

for Android */



222);

rial-Black-13.v1w");

25, 40);

", 25, 60);

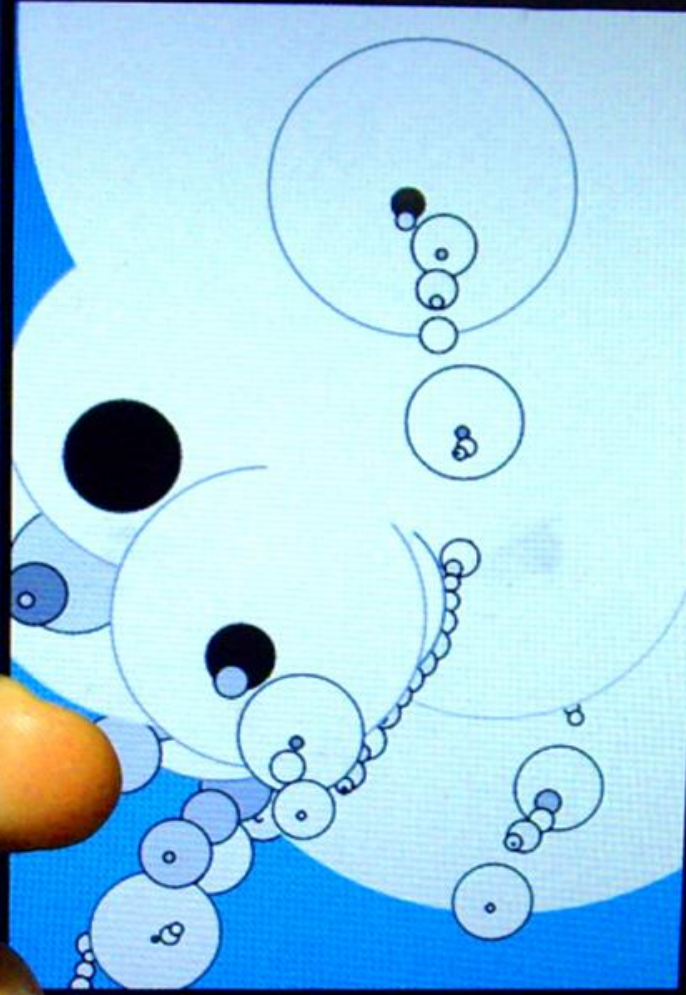
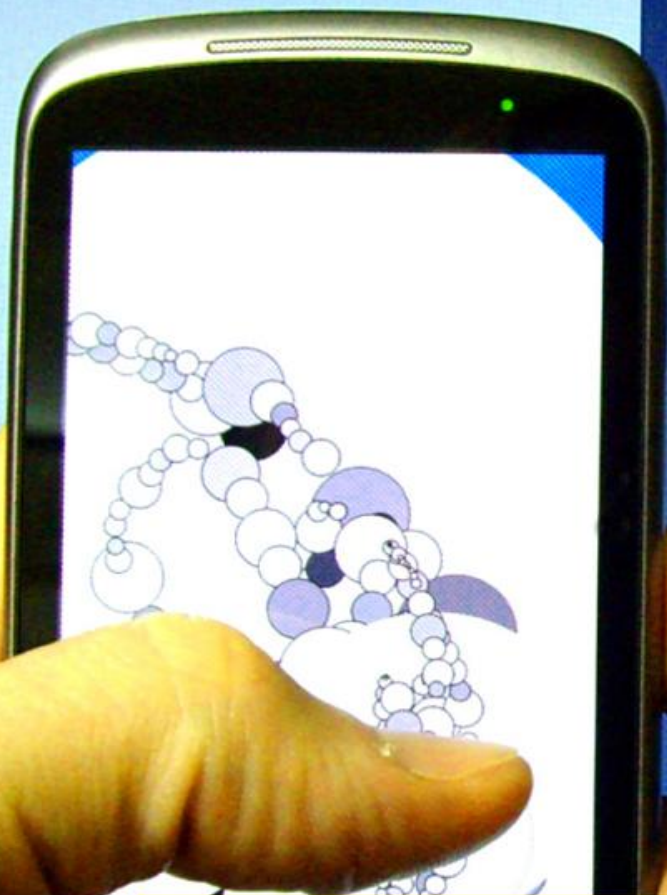
useX, mouseY, pmouse

(int x, int y,

(x-px) + abs

ed, r

;



1	2@	3#	4\$
Q	W	E	R
A	S	D	F
↑	Z	X	C
ALT	SYM	@	

Introduction to Processing on Android devices

Introduction to

Processing



on Android devices



Saturday, December 18th (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 the **Android platform**, running simple graphic applications and **uploading them to the phones**, 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

(14:15 - 16:00, Presenter: Jihyun Kim)

1. What is Android? (10min)
2. What is Processing? (10min)
3. Basic concepts in Android applications (15min)
4. First steps with Processing for Android (20min)
5. Basic Processing use (30min)
6. Extending Processing (20min)

PART II:

Fast 3D Graphics in Processing for Android

(16:15 - 18:00, Presenter: Andres Colubri)

7. OpenGL and Processing (15min)
8. Geometrical transformations (10min)
9. Camera and perspective (10min)
10. Creating 3D objects (30min)
11. 3D text (10min)
12. Models: the PShape3D class (30min)

PART I:

Introduction to Processing for Android

(14:15-16:00) Presenter: Jihyun Kim

1.What is Android? (14:15 – 14:25)

- 1.1 Android devices
- 1.2. Development process and Android Market
- 1.3. Hardware (input devices, touchscreen, accelerometer, GPS, compass)

2. What is Processing? (14:25 – 14:35)

- 2.1 Main goals of Processing
- 2.2 Use in education and artistic/design production
- 2.3 Overview of Processing libraries

3. Basic concepts in Android applications (14:35 – 14:50)

- 3.1 Views, Activities, Intents, and the manifest file
- 3.2 Android Software Development Kit (SDK)
- 3.3 Use of Eclipse for Android development

PART I:

Introduction to Processing for Android

(14:15-16:00) Presenter: Jihyun Kim

4. First steps with Processing for Android (14:50 – 15:10)

- 4.1 Installation of the Android SDK and Processing
- 4.2. Android mode in Processing
- 4.3 Running sketches on the emulator and the device

5. Basic Processing use (15:10 – 15:40)

- 5.1 Drawing of 2D shapes and use of color
- 5.2 Animation and motion
- 5.3 Media (Images and Fonts)
- 5.4 Interaction (keyboard, touchscreen, multitouch handling)

6. Extending Processing (15:40 – 16:00)

- 6.1 Libraries for Android (oscP5, controlP5, Ketai in Motion)
- 6.2 Accessing other Android hardware (camera, GPS)

PART II:

Fast 3D Graphics in Processing for Android

(16:15 – 18:00) Presenter: Andres Colubri

7. OpenGL and Processing (16:15 – 16:30)

- 7.1 OpenGL ES for mobile devices
- 7.2 Mobile GPUs (Adreno, PowerSVG, Tegra)
- 7.3 Hardware requirements for 3D in Processing for Android
- 7.4 The A3D renderer
- 7.5 Using offscreen drawing

8. Geometrical transformations (16:30 – 16:40)

- 8.1 Translations, rotations, scaling
- 8.2 The transformation matrix stack

9. Camera and perspective (16:40 – 16:50)

- 9.1 Camera placement
- 9.2 Perspective and orthographic views

PART II:

Fast 3D Graphics in Processing for Android

(16:15 – 18:00) Presenter: Andres Colubri

10. Creating 3D objects (16:50 – 17:20)

10.1 Using the Processing API to create basic 3D primitives: boxes, spheres, vertex shapes

10.2 Texturing and lighting

11. 3D text (17:20 – 17:30)

11.1 Using system fonts to create text on the fly

11.2 API to handle and transform text strings

12. Models: the PShape3D class (17:30 – 18:00)

12.1 Introduction to Vertex Buffer Objects (VBOs)

12.2 Techniques for creating PShape3D models: OBJ loading, shape recording, SVG copy

12.3 Particle systems

PART I:

Introduction to

Processing for

Android

1. What is Android?



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

"Android is the first truly open and comprehensive platform for mobile devices. It includes an operating system, user-interface and applications -- all of the software to run a mobile phone, but **without the proprietary obstacles** that have hindered mobile innovation."

(<http://googleblog.blogspot.com/2007/11/wheres-my-gphone.html>)

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. **NAVIGATION 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.

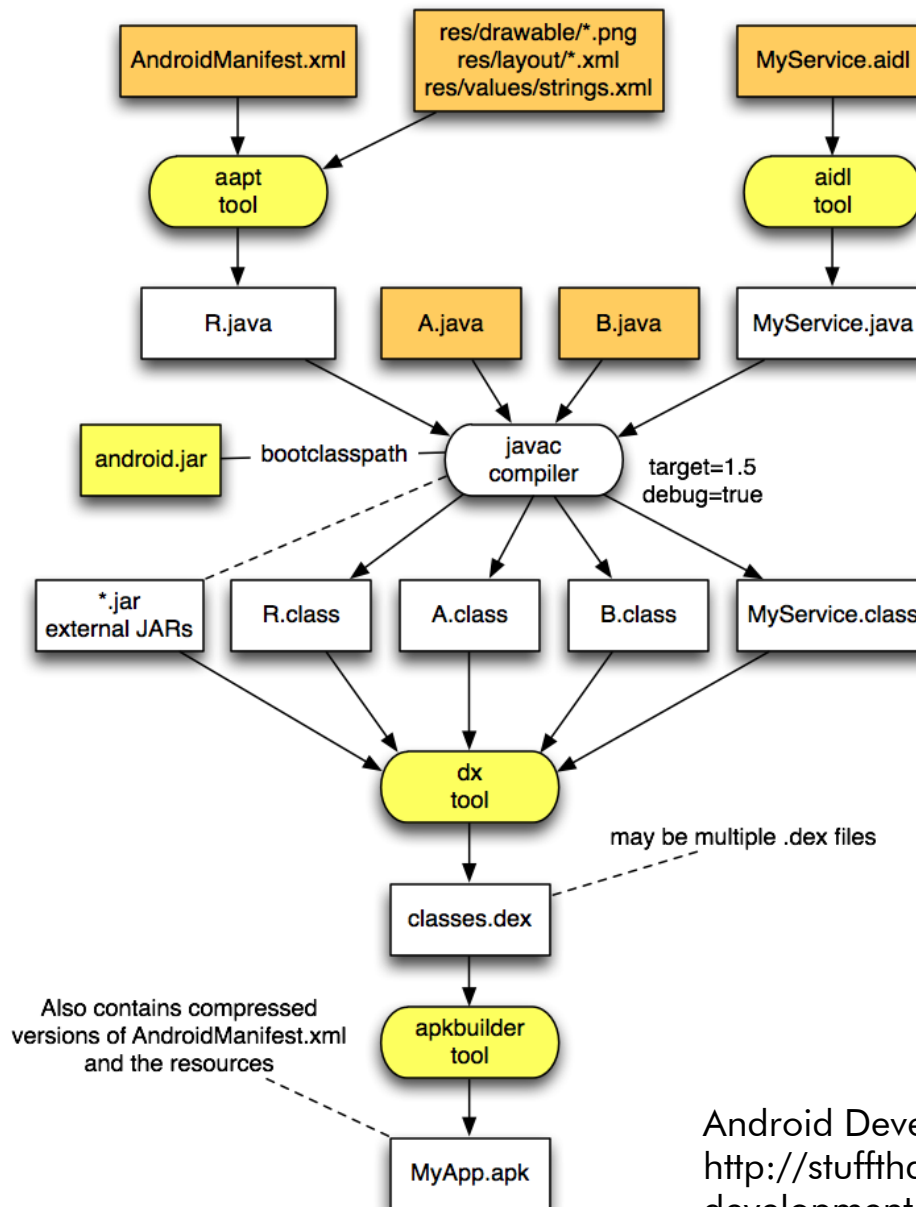
Android Compatibility Program: <http://source.android.com/compatibility/overview.html>

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 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)



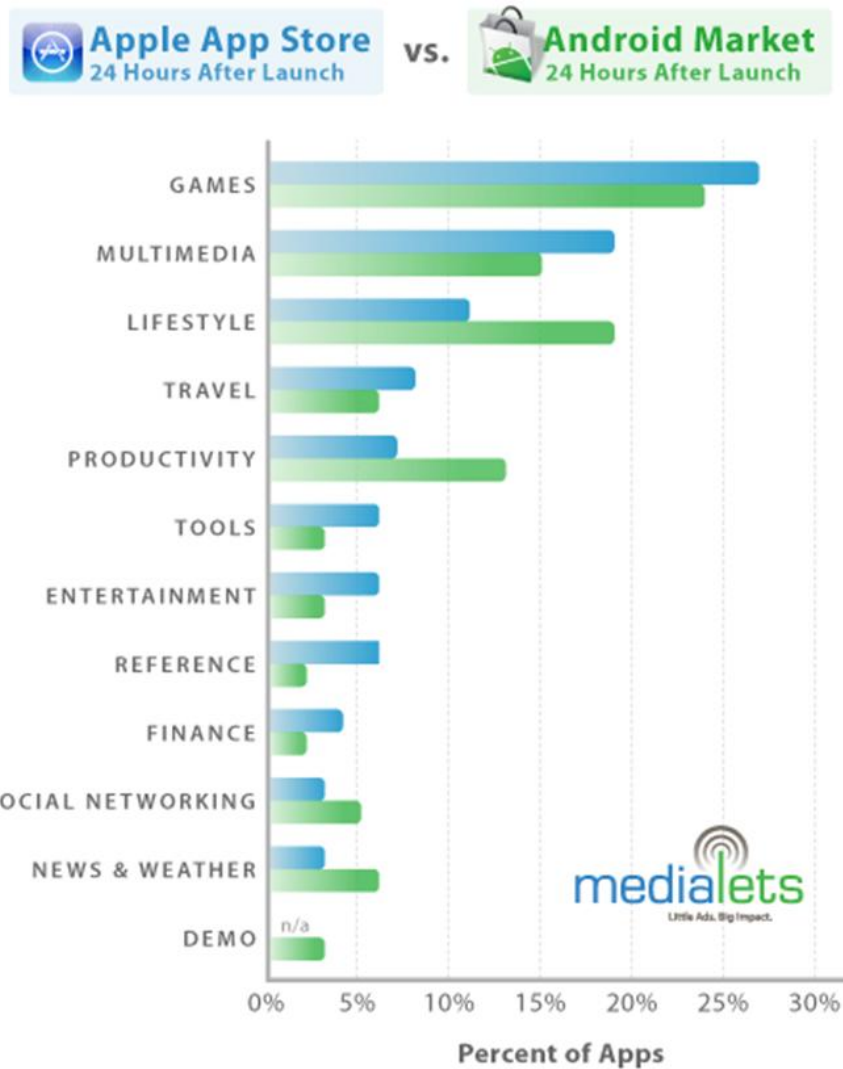
Kindle for Android

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.

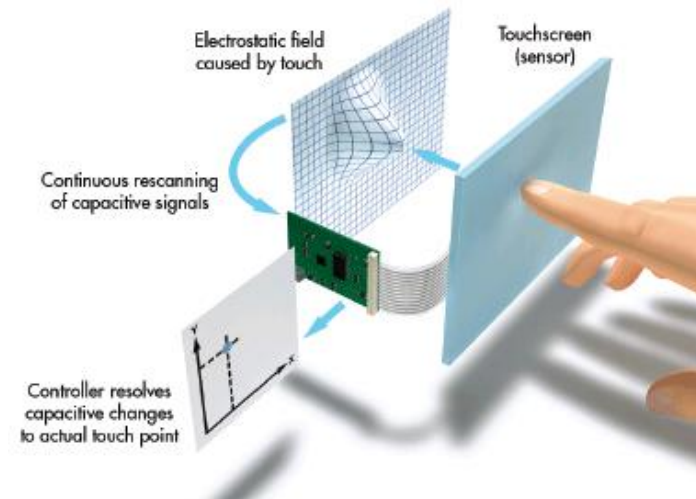
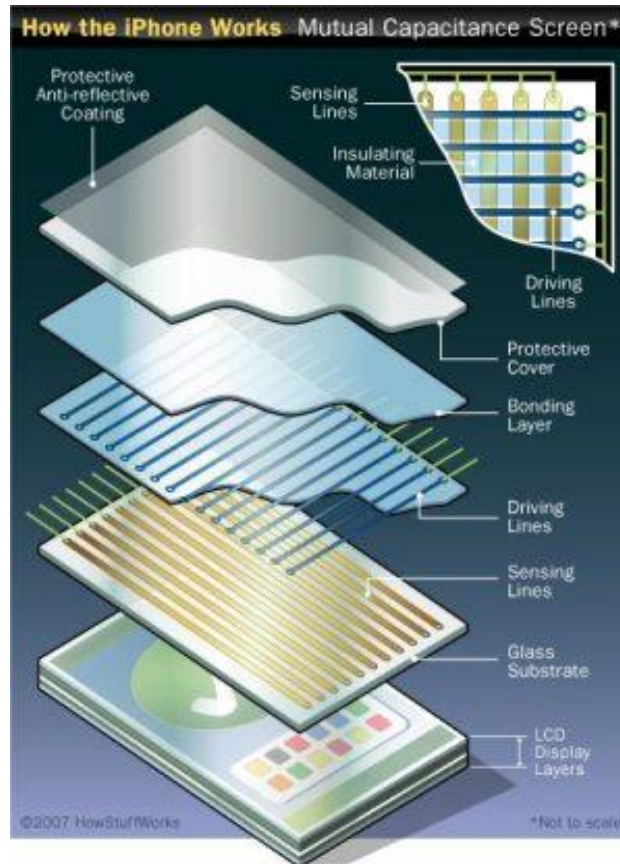
<http://www.android.com/market/>

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**...

