
vertex(30, 20, 0);  
vertex(30, 75, 0);  
vertex(50, 75, 0);  
vertex(50, 20, 0);  
vertex(65, 20, 0);  
vertex(65, 75, 0);  
vertex(85, 75, 0);  
vertex(85, 20, 0);  
endShape();
```

## Individual quads

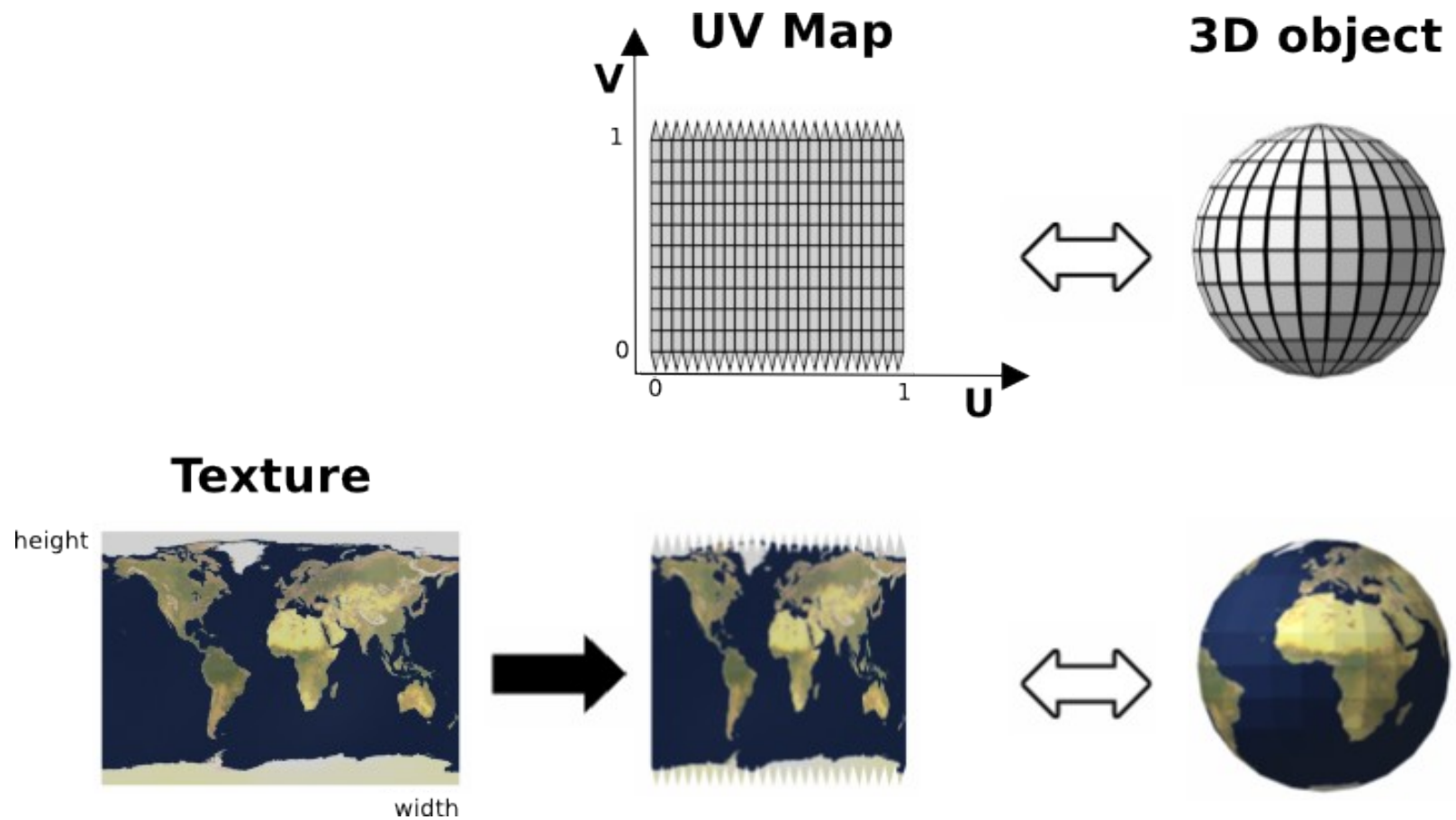
Check the Processing reference for more details:  
[http://processing.org/reference/beginShape\\_.html](http://processing.org/reference/beginShape_.html)

# 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](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

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



**Ambient**



**Diffuse**



**Specular**

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

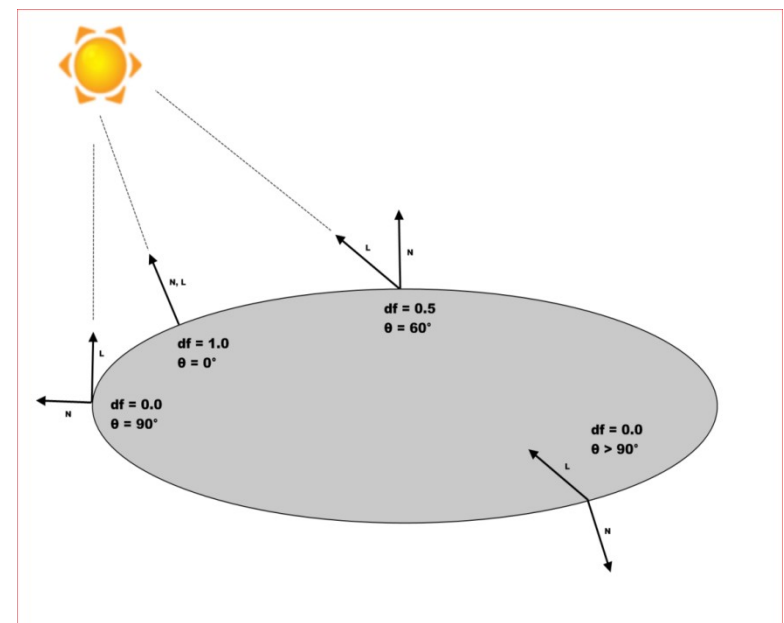
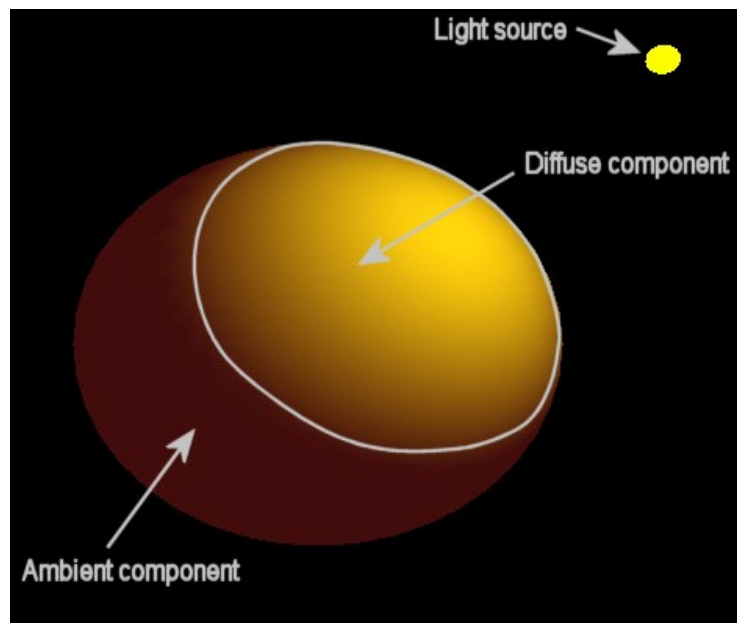
Some more good resources about lights in OpenGL:

<http://jerome.jouvie.free.fr/OpenGL/Lessons/Lesson6.php>

<http://jerome.jouvie.free.fr/OpenGL/Tutorials/Tutorial12.php> - [Tutorial15.php](http://jerome.jouvie.free.fr/OpenGL/Tutorials/Tutorial15.php)

[http://www.sjbaker.org/steve/omniv/opengl\\_lighting.html](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>  
<http://jerome.jouvie.free.fr/OpenGl/Lessons/Lesson6.php>