1.3 Hardware



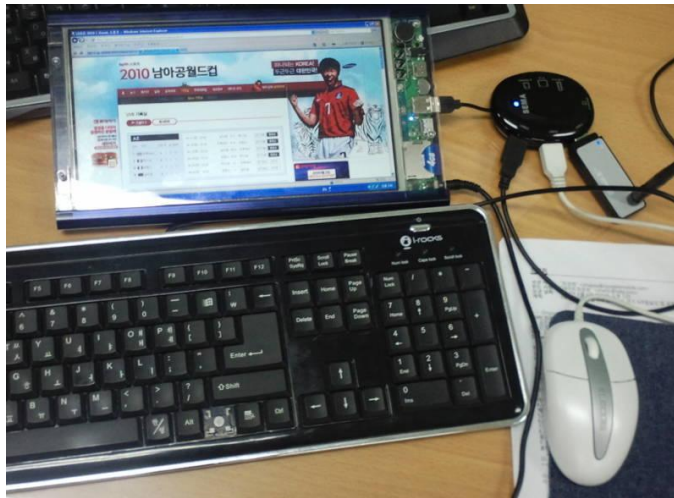
Accelerometer



GPS

1.3 Hardware

Now there is a lot of interest in the upcoming Android tablets (Samsung Tab, Odroid-T), which extend Android 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:

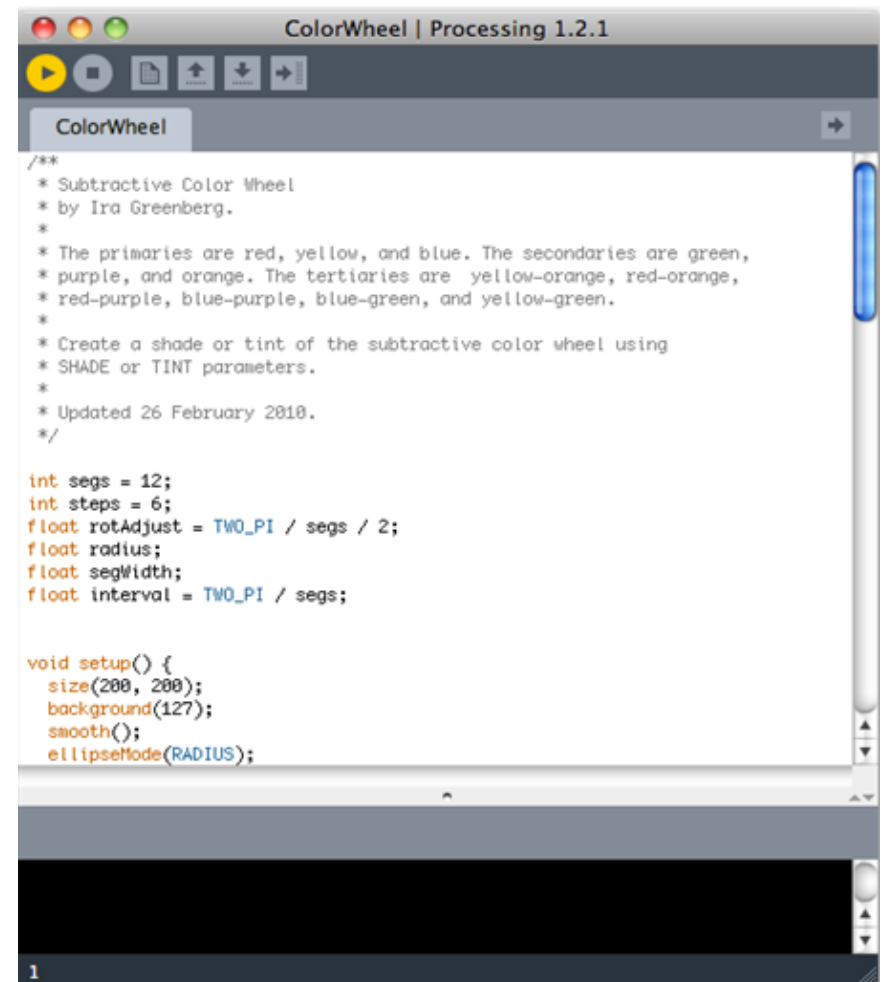
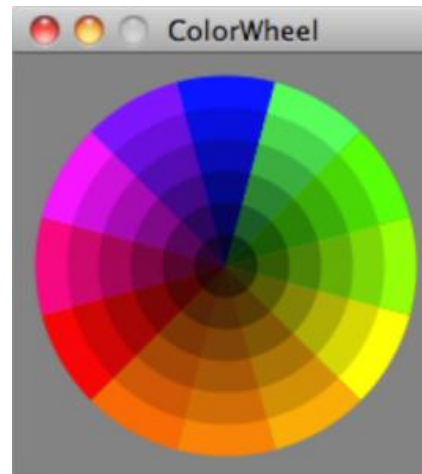
1. A minimal Development Environment (Called PDE), that favors ease of use over functionality
2. A programming language, and as such is basically a dialect built on top of Java to make graphics programming less cumbersome thanks to a simple API.

```
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

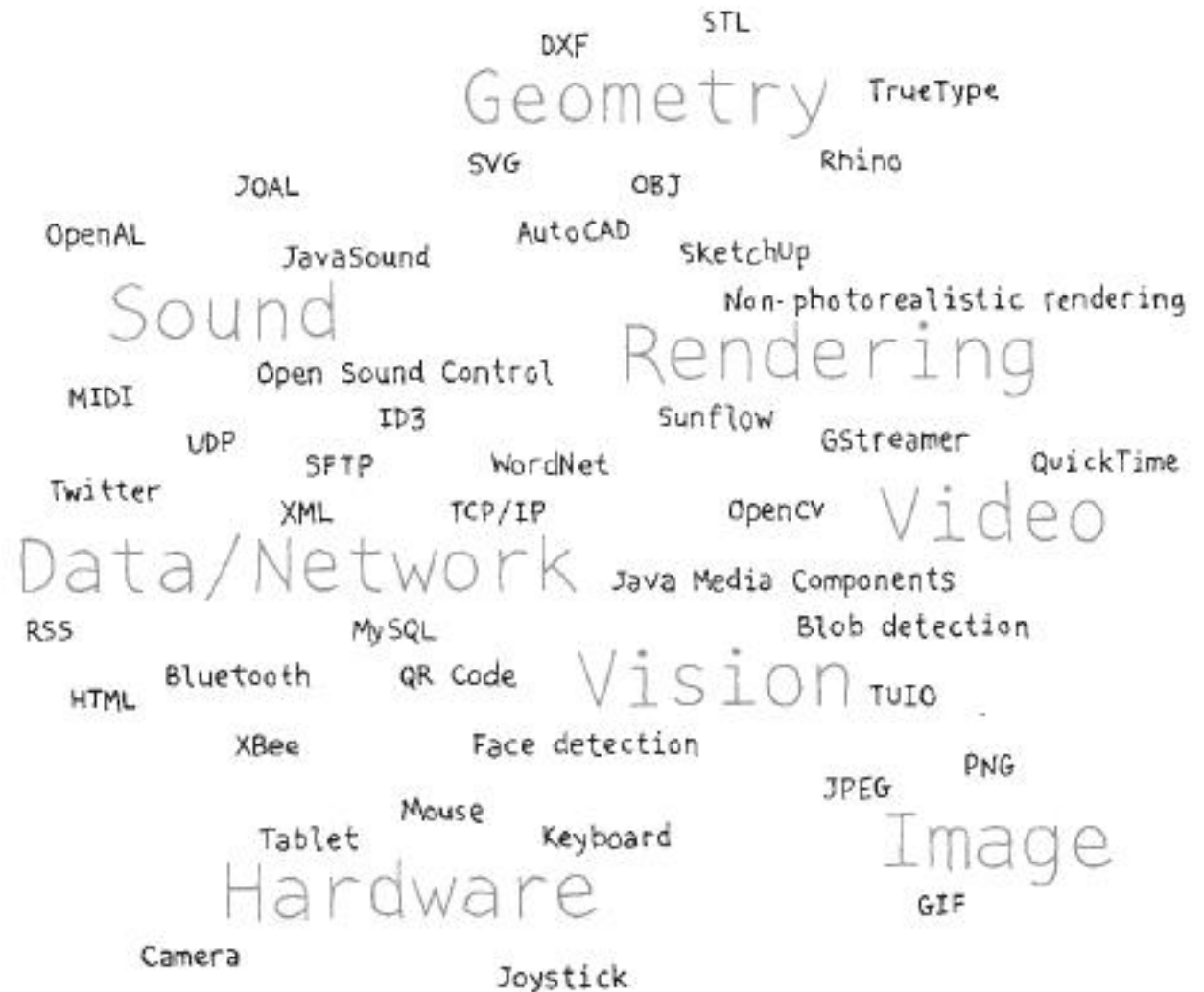
Simplicity



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

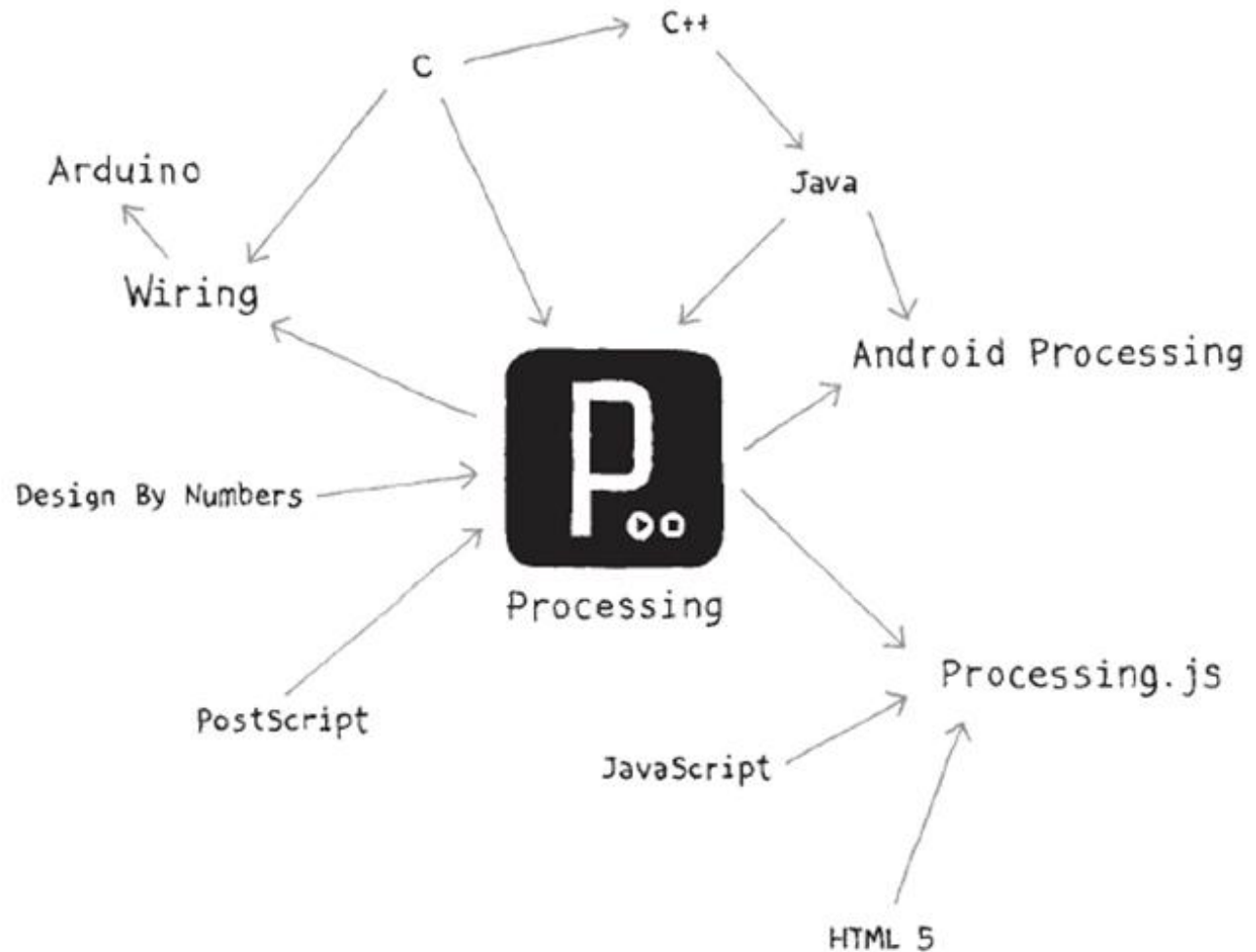


Many types of information can flow in and out of Processing.

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

2.1. Main goals of Processing

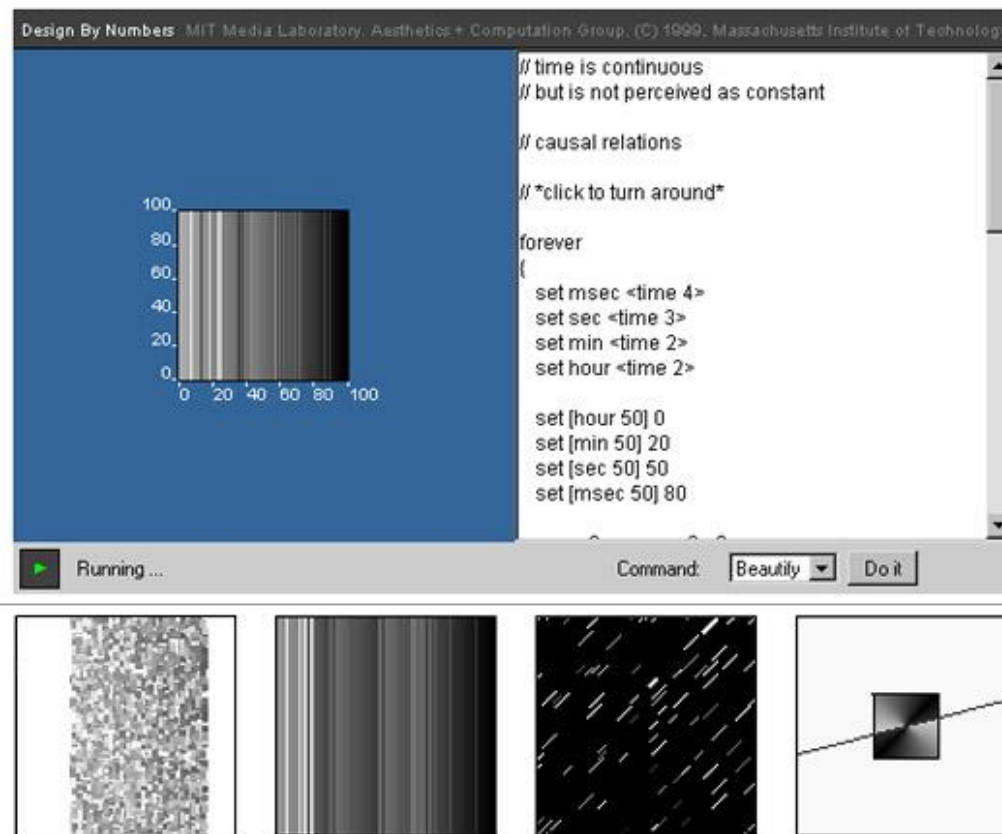
Family Tree



Processing has a large family of related languages and programming environments

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

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

OpenProcessing^{beta}
not logged in yet
[login/register](#)

[home](#) [browse](#) [classrooms & collections](#) [books](#) [blog](#) [search](#)

Classrooms^{beta}
[klas-room, -room, klahs-]: any place where one learns or gains experience: *"The sea is the sailor's classroom"*
Creating a classroom is currently by invitation only; email info@openprocessing.org to request an invitation. [Learn more](#)

Collections^{beta}
[kuh-lek-shuhn]: a group of objects or an amount of material accumulated in one location. *"an excellent Picasso collection"*
Creating a collection is currently by invitation only; email info@openprocessing.org to request an invitation. [Learn more](#)

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

Tree Generation
by Sinan Ascioglu
43 sketches by 36 users

Simply Object-Oriented
by Sinan Ascioglu
18 sketches by 16 users

The Nature of Code
by Daniel Shiffman
12 sketches by 3 users

Generating Visuals, MAD '08 at UPF
by Ricard Marxer Piñón
73 sketches by 16 users

Games
by Sinan Ascioglu
59 sketches by 45 users

default title
by Thiago Bueno Silva
0 sketches by 0 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

Physics
by Sinan
15 sketches by 9 users

Softwarelike Programs
by Riley Galloway
29 sketches by 13 users

Predictions
by Ultimate One
1 sketch by 1 user

M&A
by VIT
1 sketch by 1 user

Mal
1 sketch by 1 user

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](#)



[The 2009 Feltron 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: [geschloir.de](#), [theplaceofindme.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.

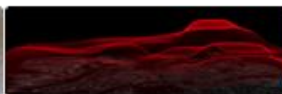
Links: [blipmt.com](#)



[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.

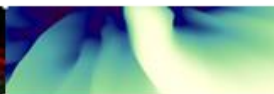
Links: [brvsonian.com](#)



[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



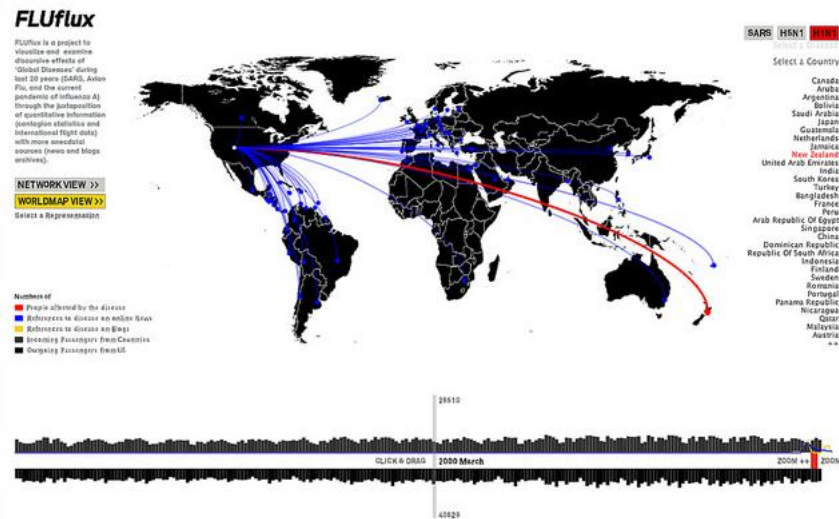
<Oasis>, 2008: Yunsil Heo, Hyunwoo Bang
http://everyware.kr/portfolio/contents/09_oasis/



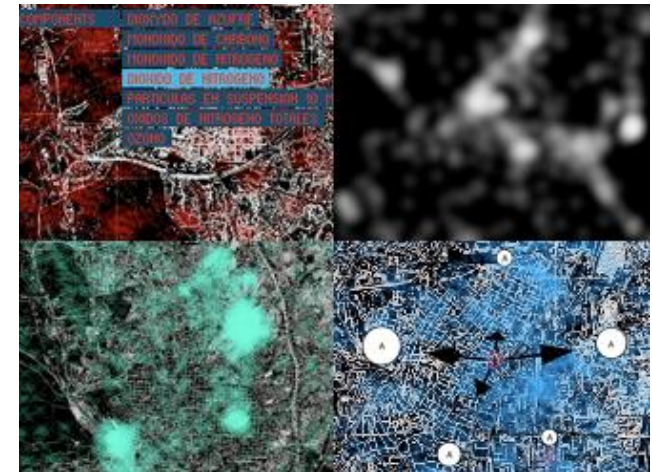
<Shadow Monsters>, 2005, Philip Worthington
<http://worthersoriginal.com/>

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, Victor 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/>



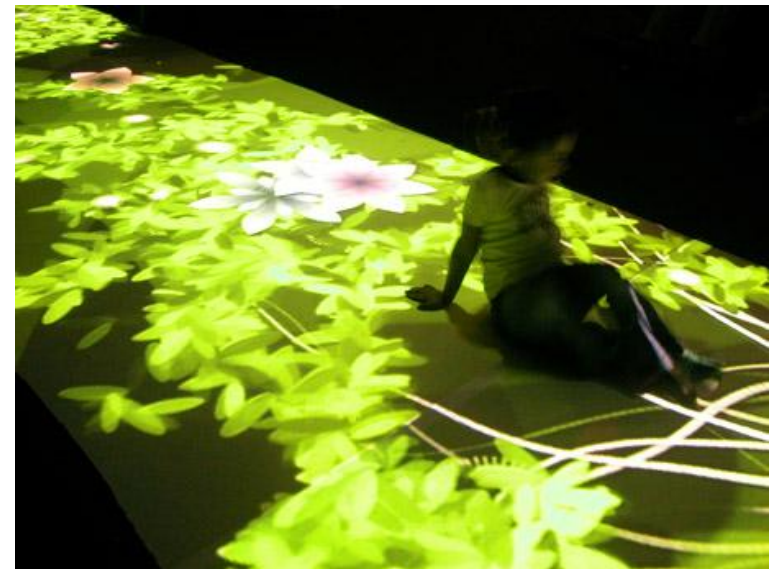
<Light Roller>, 2006, Random International
<http://www.random-international.com/>

2.2 Use in education and Artistic/Design Production

Real-time graphics and video



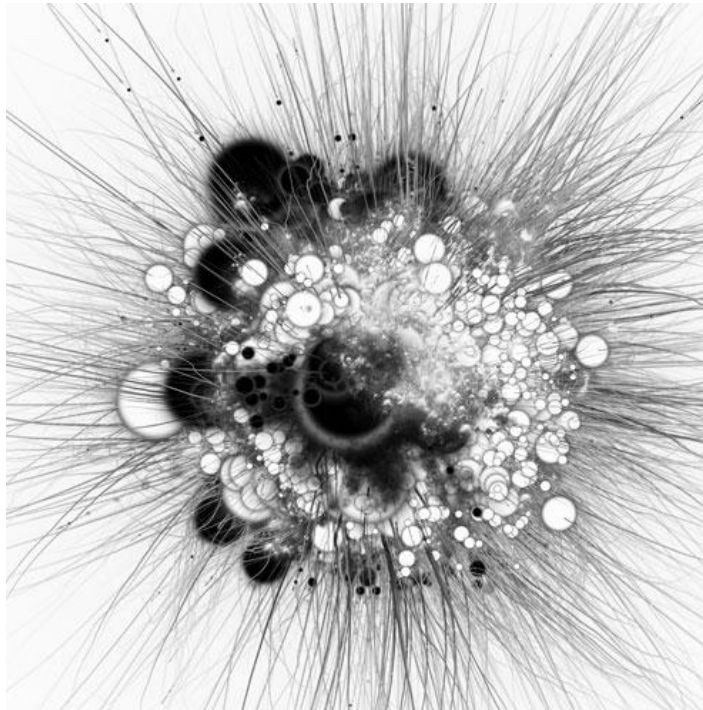
< Latent State >, 2009, Andres Colubri_
live cinema performance



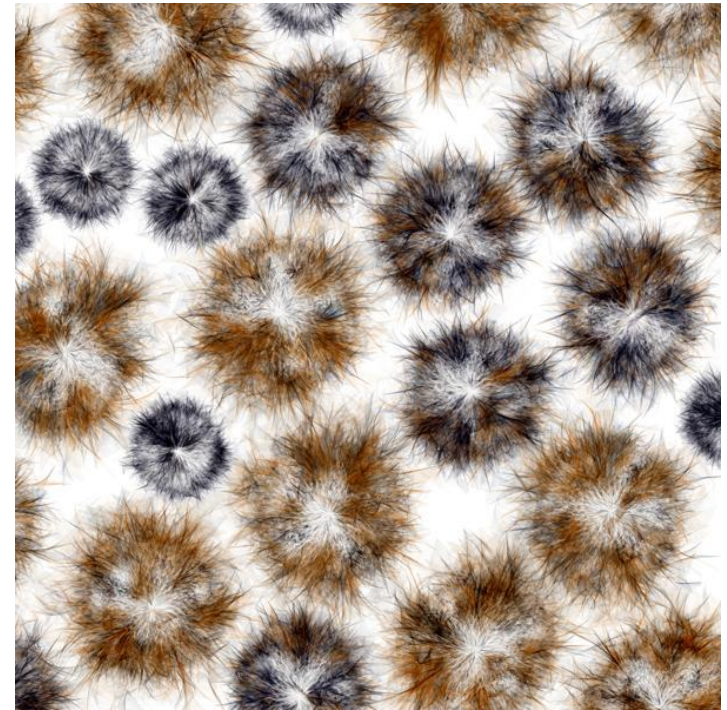
<Vattenfall Media Façade> Art+Com
<http://www.artcom.de/>

2.2 Use in education and Artistic/Design Production

2.2.5 Generative art

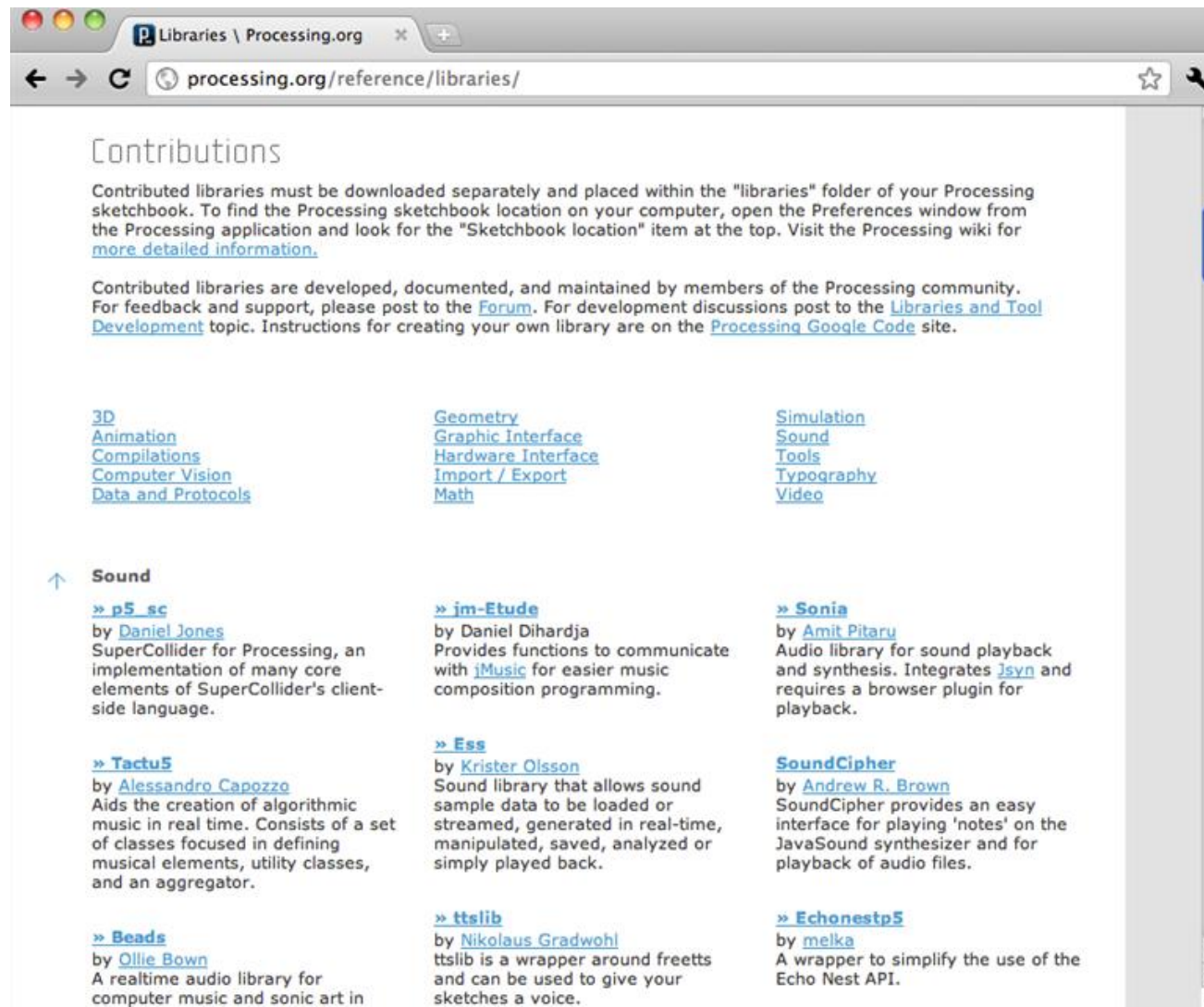


<Magnetic Ink, pt. 3> Robert Hodgkin
<http://www.flight404.com/blog/?p=103>



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

2.3 Overview of Processing Libraries



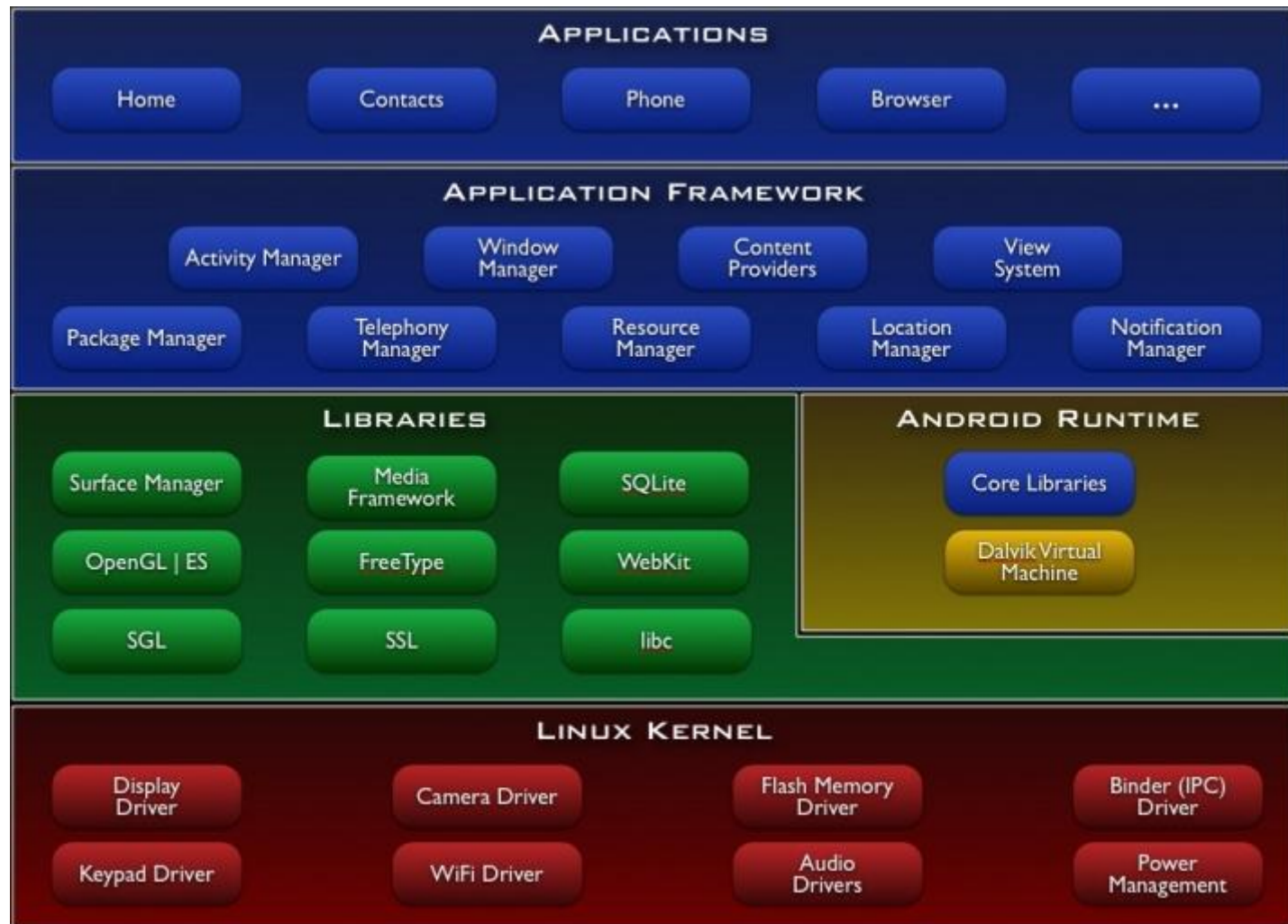
Processing allows more than 100 external libraries contributed by the community.

3. Basic Concepts in Android application

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 SQLite 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/>