## 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 light  
x, 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 gentle angle. After hitting a surface, a directional lights scatters in all directions.

`directionalLight(v1, v2, v3, nx, ny, nz)`  
v1, v2, v3: rgb color of the light  
nx, ny, and nz the direction the light is facing.

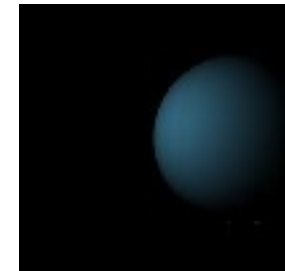


**Point:** Point light irradiates from a specific position.

`pointLight(v1, v2, v3, x, y, z)`

`v1, v2, v3`: rgb color of the light

`x, y, z` position:



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

`spotLight(v1, v2, v3, x, y, z, nx, ny, nz, angle, concentration)`

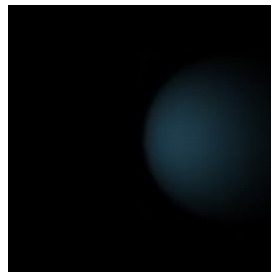
`v1, v2, v3`: rgb color of the light

`x, y, z` position:

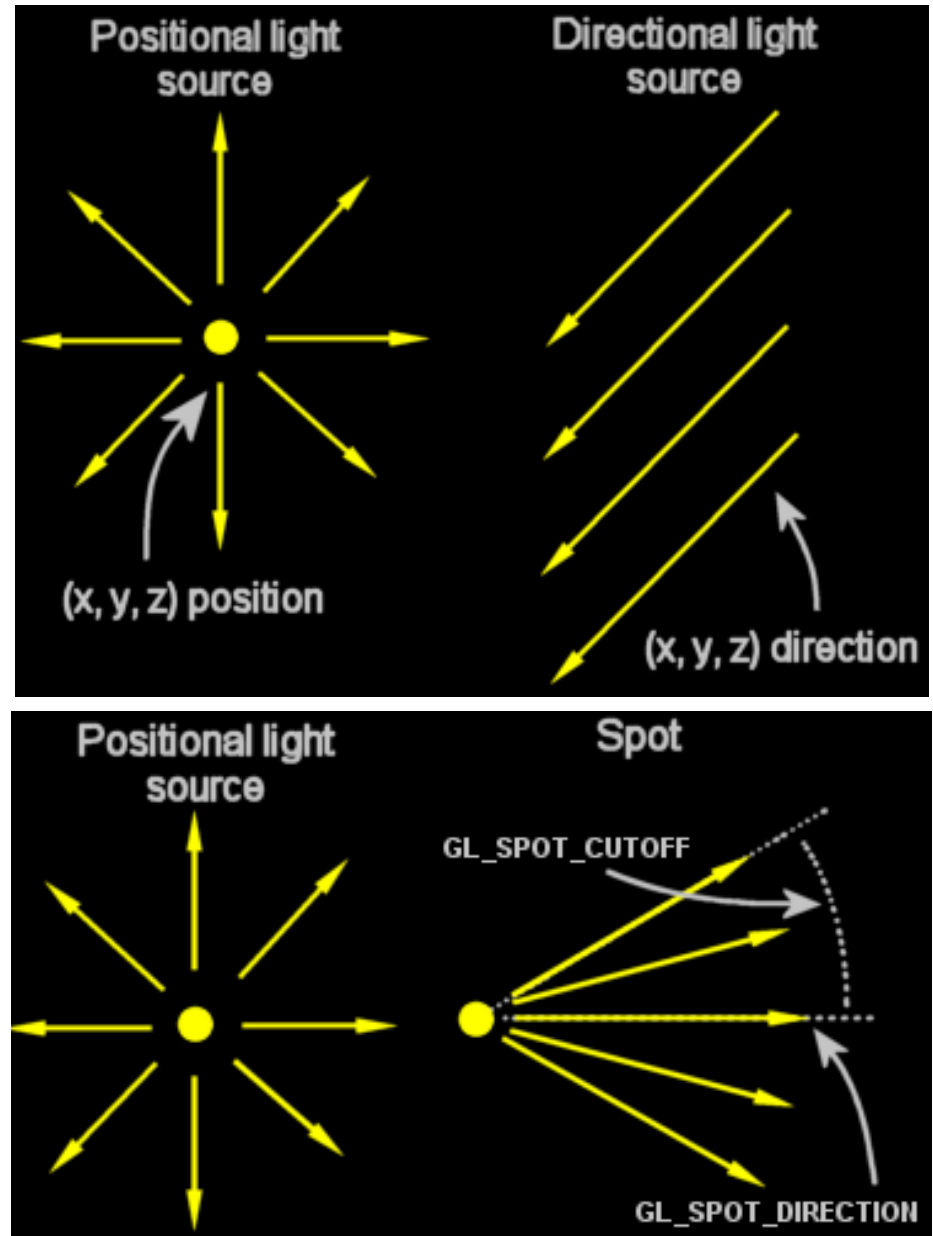
`nx, ny, nz` specify the direction or light

`angle` float: angle the spotlight cone

`concentration`: exponent determining the center bias of the cone



There plenty of information online about topics such as opengl lighting, which can be translated very directly into A3D's terminology.

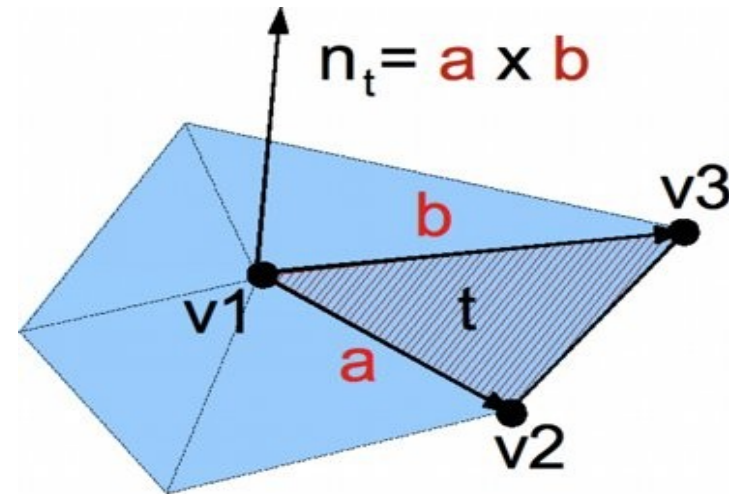


<http://jerome.jouvie.free.fr/OpenGL/Lessons/Lesson6.php>

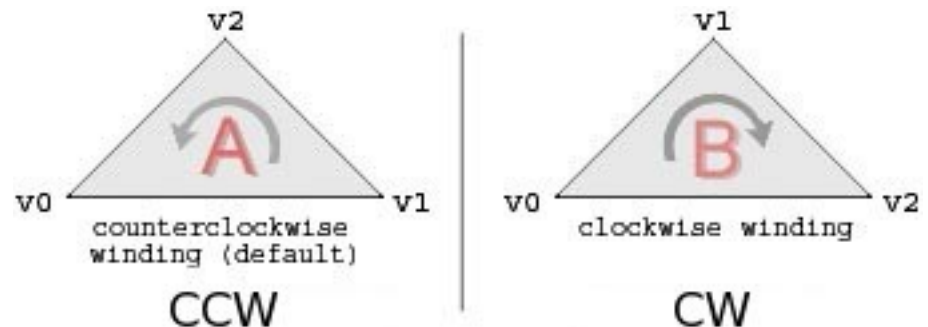
**Normals:** each vertex needs to have a normal defined so the light calculations can be performed correctly