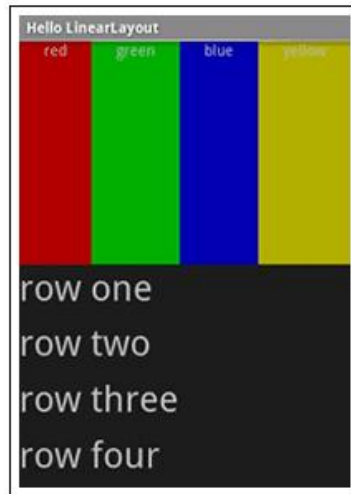
the major components of the **Android** operating system

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

3.1 Views, Activities, Intents, and the manifest file

Views

Linear Layout



Relative Layout

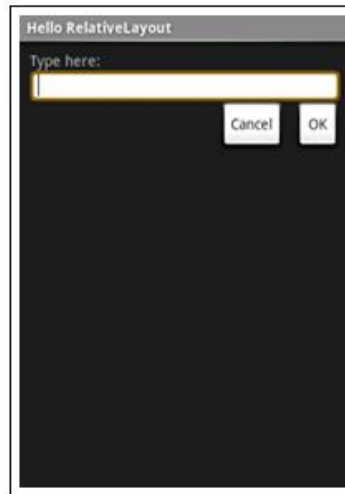
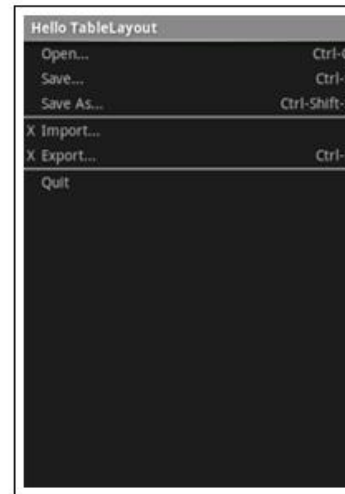
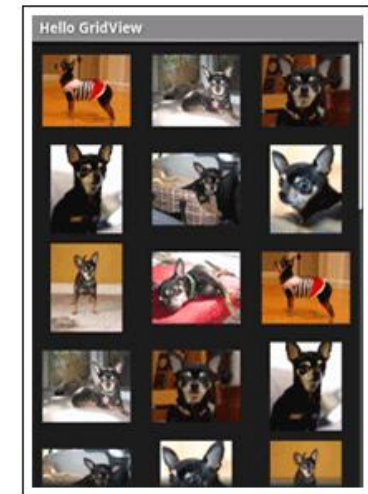


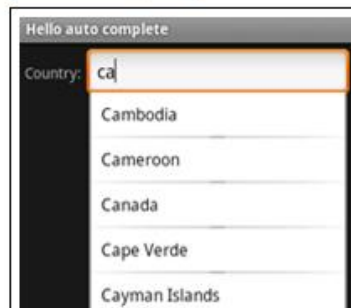
Table Layout



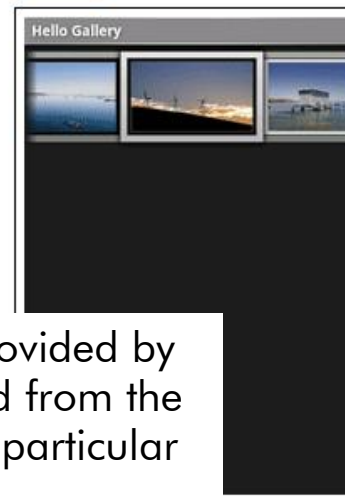
Grid View



Auto Complete



Gallery



Google Map View



Web View

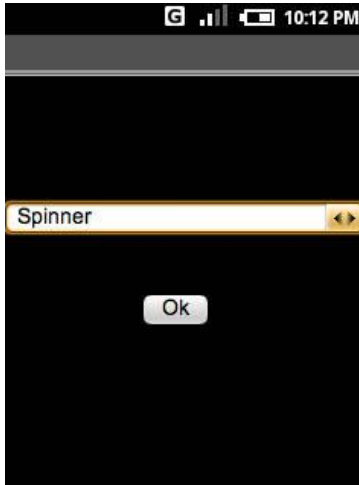


The visual content of the window is provided by a hierarchy of views — objects derived from the base **View** class. Each view controls a particular rectangular space within the window.

<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

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..

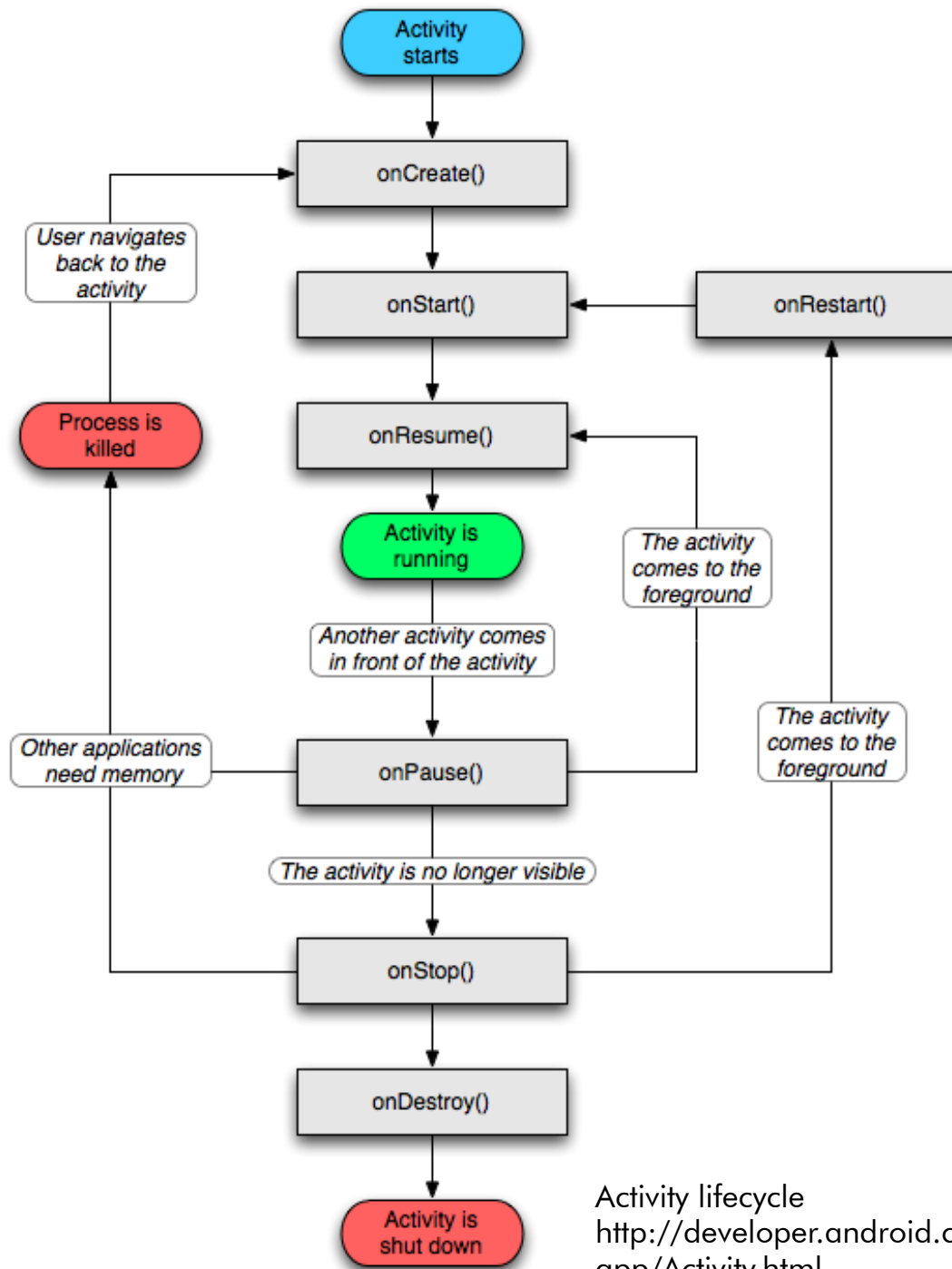
3.1 Views, Activities, Intents, and the manifest file

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.

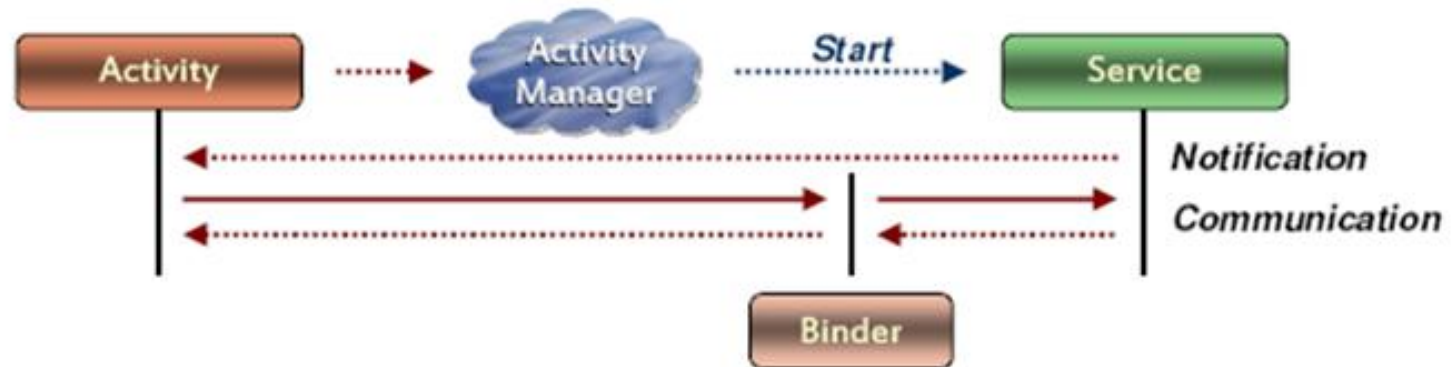
<http://developer.android.com/guide/topics/fundamentals.html>



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



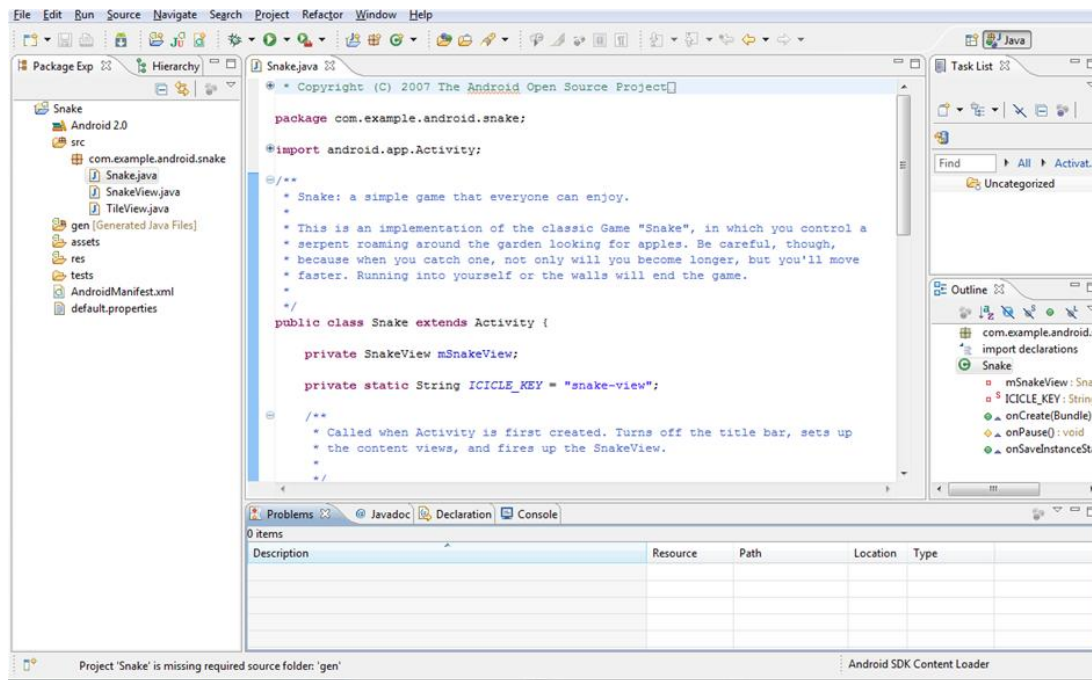
Eclipse can be used as the IDE for Android development. To do so, you need to install the SDK and then the: ADT (Android Development Tools)

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**.

3.3 Use of Eclipse for Android development

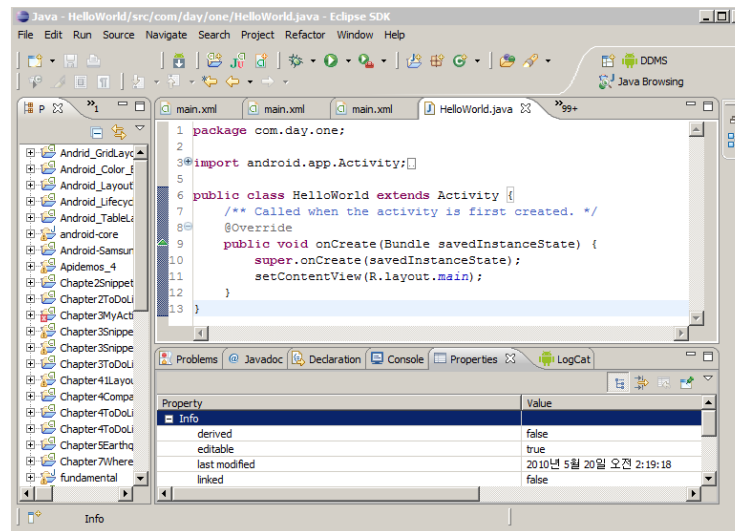
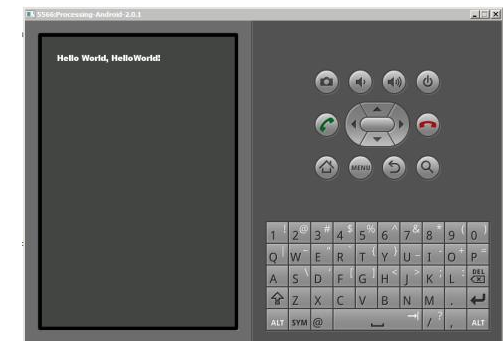
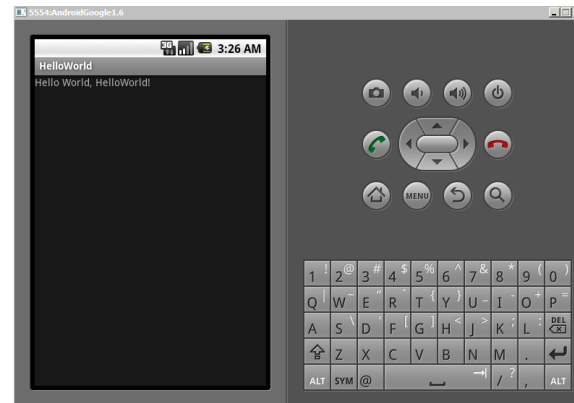


Solution: **Processing for Android!**

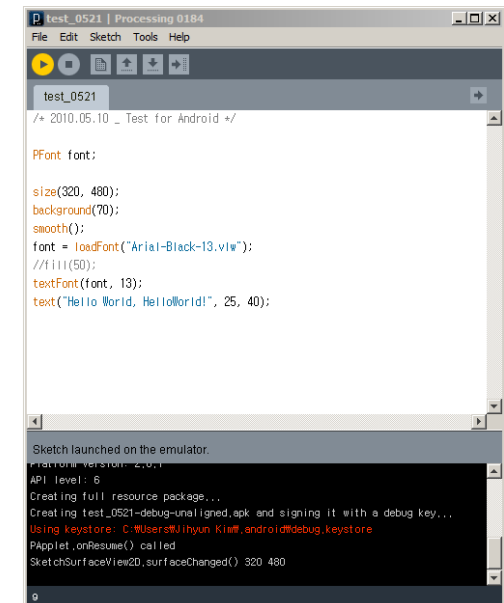
from the Android section in the Processing wiki:

"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

4.1 Installation of the Android SDK and Processing

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-g.html>

Here are the Java SE downloads in detail.

Java Platform, Standard Edition

JDK 6 Update 20 (JDK or JRE)

This release contains critical security updates to the Java runtime. Please update now to take advantage of these enhancements. » Learn more

[Download JDK](#)

[Docs](#)

What Java Do I Need? 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.

[Download JRE](#)

[Docs](#)

NOTE: The Firefox 3.6 browser requires Java SE 6 Update 10 or later. Otherwise, Java-based web applications will not work.

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

Platform	Package	Size	MD5 Checksum
Windows	android-sdk_r05-windows.zip	23449838 bytes	cc2c51a24e2f676e0fa652e182ef5840
Mac OS X (intel)	android-sdk_r05-mac_os_x.zip	19871714 bytes	6fced0e1c36624c926551637eb3308
Linux (i386)	android-sdk_r05-linux_i386.tar.gz	16208523 bytes	1d695d6a31310406f549092a1bd9850

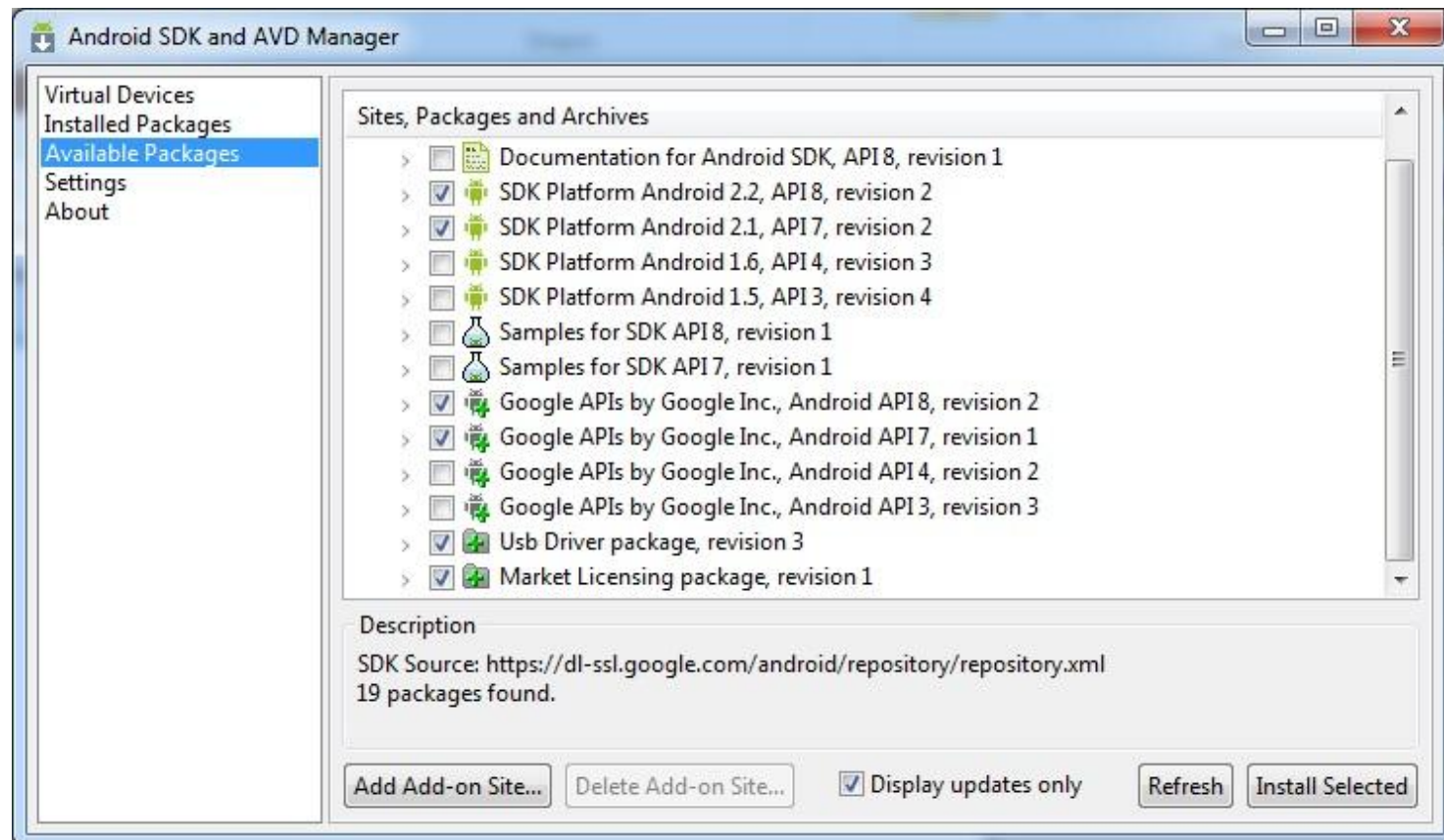
<http://developer.android.com/>

4.1 Installation of the Android SDK and Processing

Windows

1. Make sure that you install the **JDK** 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 using a tool included the SDK starter package — the Android SDK and AVD Manager . You can start it by double-clicking on android.exe in the tools folder

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

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

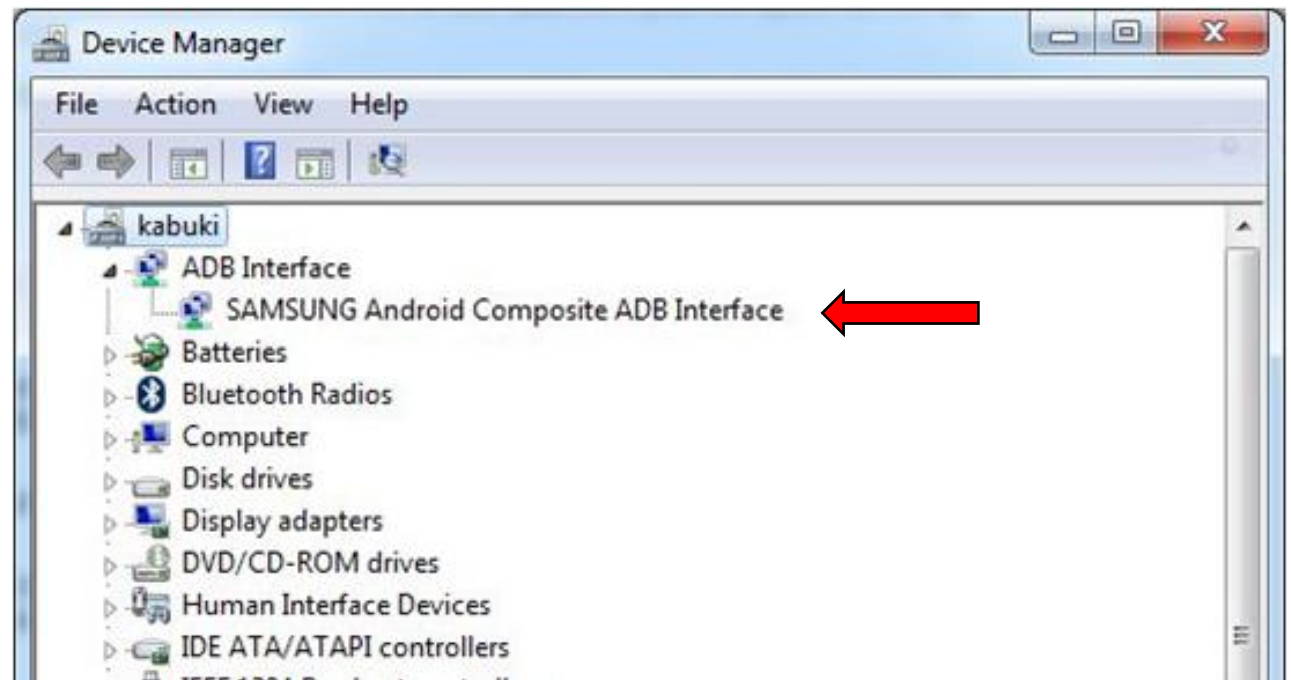
After downloading the usb driver in the previous step (it gets copied to C:\Users\andres\Coding\android-sdk-windows\usb_driver) you have to install it following the steps indicated here:

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

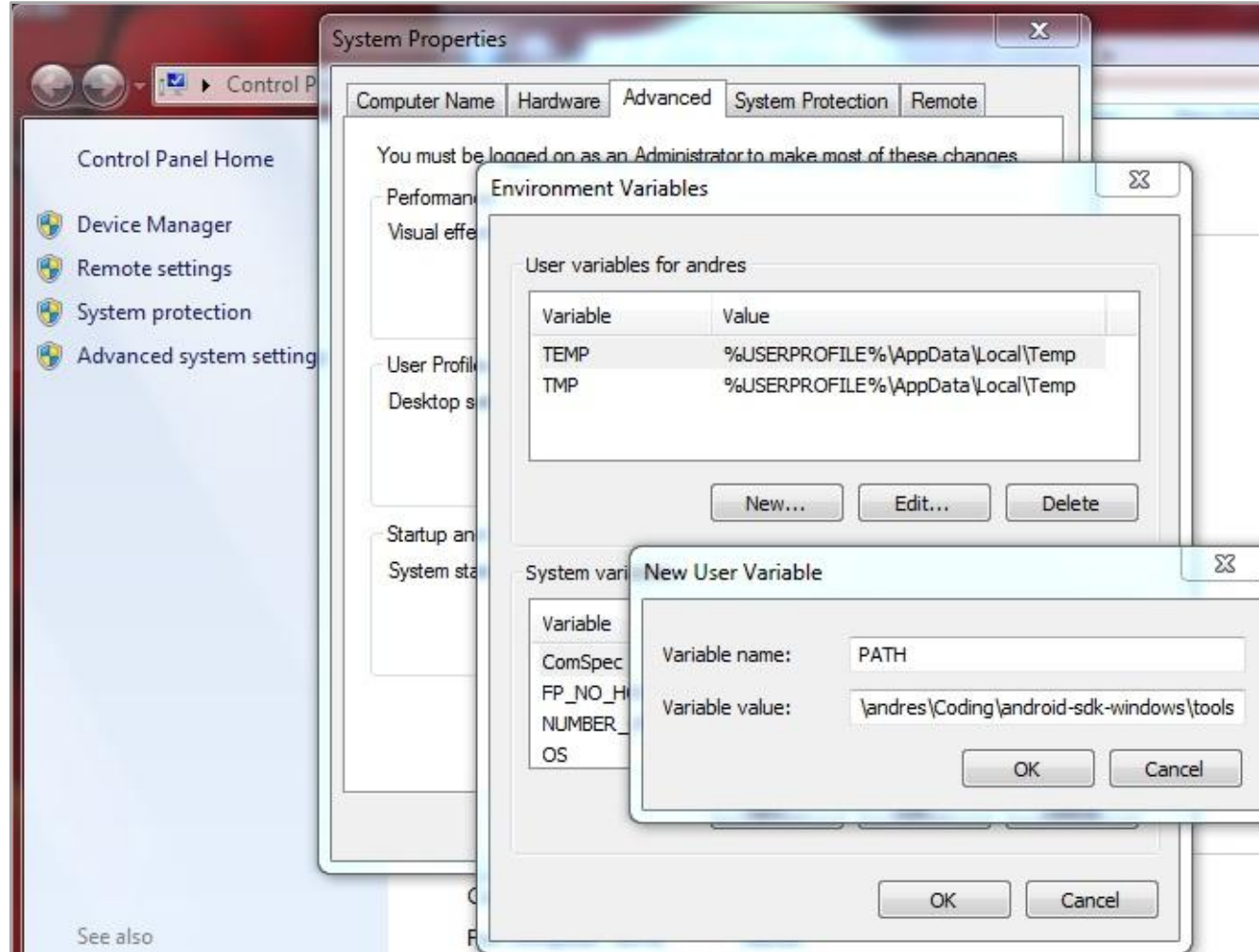
4.1 Installation of the Android SDK and Processing

Other phones might require **manufacturer's USB drivers**. 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

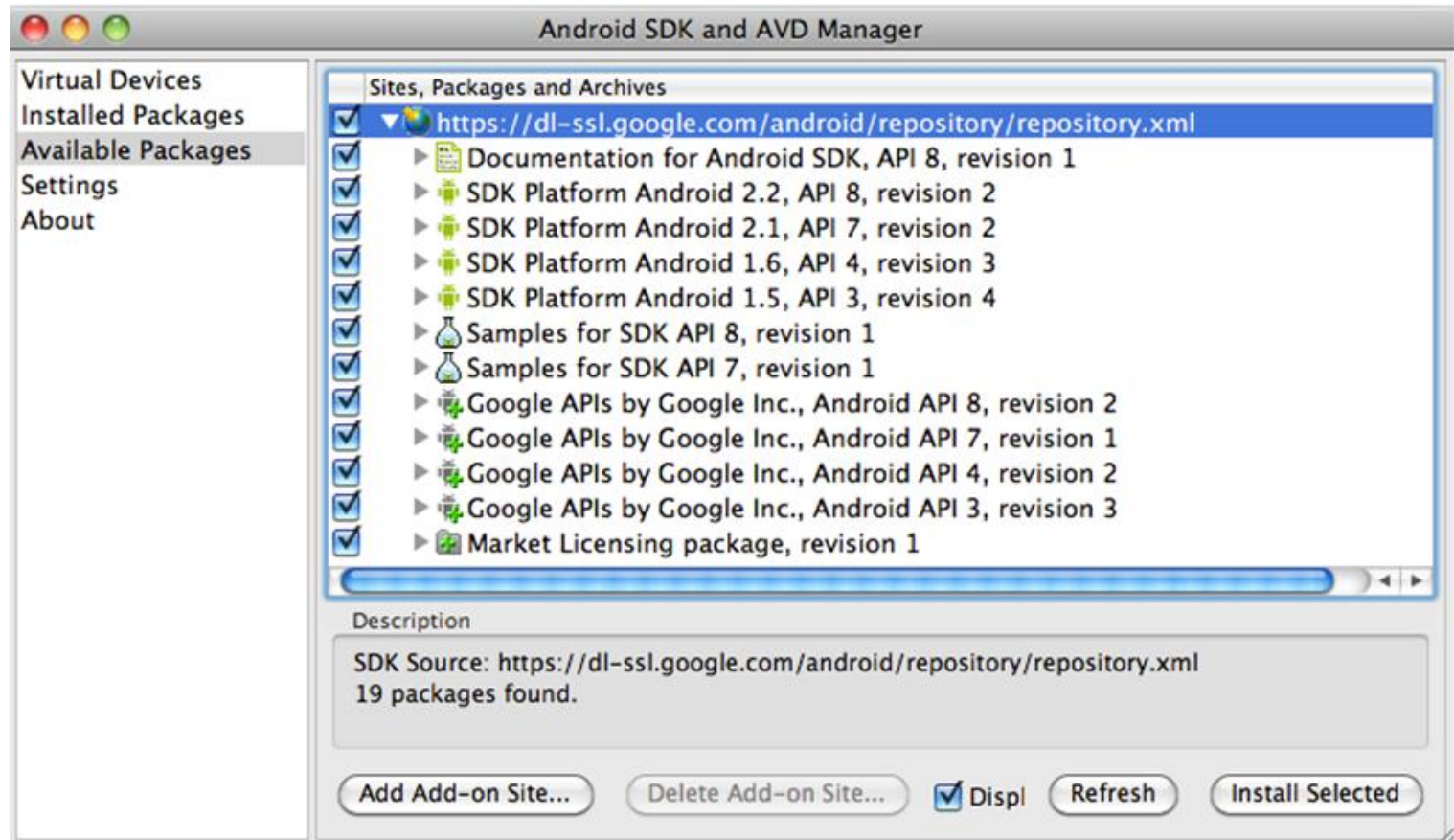
1. Download latest package from <http://developer.android.com/sdk/index.html>
2. *Unzip package* at the place of your preference...
`/Users/andres/Coding/Android/android-sdk-mac_x86`
3. On a Mac OS X, look in your home directory for `.bash_profile` and proceed as for Linux. You can create the `.bash_profile`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