```
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.



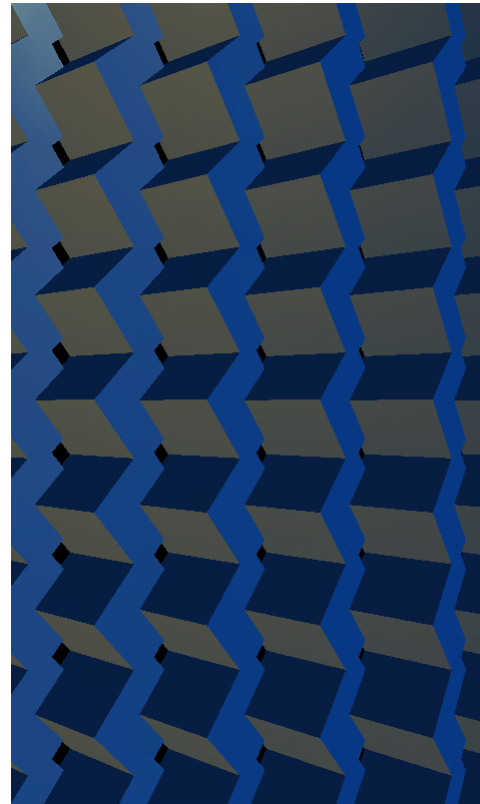
A3D can automatically calculate the normals for you, by setting the auto normal mode to true:

```
// Instantiate cubes, passing in random vals for size and position
for (int i = 0; i < cubes.length; i++){
    cubes[i] = new Cube(int(random(-10, 10)), int(random(-10, 10)),
        int(random(-10, 10)), int(random(-140, 140)), int(random(-140, 140)),
        int(random(-140, 140)));
}

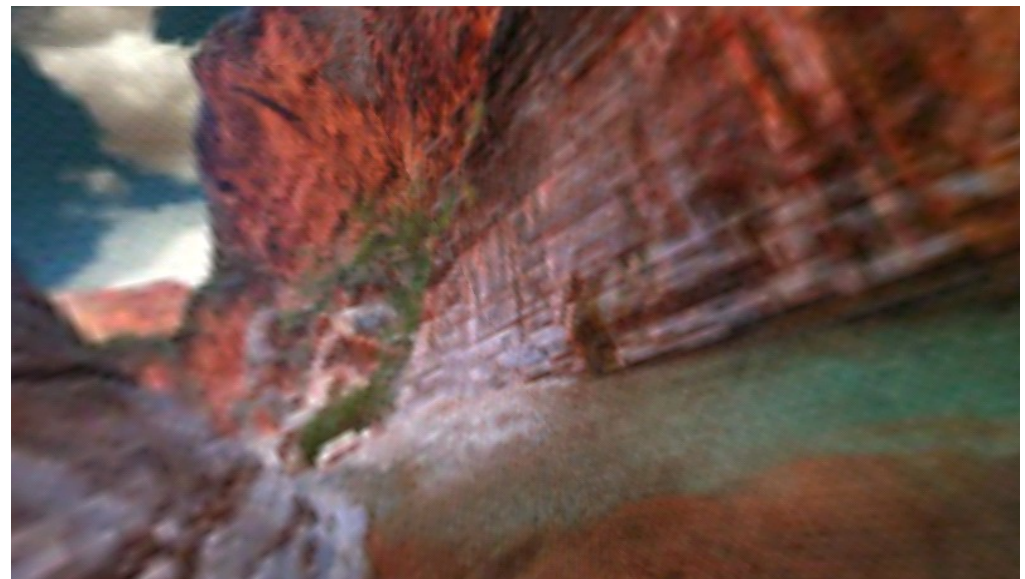
// Automatic normal calculation can be turned on/off.
autoNormal(true);
...
```

Note that this calculation might not be very accurate for complex surfaces!





So we have already shape  
creation, lights, textures,  
camera.  
We can already do quite a  
few things...



# 11. 3D Text

Text in A3D works exactly the same as in A2D:

1. load/create fonts with loadFont/createFont
2. set current font with textFont
3. 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);
}
```



1. The main addition in A3D is that text can be manipulated in three dimensions.
2. Each string of text we print to the screen with text() is contained in a rectangle that we can rotate, translate, scale, etc.
3. 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);

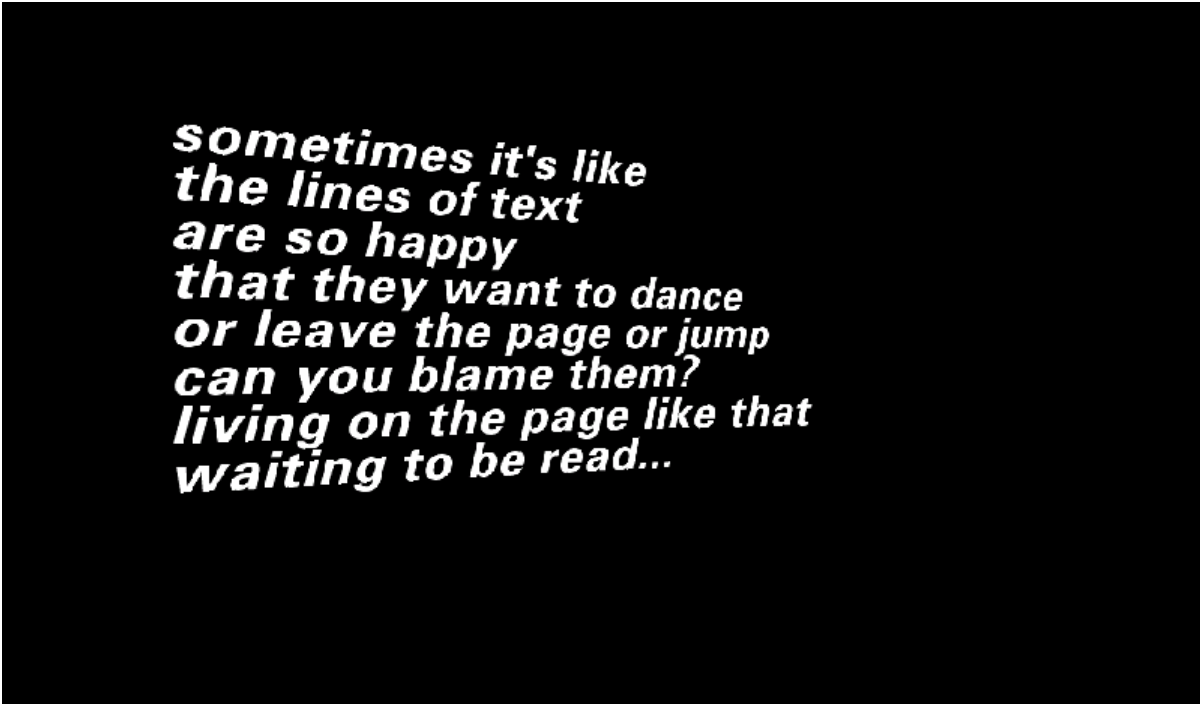
  translate(width/2, height/2, 0);

  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();
  }
}

```



Kinetic type example



*sometimes it's like  
the lines of text  
are so happy  
that they want to dance  
or leave the page or jump  
can you blame them?  
living on the page like that  
waiting to be read...*

# 12. Some special topics

## 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. We will see more of this at the end.

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

The `createGraphics()` method returns a complete rendering surface that includes RGBA color buffer as well as Z and stencil buffers.

The bit depth of these buffers depend on the configuration of the drawing surface.

By default, A3D lets Android to choose the configuration, but we can optionally force a specific one by using the `sketchColordepth` and `sketchTranslucency` methods:

```
String sketchColordepth() {
    return "8:8:8:8:16:0";
}

boolean sketchTranslucency() {
    return true;
}
```

The string returned by `sketchColordepth` must be in the format R:G:B:A:D:S, where R, G, B, A, D and S are the bit depths for the Red, Green, Blue and Alpha channels of the color buffer, Z and D the bits depths of the Z and stencil buffers.

## Blending

The `blend(int mode)` method allows to set the desired blending mode that will be used to mix a color to be written to a pixel in screen with the color that the pixel already has.

Currently supported blending modes in A3D are:

REPLACE  
BLEND  
ADD  
SUBTRACT  
LIGHTEST  
DARKEST  
DIFFERENCE  
EXCLUSION  
MULTIPLY  
SCREEN

These modes are described in detail at  
[http://processing.org/reference/blend\\_.html](http://processing.org/reference/blend_.html)



## Multitexturing (part 1)

The `texture()` and `vertex()` methods are overloaded to accept more than one texture or texture coordinates:

```
beginShape();  
texture(tex0, tex1);  
...  
vertex(x, y, z, u0, v0, u1, v1);  
...
```

The blending mode for multitextures is set with the `textureBlend(mode)` function, which currently accepts the following modes:

```
REPLACE,  
BLEND  
ADD  
SUBTRACT  
MULTIPLY
```

## Pixel operations

There are several methods to manipulate pixels directly, and then transfer this information back and forth between CPU arrays and GPU textures.

```
img1 = loadImage("image1.jpg");
img1.resize(64, 64);

int w = 230;
int h = 230;
img3 = createImage(w, h, ARGB);
int[] pix = new int[w * h];
for (int i = 0; i < h; i++) {
    for (int j = 0; j < w; j++) {
        if (i < h / 2) {
            if (j < w / 2) pix[i * w + j] = 0xFFFF0000;
            else pix[i * w + j] = 0xFF00FF00;
        } else {
            if (j < w / 2) pix[i * w + j] = 0xFF0000FF;
            else pix[i * w + j] = 0xFFFFFFFF00;
        }
    }
}

img3.loadPixels(); // Enables use of pixel array.
img3.getTexture().set(pix); // Copies pix array to texture.
img3.updateTexture(); // Copies texture to pixel array.
for (int i = h / 2 - 20; i < h / 2 + 20; i++) {
    for (int j = w / 2 - 20; j < w / 2 + 20; j++) {
        img3.pixels[i * w + j] = 0xFFFFFFFF;
    }
}
img3.updatePixels(w / 2 - 20, h / 2 - 20, 40, 40);

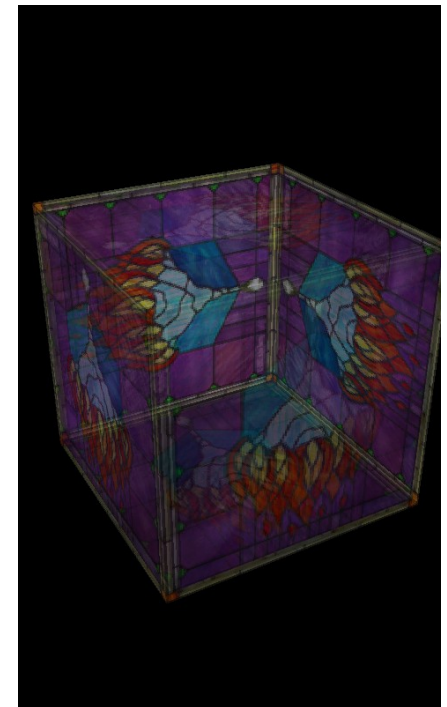
img2 = createImage(w, h, ARGB);
img2.getTexture().set(img3.pixels);
```