```
export PATH=${PATH}:<your_sdk_dir>/tool  
export ANDROID_SDK=${PATH}:<your_sdk_dir> in our case this would be:
```

```
export PATH=${PATH}:/Users/andres/Coding/Android/android-sdk-mac_x86/tools  
export ANDROID_SDK=/Users/andres/Coding/Android/android-sdk-mac_x86
```

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

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

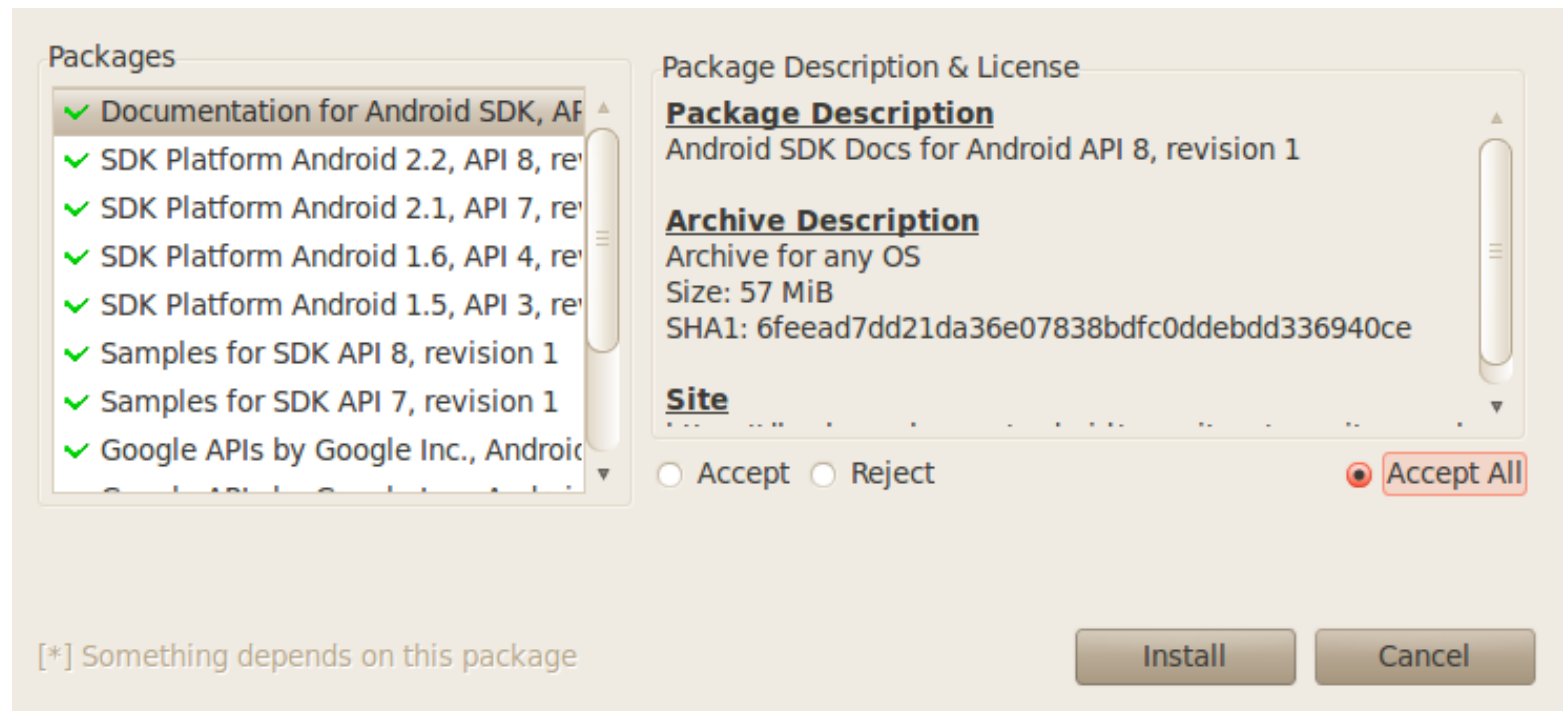
4.1 Installation of the Android SDK and Processing

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:

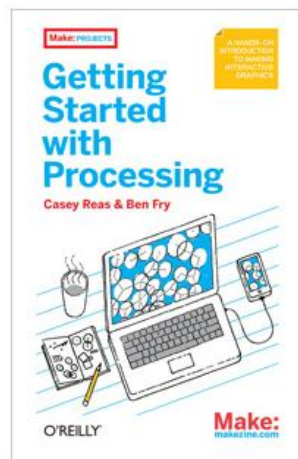
```
export PATH=${PATH}:<your_sdk_dir>/tools
export ANDROID_SDK=<your_sdk_dir>
```
6. The last step in setting up your SDK is using a tool included in the SDK starter package — the *Android SDK and AVD Manager*. You can start it running *android* from the terminal.

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...

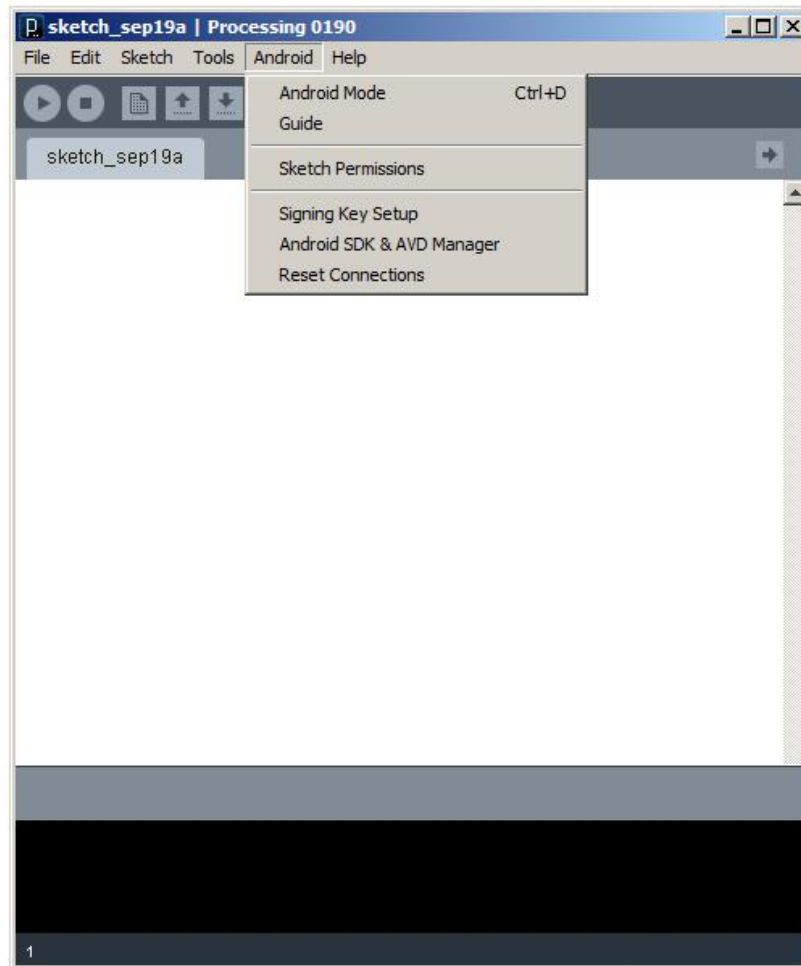
To see more of what people are doing with Processing, check out these sites:

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

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

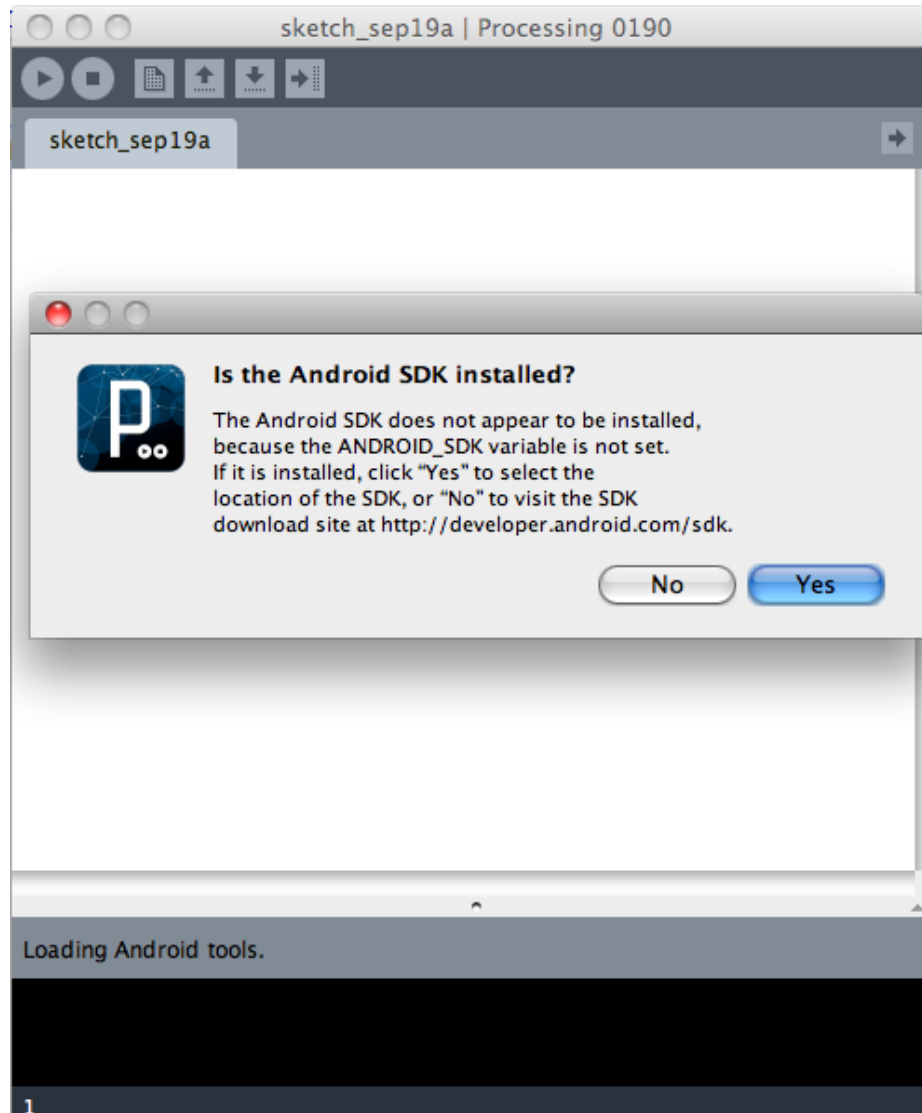
For windows and Linux, uncompress the **zip/tgz** in the desired location, in the case of **OSX** open the dmg package and copy to Applications

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

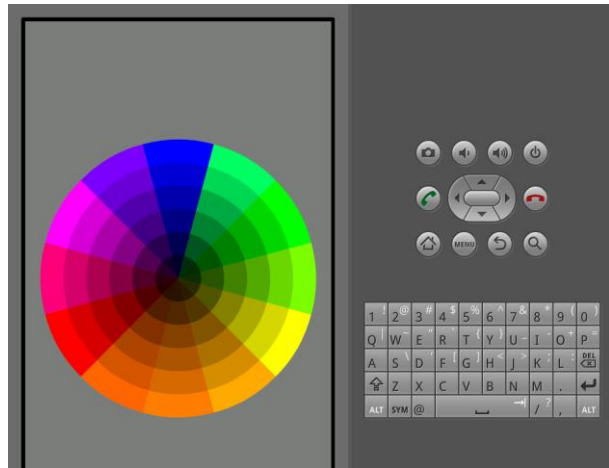
4.2 Android Mode in Processing



- **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



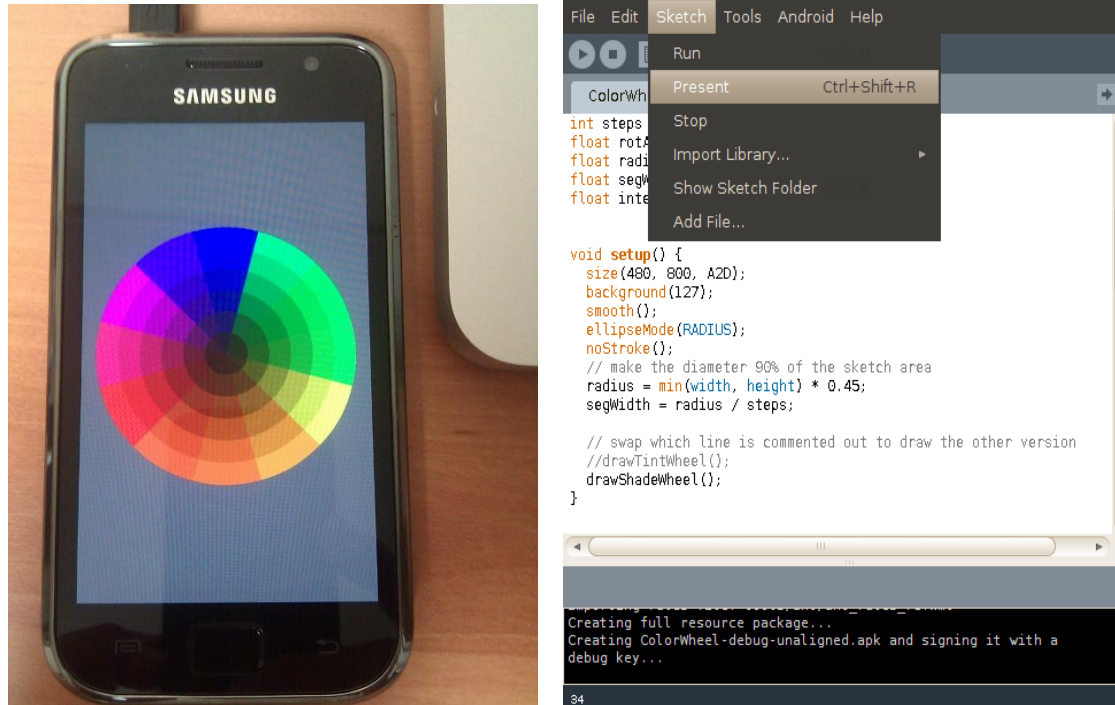
```
File Edit Sketch Tools Android Help
[Run] [Stop] [View] [Upload] [Download] [Refresh]
ColorWheel 5 [Run]
*/
int segs = 12;
int steps = 6;
float rotAdjust = TWO_PI / segs / 2;
float radius;
float segWidth;
float interval = TWO_PI / segs;

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(1);
}

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**.