## Mixing A3D code and OpenGL ES code

Within Processing we can safely mixed standard A3D code with OpenGL calls, once we get a handle to the gl object. The GL calls must be enclosed by beginGL/endGL, which ensures that the OpenGL states returns to what A3D expects after using OpenGL directly:

```
PGraphicsAndroid3D a3d = (PGraphicsAndroid3D)g;  
GL10 gl = a3d.beginGL();  
...  
a3d.endGL();
```

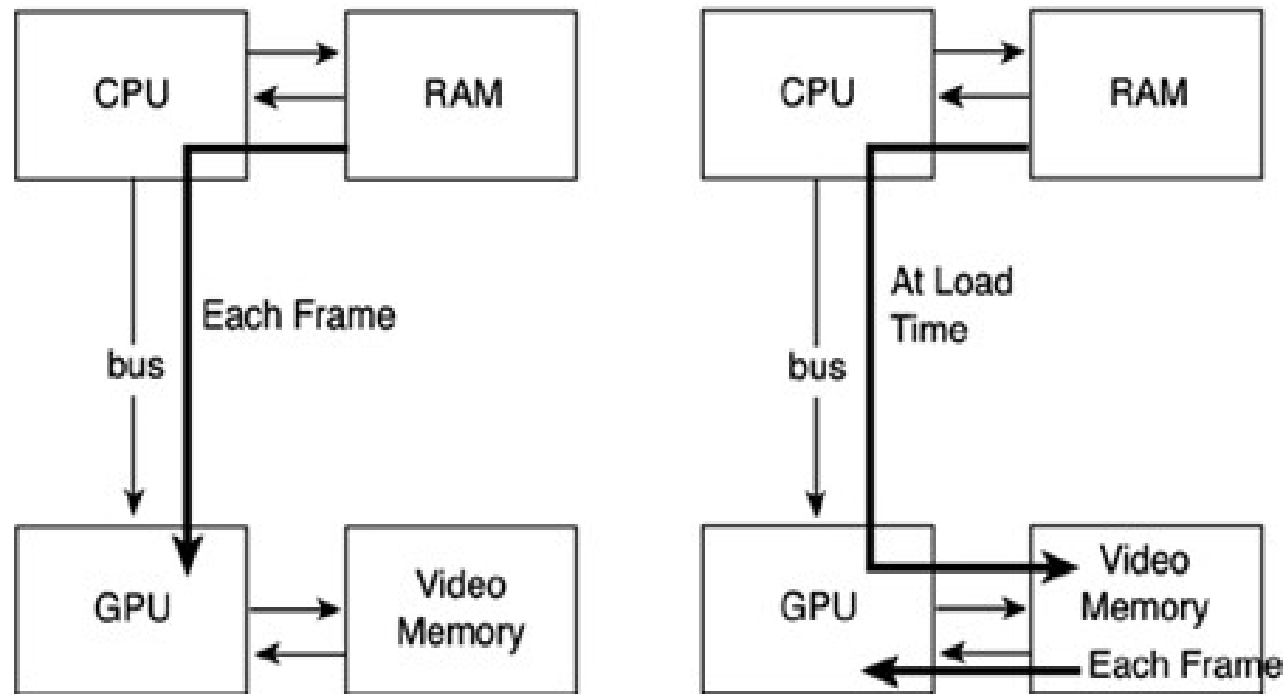
Nehe example



# 13 Models: the PShape3D class

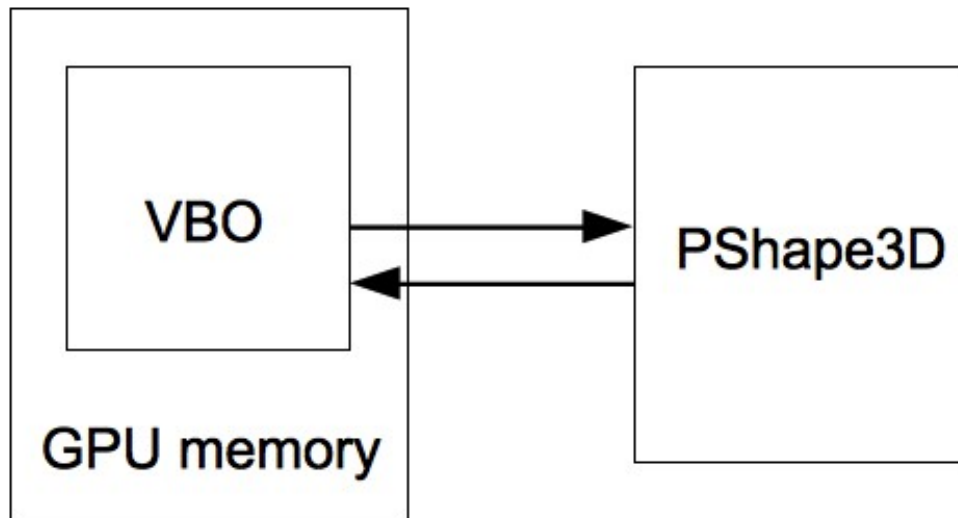
## Vertex Buffer Objects

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



For a good tutorial about VBOs, see this page:  
[http://www.songho.ca/opengl/gl\\_vbo.html](http://www.songho.ca/opengl/gl_vbo.html)

## The PShape3D class in A3D encapsulates VBOs



1. 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.
2. PShape3D has to be created with the total number of vertices known beforehand. Resizing is possible, but slow.
3. How vertices are interpreted depends on the geometry type specified at creation (POINT, TRIANGLES, etc), in a similar way to beginShape()/endShape()
4. Vertices in a PShape3D can be organized 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

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

### Creation

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

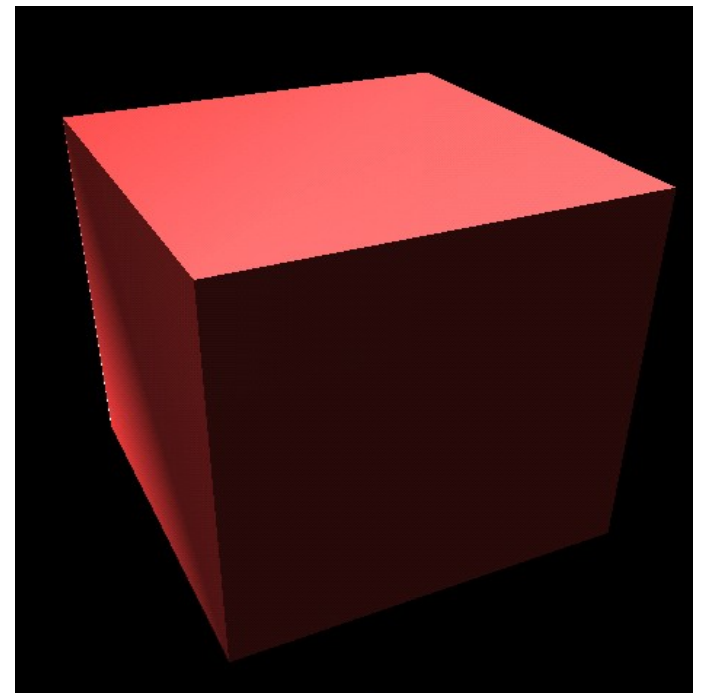
cube.loadVertices();
cube.set(0, -100, +100, -100);
cube.set(1, -100, -100, -100);
...
cube.updateVertices();

cube.loadColors();
cube.set(0, color(200, 50, 50, 150));
cube.set(1, color(200, 50, 50, 150));
...
cube.updateColors();

cube.loadNormals();
cube.set(0, 0, 0, -1);
cube.set(1, 0, 0, -1);
...
cube.updateNormals();
```

### Drawing

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

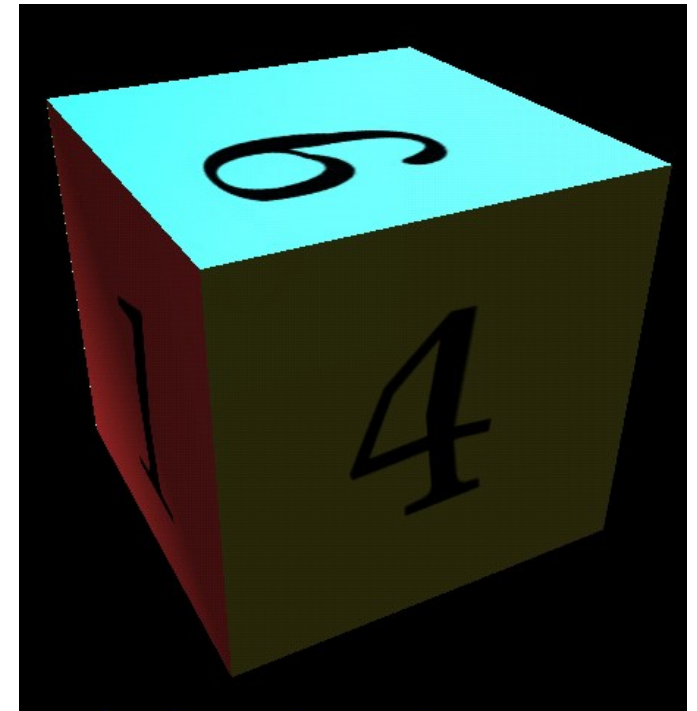


1. A PShape3D can be textured with one or more images.
2. The vertices can be hierarchically organized in groups, which allows to assign a different textures to each group.
3. Groups also facilitate the assignment of colors, normals, and styles, as well as the application of geometrical transformations (although this feature won't be available until release 0193 of Processing).

```
cube.addChild("Face 0", 0, 5);
cube.addChild("Face 1", 6, 11);
cube.addChild("Face 2", 12, 17);
cube.addChild("Face 3", 18, 23);
cube.addChild("Face 4", 24, 29);
cube.addChild("Face 5", 30, 35);

cube.setNormal(0, 0, 0, -1);
cube.setNormal(1, +1, 0, 0);
cube.setNormal(2, 0, 0, +1);
cube.setNormal(3, -1, 0, 0);
cube.setNormal(4, 0, +1, 0);
cube.setNormal(5, 0, -1, 0);

cube.setTexture(0, loadImage("1.jpg"));
cube.setTexture(1, loadImage("2.jpg"));
cube.setTexture(2, loadImage("3.jpg"));
cube.setTexture(3, loadImage("4.jpg"));
cube.setTexture(4, loadImage("5.jpg"));
cube.setTexture(5, loadImage("6.jpg"));
```

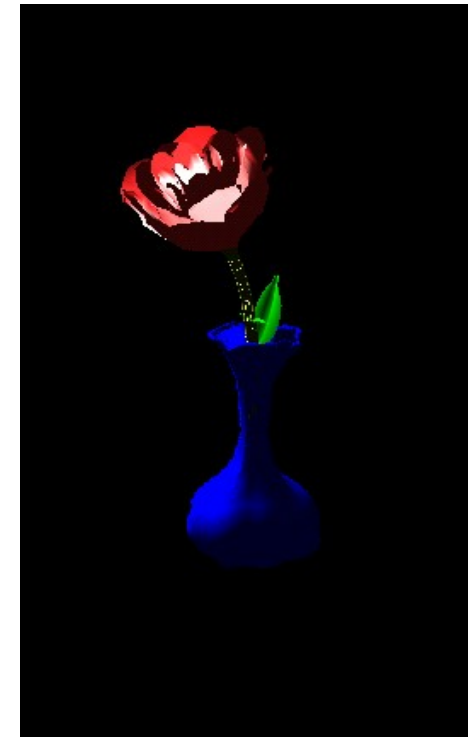




## OBJ loading

1. The OBJ format is a text-based data format to store 3D geometries. There is an associated MTL format for materials definitions.
2. It is supported by many tools for 3D modeling (Blender, Google Sketchup, Maya, 3D Studio). For more info: <http://en.wikipedia.org/wiki/Obj>
3. A3D supports loading OBJ files into PShape3D objects with the loadShape() function.
4. Depending the extension of the file passed to loadShape (.svg or .obj) Processing will attempt to interpret the file as either SVG or OBJ.
5. The 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);  
}
```

## Now, some digression...

### Direct mode versus vertex arrays in OpenGL

```
glBegin();  
glVertex3f(1.0, 2.0, 0.0);           gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 300);  
...  
glEnd();
```

People like direct mode because is easy to use and intuitive. But it is also very inefficient. We only start to take full advantage of the GPU hardware when we move to vertex arrays, VBOs, etc.