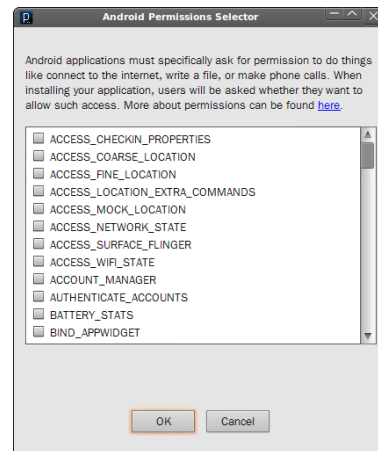
On Linux, you might need to run first the following commands from the terminal:

```
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



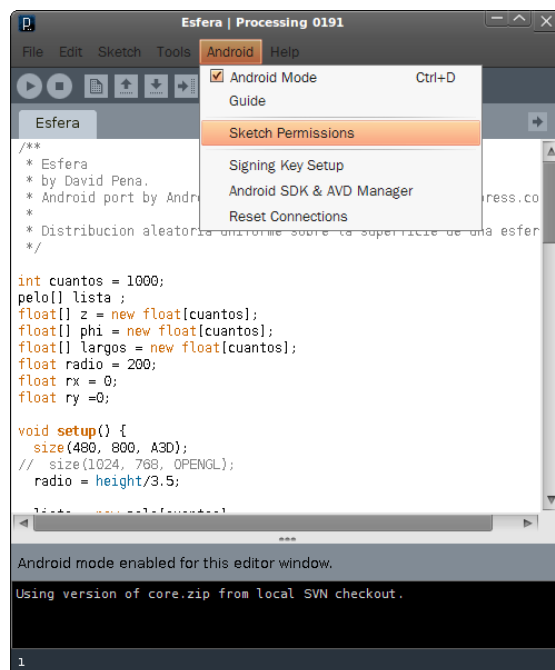
If you want to load data from the internet, or otherwise connect to other servers, you'll need to enable INTERNET permission for your sketch. To do so, use **Tools** → **Android Permissions** to bring up the permissions editor. Check the box next to internet.

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 or 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>

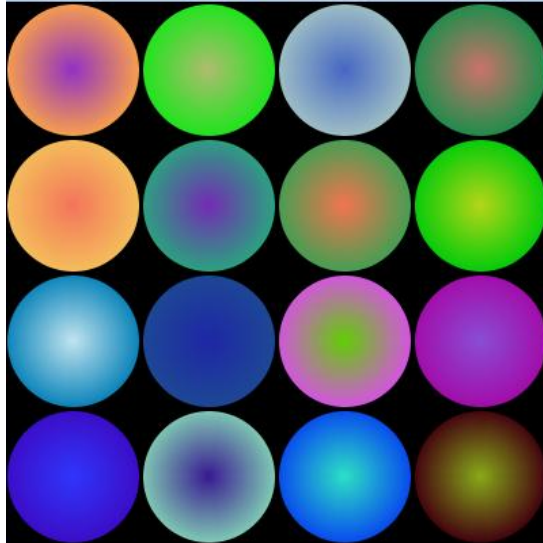


5. Basic Processing use

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>

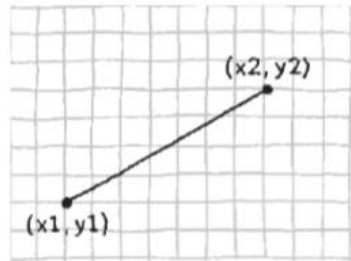
Functions and Parameters

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

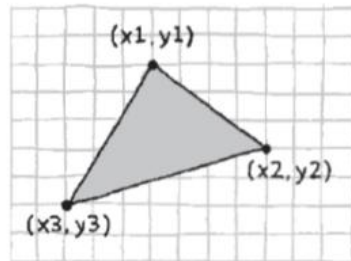
Color

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

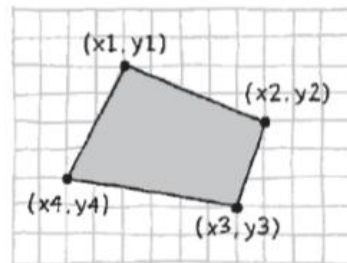
5.1 Drawing of 2D shapes and use of color



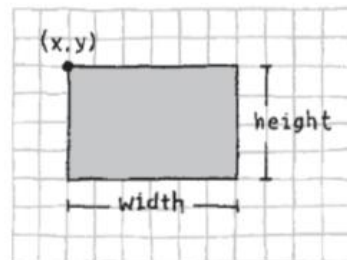
```
line(x1, y1, x2, y2)
```



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



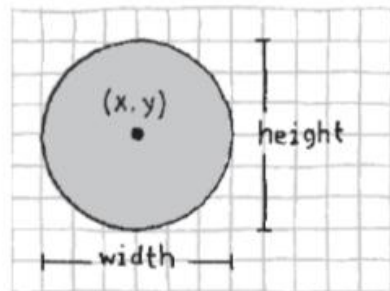
```
quad(x1, y1, x2, y2, x3, y3, x4, y4)
```



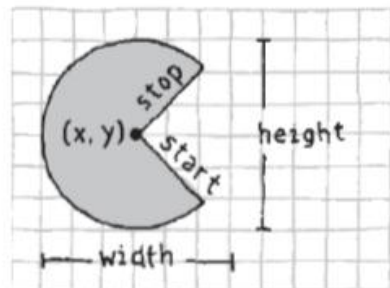
```
rect(x, y, width, height)
```

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

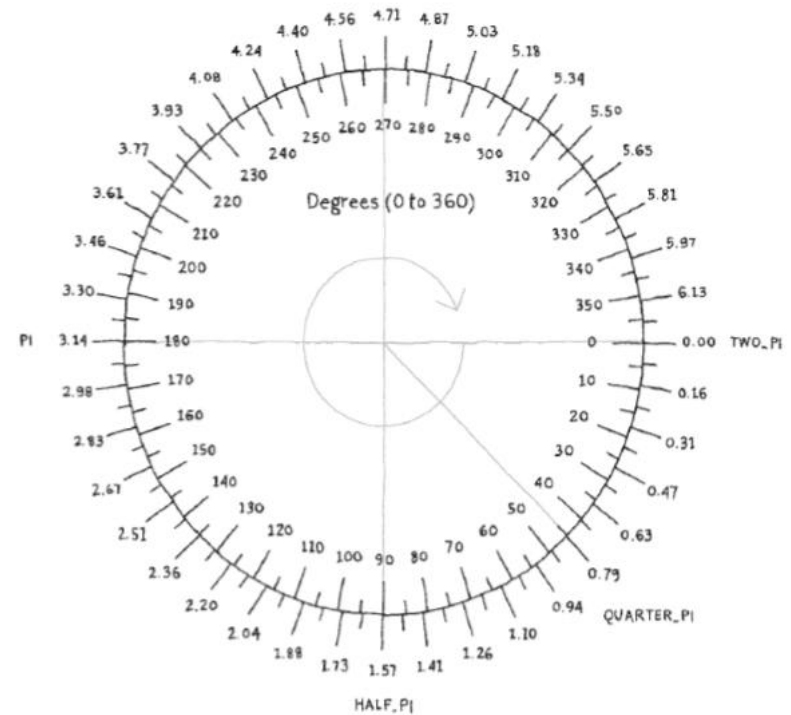
5.1 Drawing of 2D shapes and use of color



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



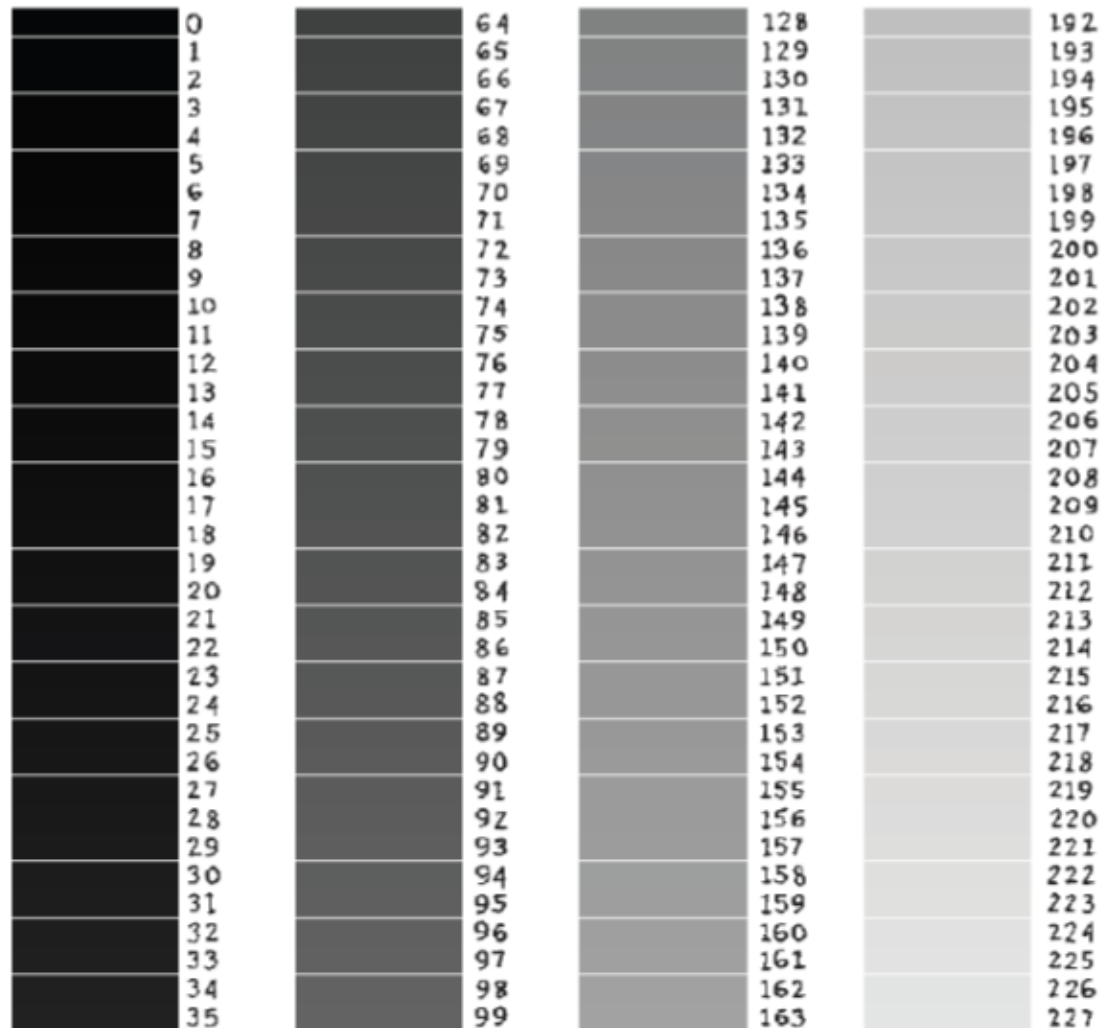
`arc(x, y, width, height, start, stop)`



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

5.1 Drawing of 2D shapes and use of color

Grayscale

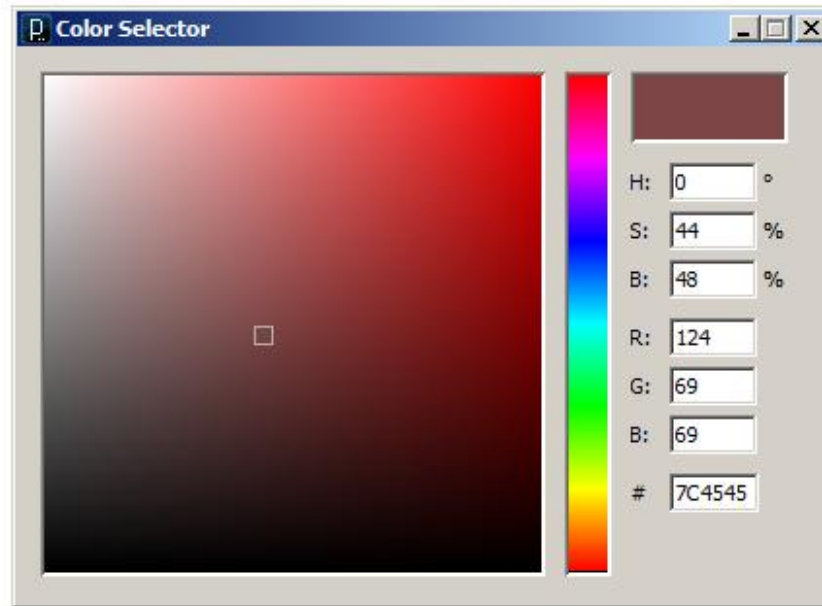


0(black) – 255(white)

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

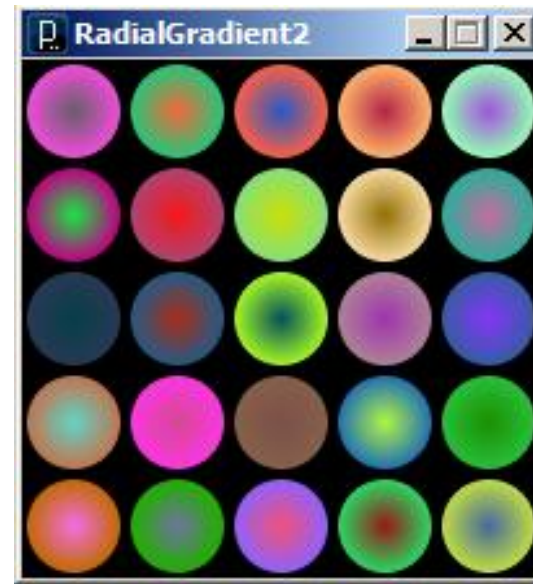
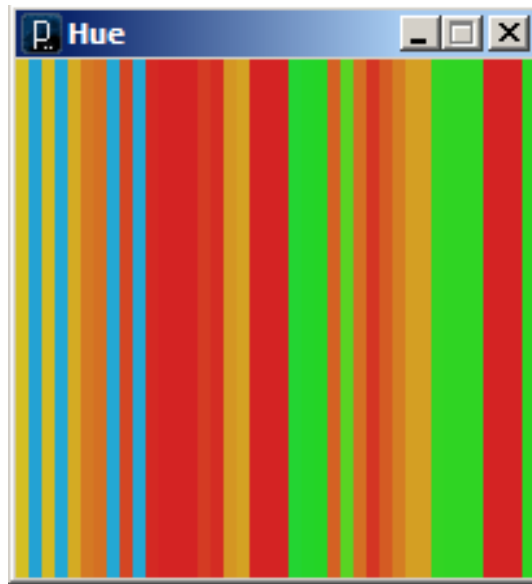
5.1 Drawing of 2D shapes and use of color

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");  
}
```

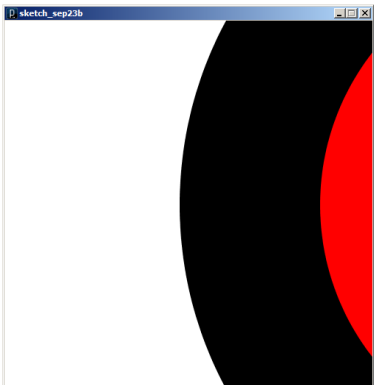
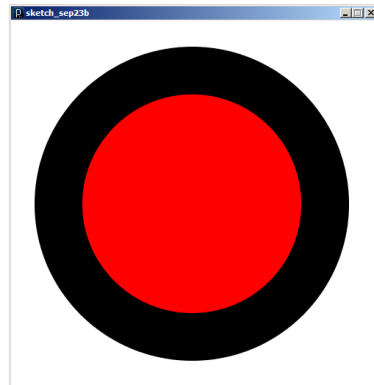
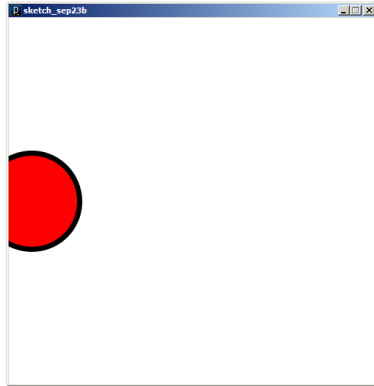
When this code is run, the following is written to the Console:

```
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

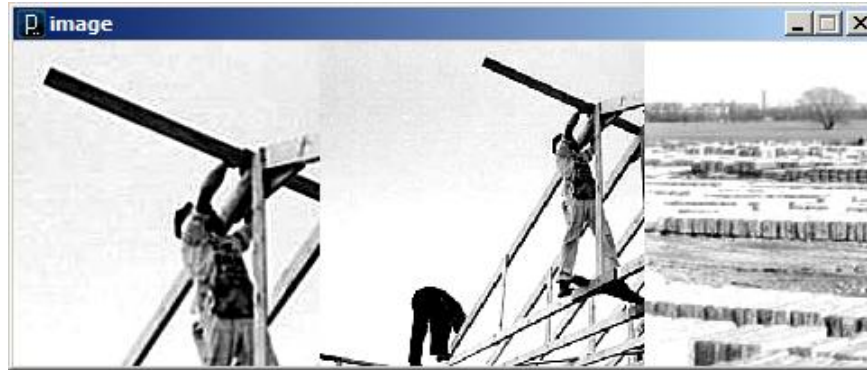


```
int x=0;

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)



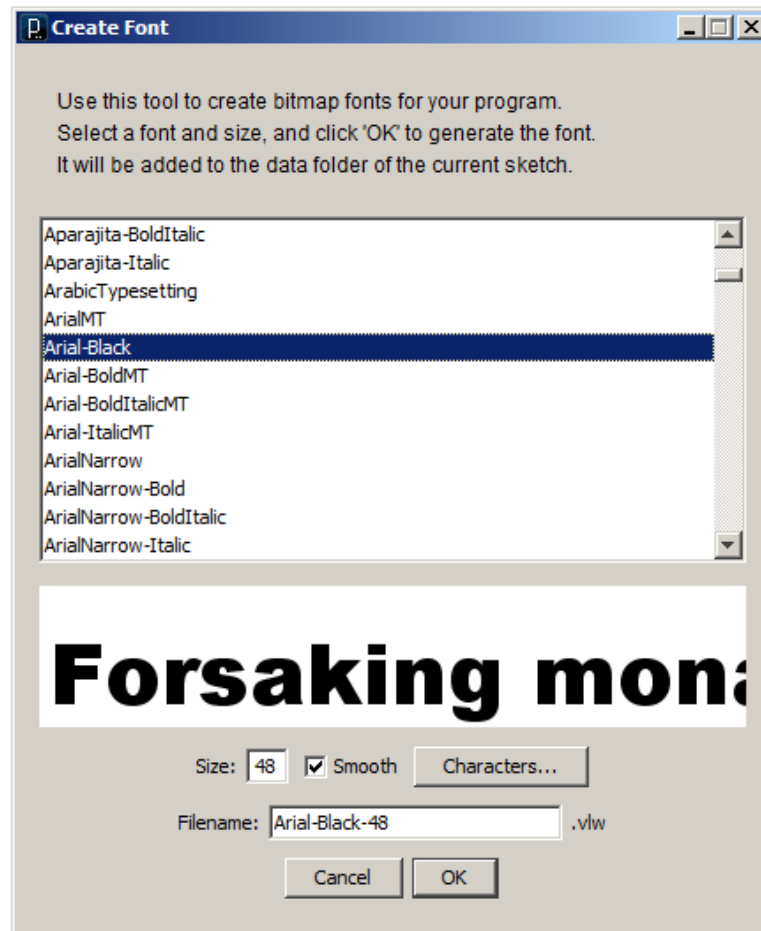
```
PImage img1;  
PImage img2;
```

```
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)

Touchscreen

```
mouseX  
mouseY  
mousePressed()  
mouseReleased()  
mouseMoved()  
mouseDragged()
```

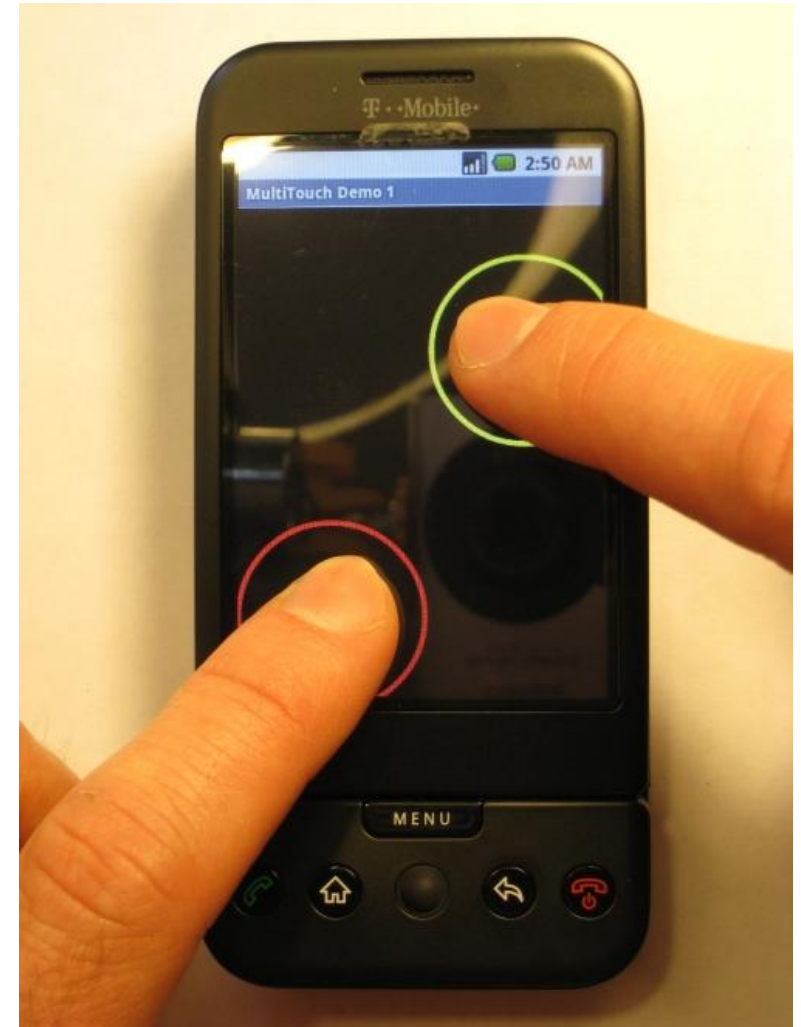
5.4 Interaction (keyboard, touchscreen, multitouch handling)

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/>




```

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

This part will be updated during the course.

PART II:

Fast 3D Graphics

in Processing for

Android



7. OpenGL and Processing

OpenGL ES

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

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

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

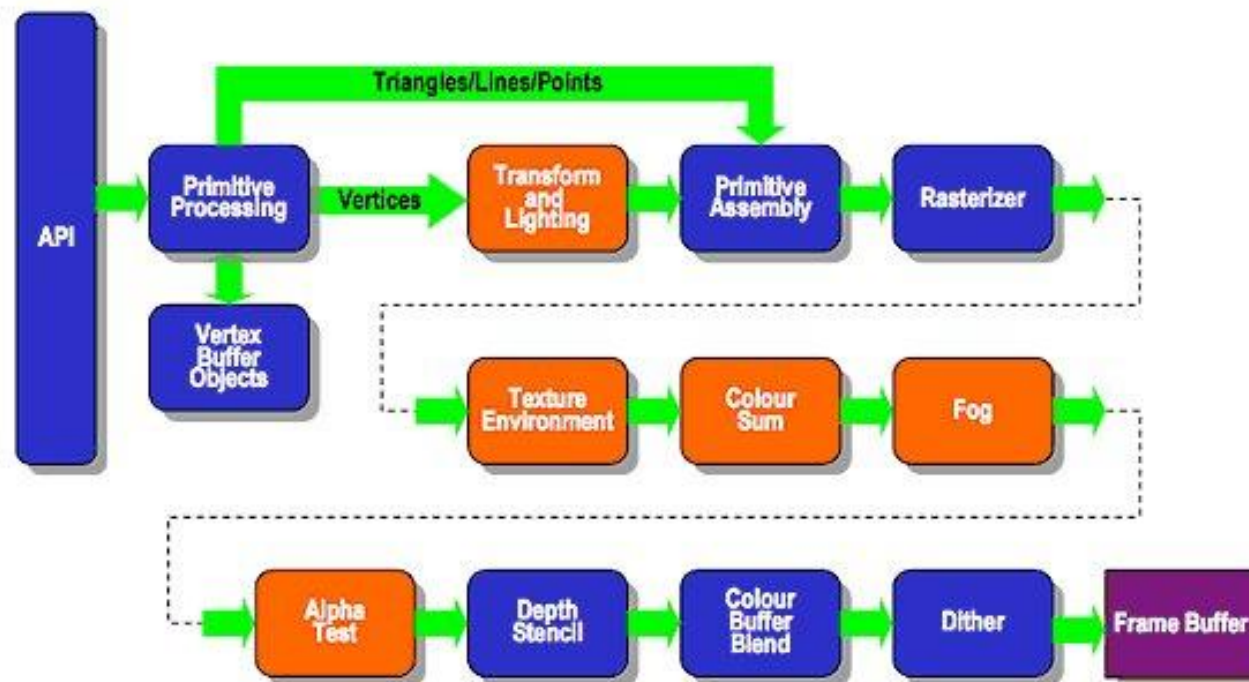
OpenGL ES consists in a subset of OpenGL.

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



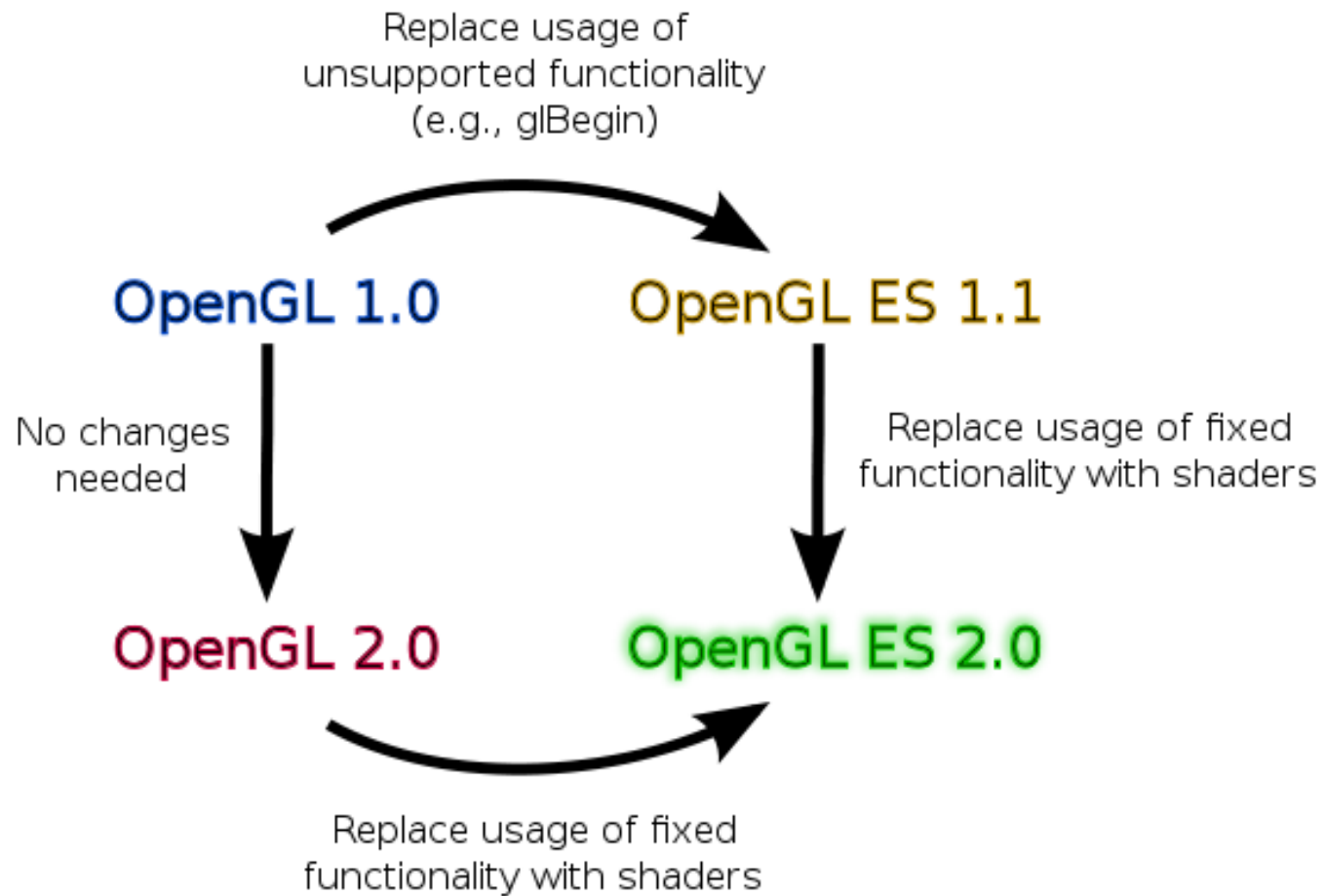
OpenGL ES is the 3D API for other platforms, such as Nokia and iPhone:

The graphics pipeline 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)

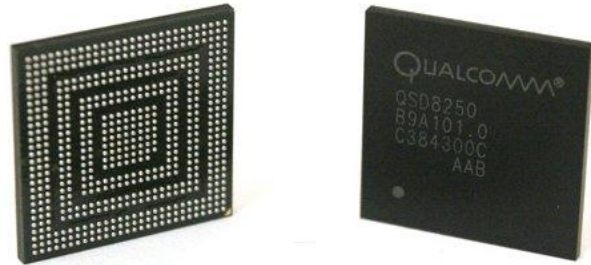
Relationship between OpenGL and OpenGL ES



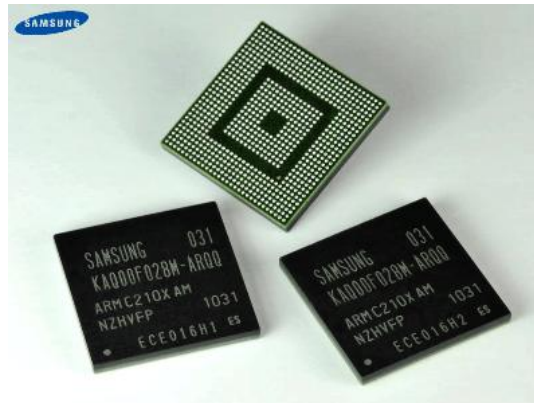
Source: <http://wiki.maemo.org/OpenGL-ES>

Mobile GPUs

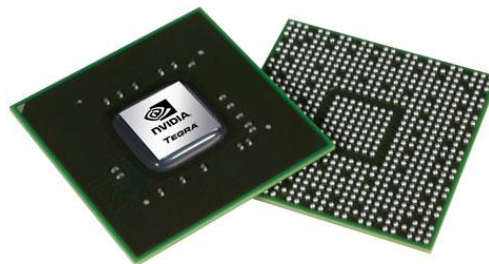
The main GPUs manufacturers for mobile devices are Qualcomm, PowerVR and NVidia.



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

Useful websites for benchmark information about mobile GPUs:

<http://smartphonebenchmarks.com/>

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
2. GPUs such as the Adreno 200 or PowerVR SVG 540/530 are recommended.
3. Older GPUs found on devices such as the G1 might work, but the performance is limited.
4. As a general requirement for Processing, Android 2.1. However, certain OpenGL features were missing in 2.1, so froyo (2.2) is needed to take full advantage of the hardware.

The A3D renderer

1. In Processing for Android there is no need to use OpenGL ES directly (although it is possible).
2. The drawing API in Processing uses OpenGL internally when selecting the A3D (Android 3D) renderer.
3. The renderer in Processing is the module that executes all the drawing commands.
4. During the first part of this workshop we used the A2D renderer, which only supports 2D drawing.
5. 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.

Offscreen drawing

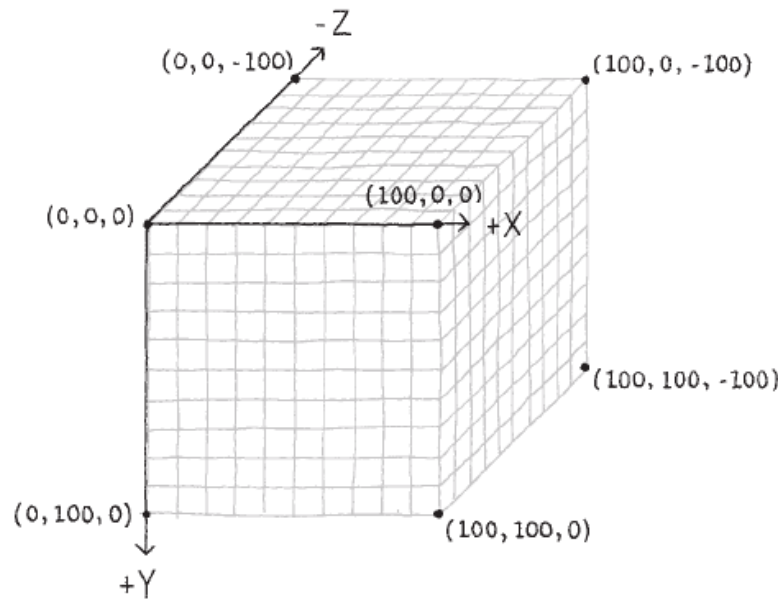
We can create an offscreen A3D surface by using the `createGraphics()` method:

```
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.getOffscreenImage());  
    ...  
}
```


8. Geometrical transformations



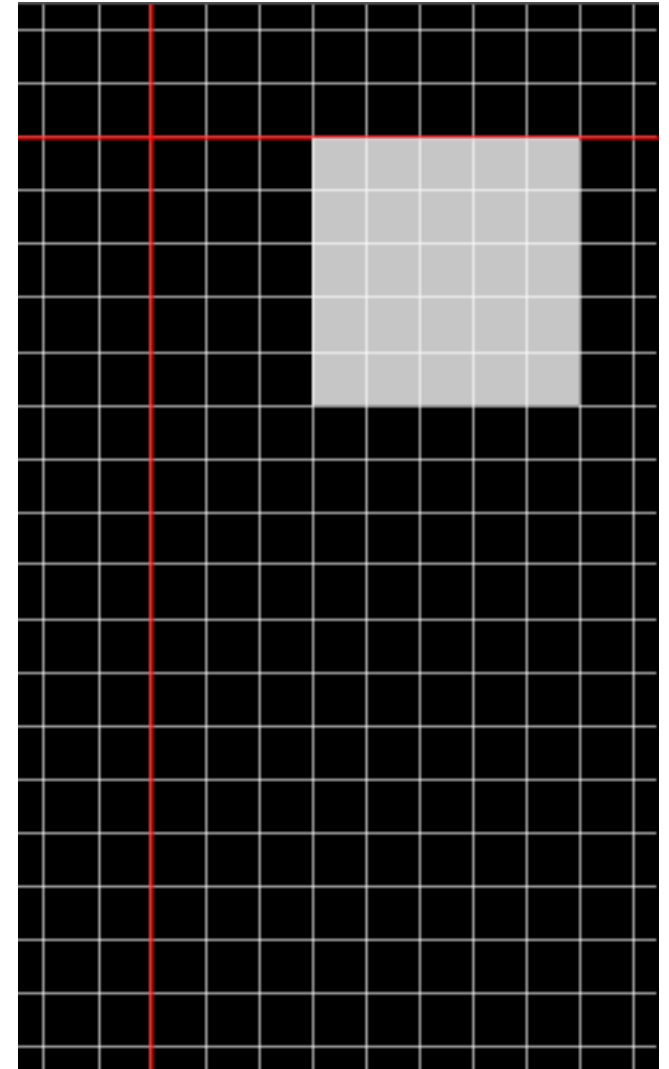
Casey Reas and Ben Fry.
<Getting Started with Processing>.
O'Really Media, 2010

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);  
}
```

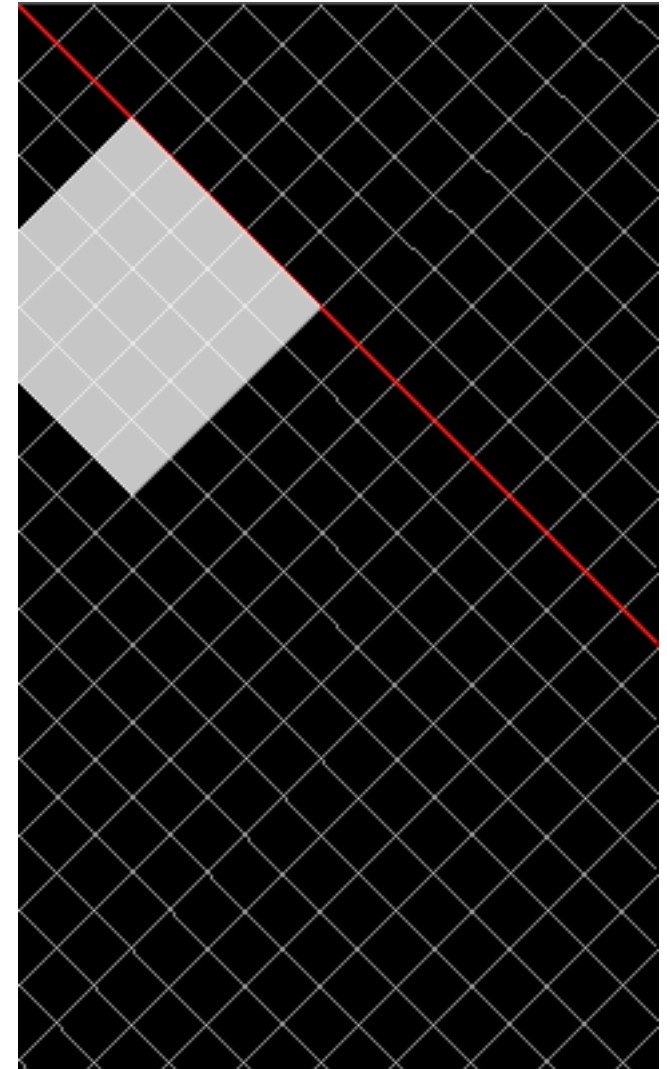


Rotations

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)
```

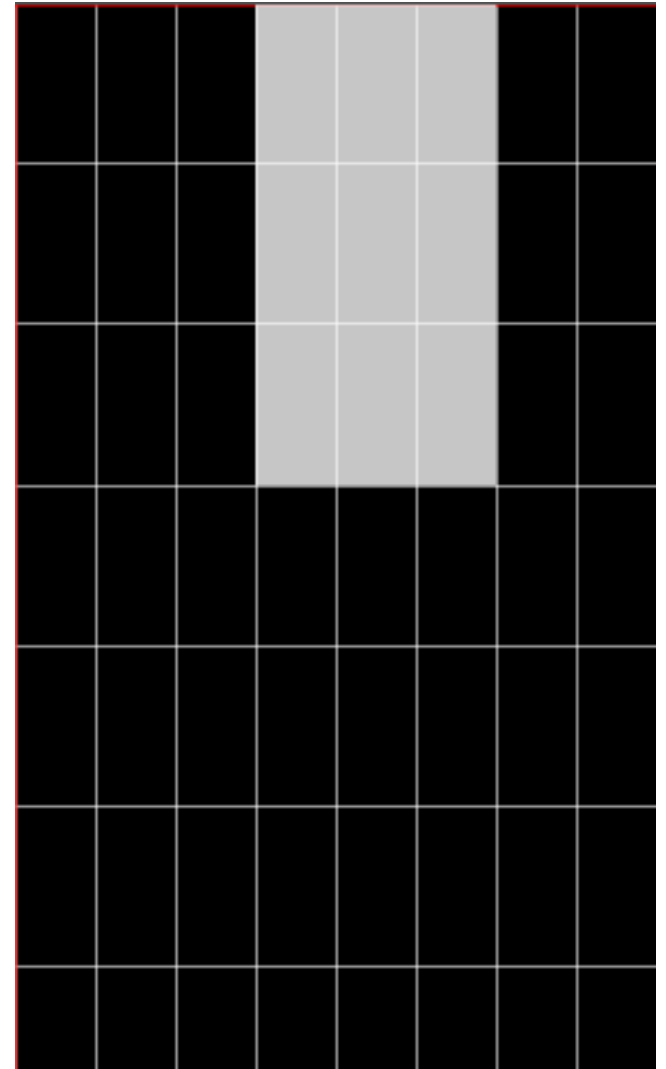
```
void setup() {  
  size(240, 400, A3D);  
  stroke(255, 150);  
}  
  
void draw() {  
  background(0);  
  rotateZ(PI / 4);  
  noStroke();  
  fill(255, 200);  
  rect(60, 0, 100, 100);  
}
```



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.