It would be nice if we somehow can combine both...

## So... A3D combines direct mode and VBOs with **shape recording**

1. Shape recording into a PShape3D object is a feature that can greatly increase the performance of sketches that use complex geometries.
2. The basic idea of shape recording is to save the result of standard Processing drawing calls into a Pshape3D object.
3. Recording is enabled by using the beginRecord()/endRecord() methods. Anything that is drawn between these two calls will be stored in the Pshape3D returned by beginRecord():

```
Pshape recShape;

void setup() {
  size(480, 800, A3D);
  ...
  recShape = beginRecord();
  beginShape(QUADS);
  vertex(50, 50);
  vertex(width/2, 50);
  vertex(width/2, height/2);
  vertex(50, height/2);
  endShape();
  endRecord();
}

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

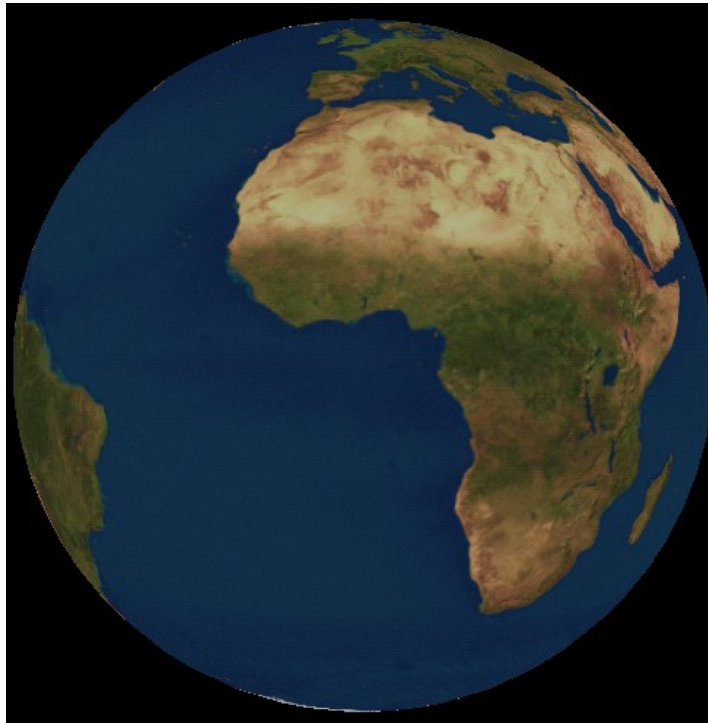
```
Pshape objects;

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

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

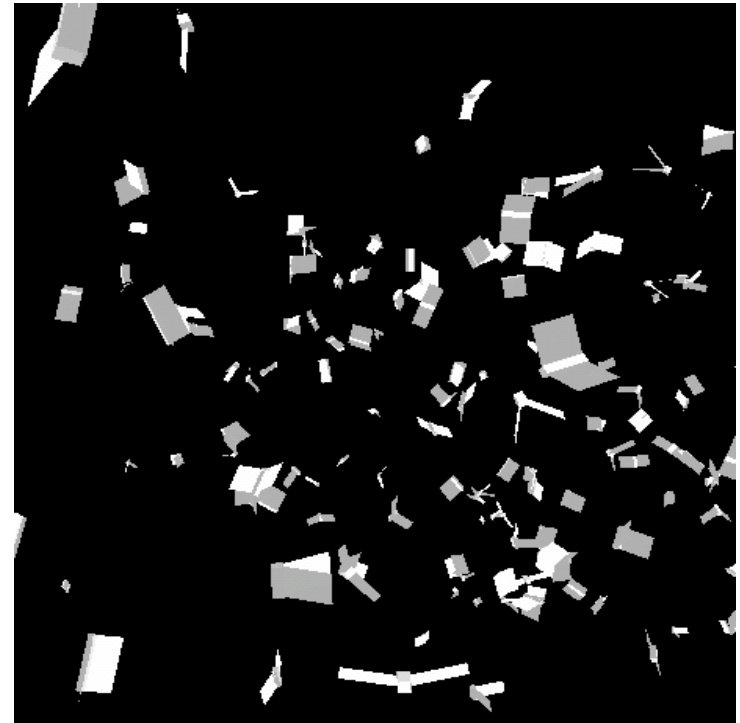
The performance gains of using shape recording are quite substantial. It usually increases the rendering framerate by 100% or more.

Textured sphere



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

Birds flock



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

Earth example (w/out model recording)

# 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.loadVertices();
for (int i =0; i < particles.getNumVertices(); i++) {
    float x = random(-30, 30);
    float y = random(-30, 30);
    float z = random(-30, 30);
    particles.set(i, x, y, z);
}
particles.updateVertices();
sprite = loadImage("particle.png");
particles.setTexture(sprite);
```

Creation/initialization

```
particles.beginUpdate(VERTICES);
for (int i =0; i < particles.getVerticesCount(); i++) {
    particles.vertices[3 * i + 0] += random(-1, 1);
    particles.vertices[3 * i + 1] += random(-1, 1);
    particles.vertices[3 * i + 2] += random(-1, 1);
}
particles.updateVertices();
```

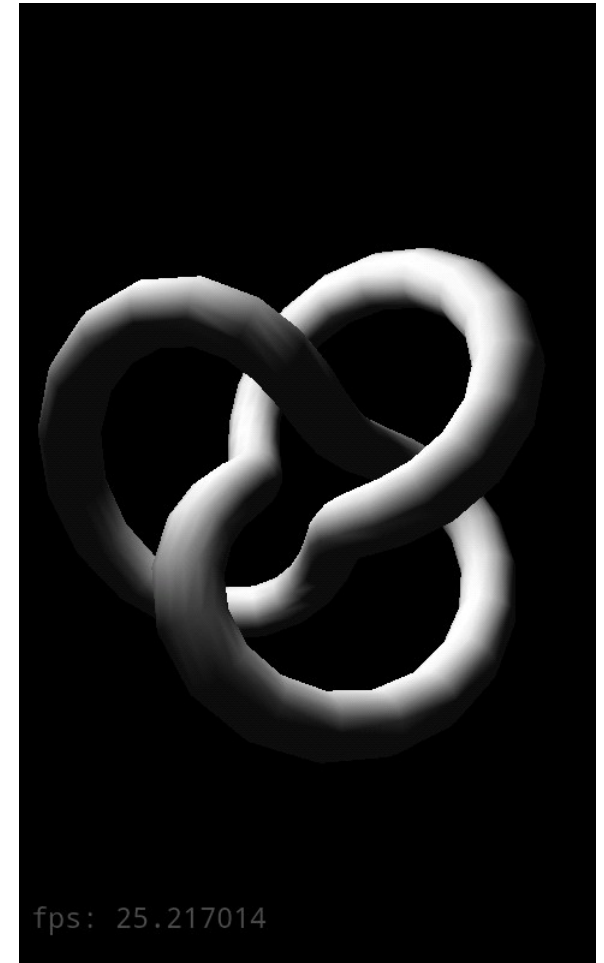
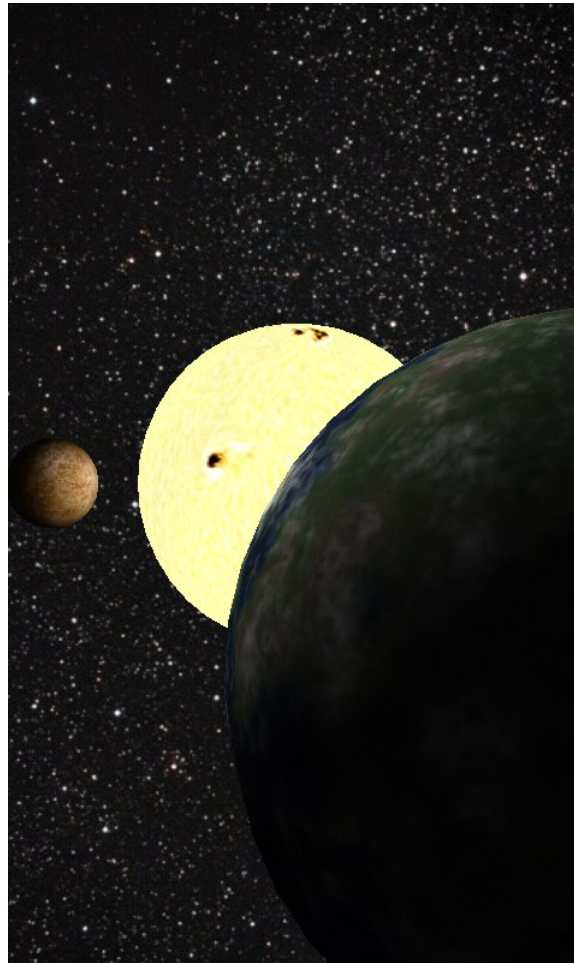
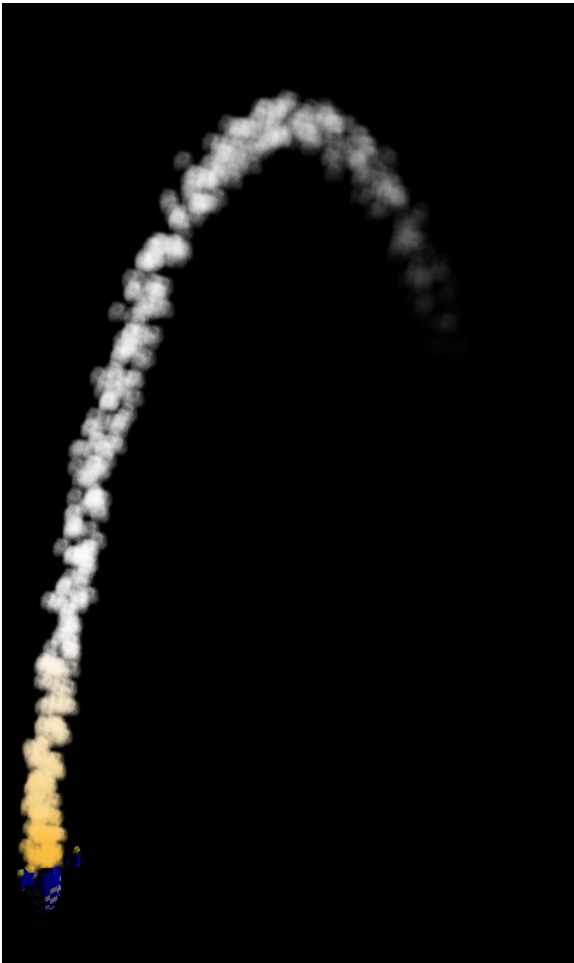
Dynamic update

## Multitexturing (part 2)

If a geometry is recorded with multiple textures, the resulting PShape3D object will store all these textures and associated texture coordinates. We can dynamically edit their values by using the texcoords array:

```
p.loadTexcoords(1);  
for (int i = 0; i < p.getVertexCount(); i++) {  
    float u = p.texcoords[2 * i + 0];  
    u += 0.002;  
    p.texcoords[2 * i + 0] = u;  
}  
p.updateTexcoords();
```





To wrap up, let's look at some advanced examples where we do model recording, particles systems, multitexturing, offscreen rendering, etc.

## Links

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

Official google resources: <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/>

GL ES 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/>

Processing resources: <http://processing.org/>

Processing.Android forum: <http://forum.processing.org/android-processing>

Processing.Android forum: <http://wiki.processing.org/w/Android>

## Books

### **Hello, Android: Introducing Google's Mobile Development Platform**

Ed Burnette

Pragmatic Bookshelf; 3 edition (July 20, 2010)

### **Professional Android 2 Application Development**

Rato Meier

Wrox; 1 edition (March 1, 2010)

### **Pro Android 2**

Sayed Hashimi

### **Getting Started with Processing**

Casey Reas and Ben Fry.

Published June 2010, O'Reilly Media.

### **Processing: A Programming Handbook for Visual Designers and Artists**

Casey Reas and Ben Fry

Published August 2007, MIT Pres

Introduction to  
**Processing**  
on **Android devices**  
**SIGGRAPH ASIA 2010**

Andres Colubri ([andres.colubri@gmail.com](mailto:andres.colubri@gmail.com))  
Jihyun Kim ([kimjihyunn@gmail.com](mailto:kimjihyunn@gmail.com))