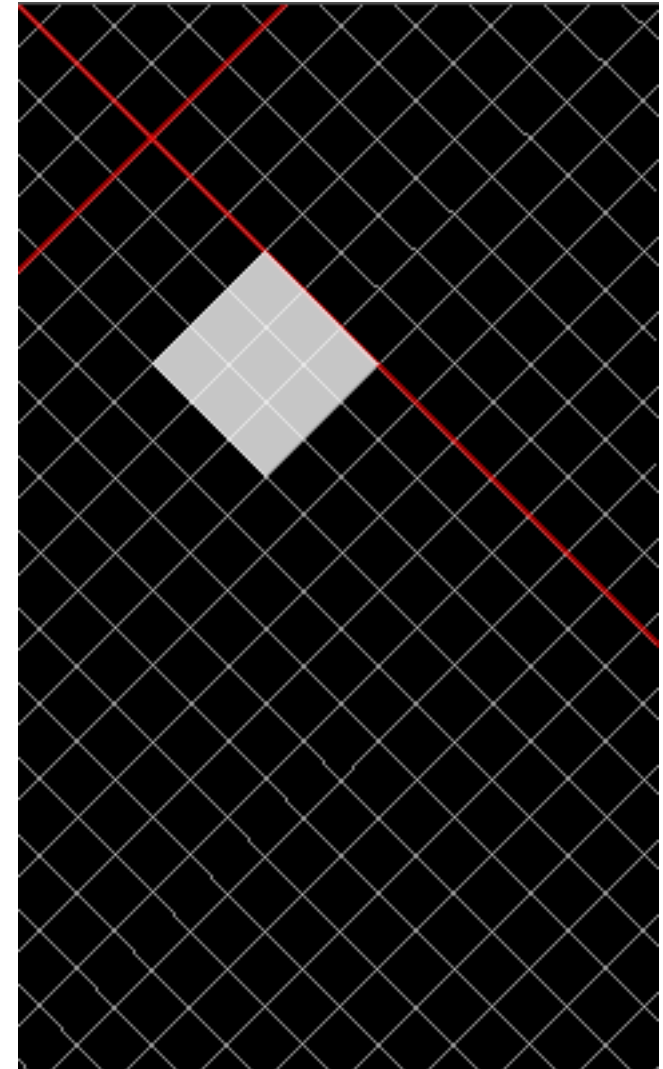
```
void setup() {  
  size(240, 400, A3D);  
  stroke(255, 150);  
}  
  
void draw() {  
  background(0);  
  scale(1.5, 3.0, 1.0);  
  noStroke();  
  fill(255, 200);  
  rect(60, 0, 60, 60);  
}
```



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

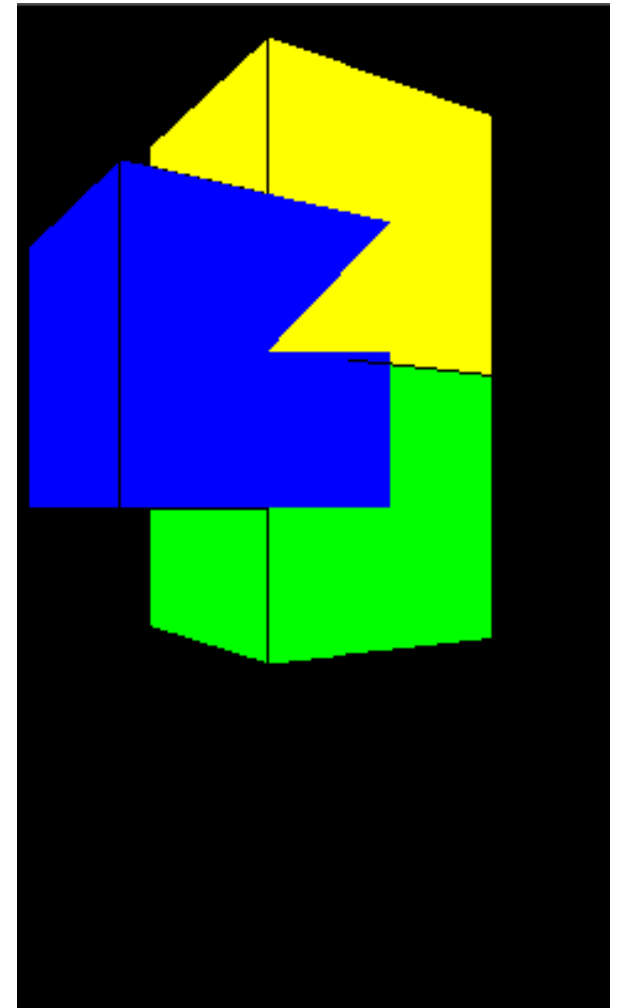
```
void setup() {  
  size(240, 400, A3D);  
  stroke(255, 150);  
}  
  
void draw() {  
  background(0);  
  translate(50, 50, 0);  
  rotateZ(PI / 4);  
  noStroke();  
  fill(255, 200);  
  rect(60, 0, 60, 60);  
}
```



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.

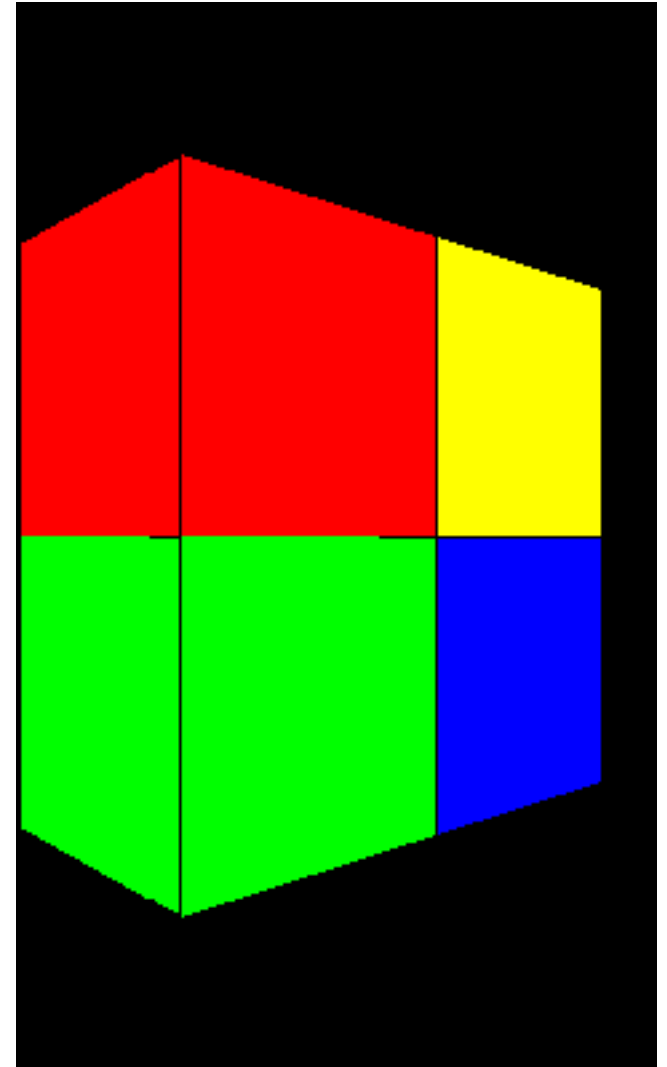
```
void setup(){
  size(240, 400, A3D);
}
void draw(){
  background(0);
  translate(width/2, height/2);
  rotateY(frameCount*PI/60);
  translate(-50, -50);
  fill(255, 0, 0);
  box(100, 100, 100);
  translate(50, -50);
  fill(255, 255, 0);
  box(100, 100, 100);
  translate(-50, 50);
  fill(0, 0, 255);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



```

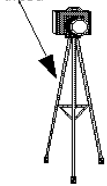
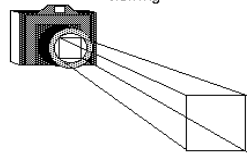
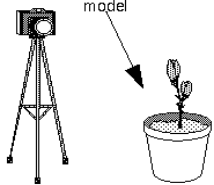
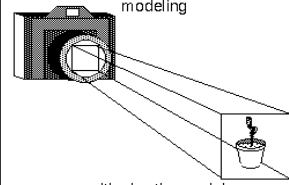
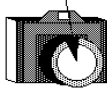
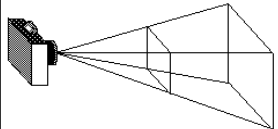
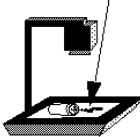

void setup(){
  size(240, 400, A3D);
}
void draw(){
  background(0);
  translate(width/2, height/2);
  rotateY(frameCount*PI/60);
  pushMatrix();
  translate(-50, -50);
  fill(255, 0, 0);
  box(100, 100, 100);
  popMatrix();
  pushMatrix();
  translate(50, -50);
  fill(255, 255, 0);
  box(100, 100, 100);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

```



9. Camera and perspective

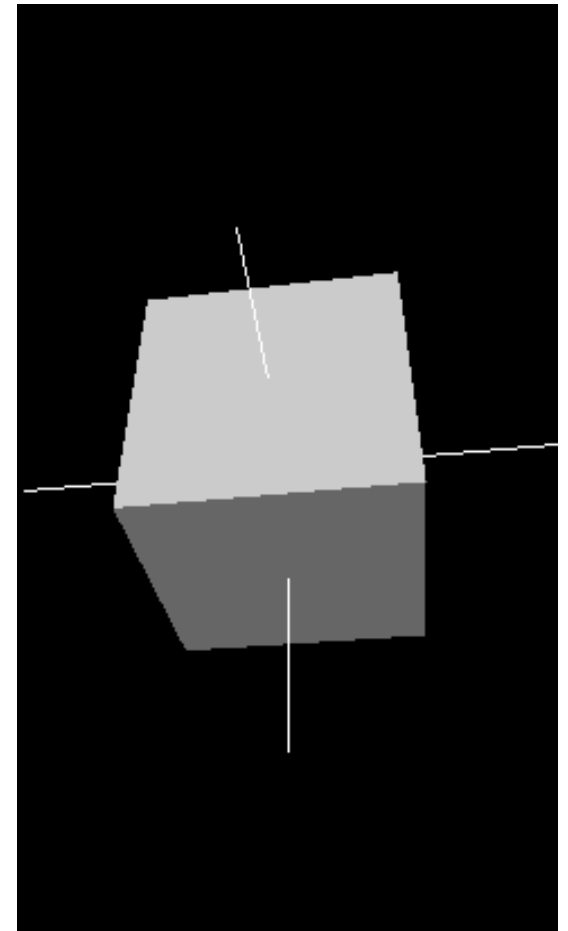
1. Configuring the view of the scene in A3D requires setting the camera location and the viewing volume.
2. This can be compared with setting a physical camera in order to take a picture:

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

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`

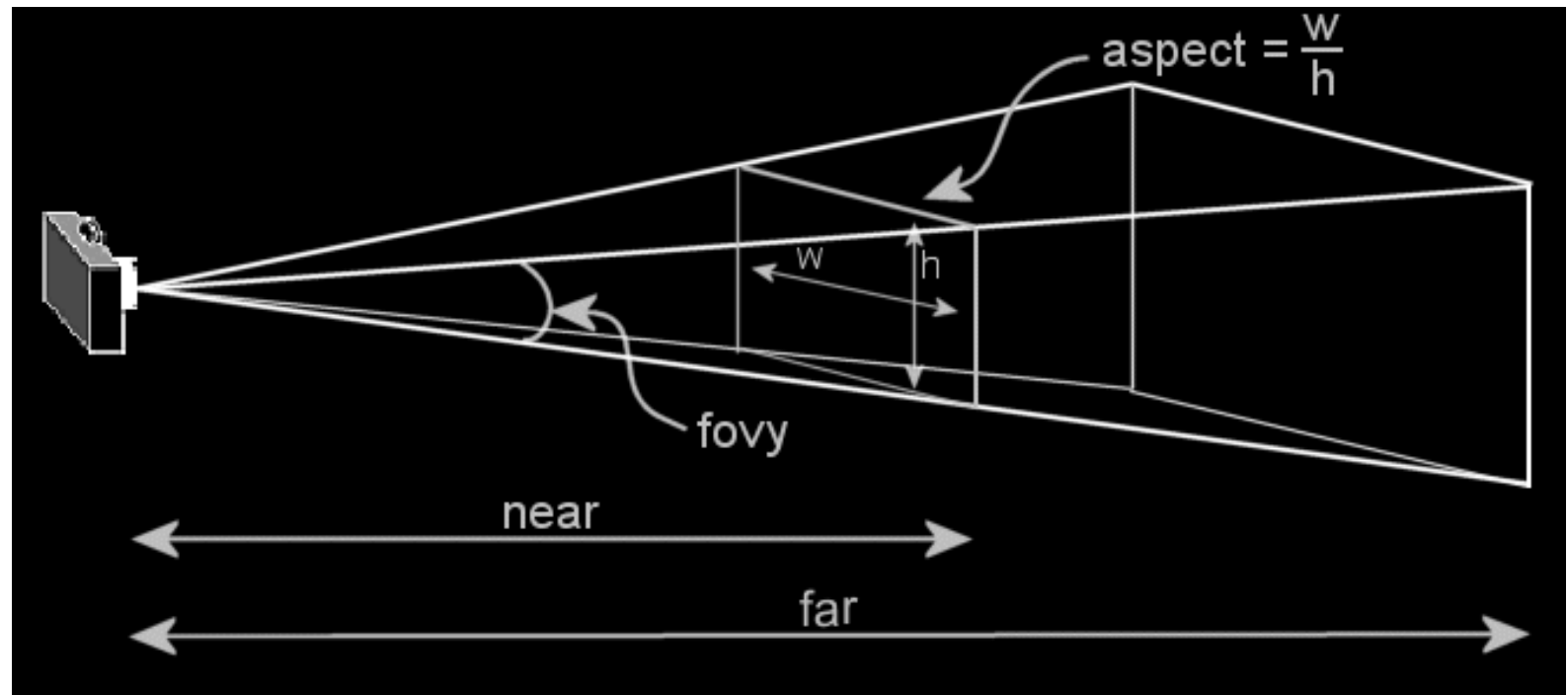
```
void setup() {  
  size(240, 400, A3D);  
  fill(204);  
}  
  
void draw() {  
  lights();  
  background(0);  
  camera(30.0, mouseY, 220.0,  
         0.0, 0.0, 0.0,  
         0.0, 1.0, 0.0);  
  noStroke();  
  box(90);  
  stroke(255);  
  line(-100, 0, 0, 100, 0, 0);  
  line(0, -100, 0, 0,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

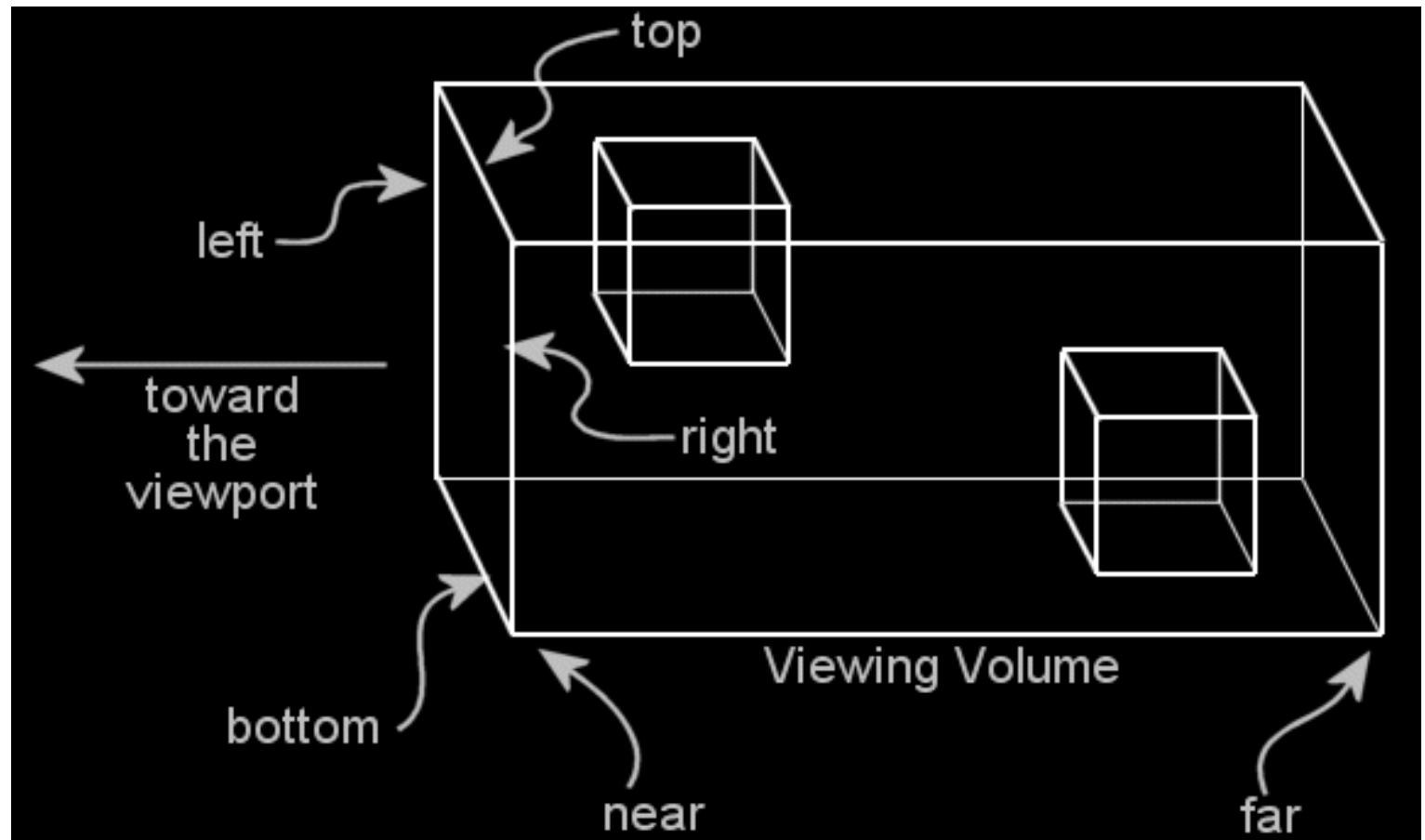


`perspective(fov, aspect, zNear, zFar)`

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

```

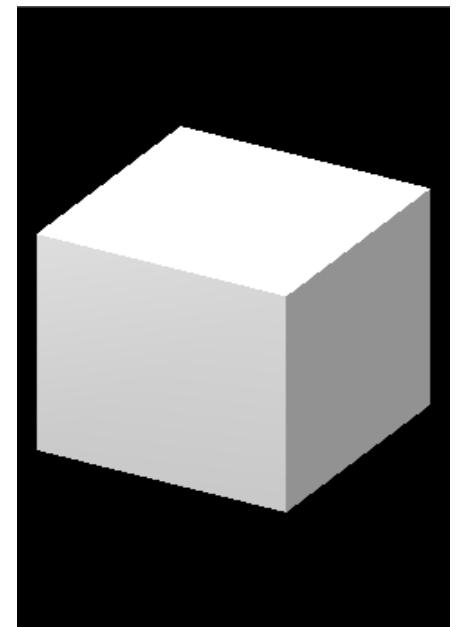
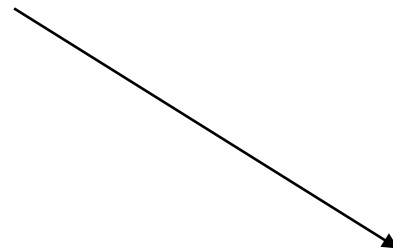
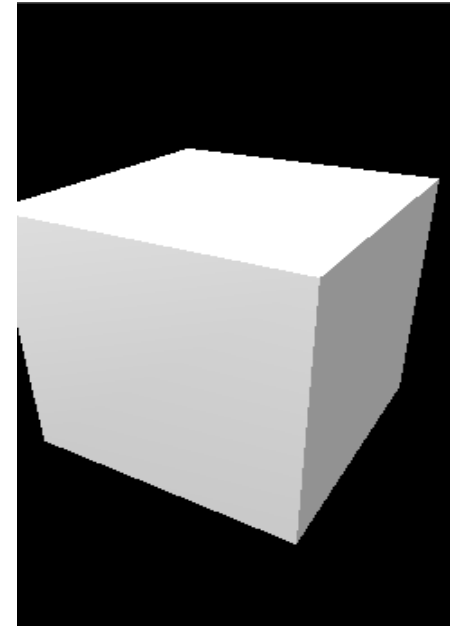
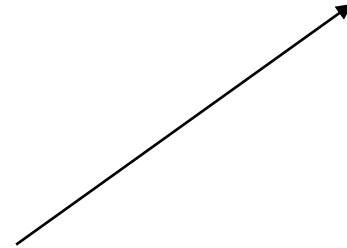
void setup() {
  size(240, 400, A3D);
  noStroke();
  fill(204);
}

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);
}

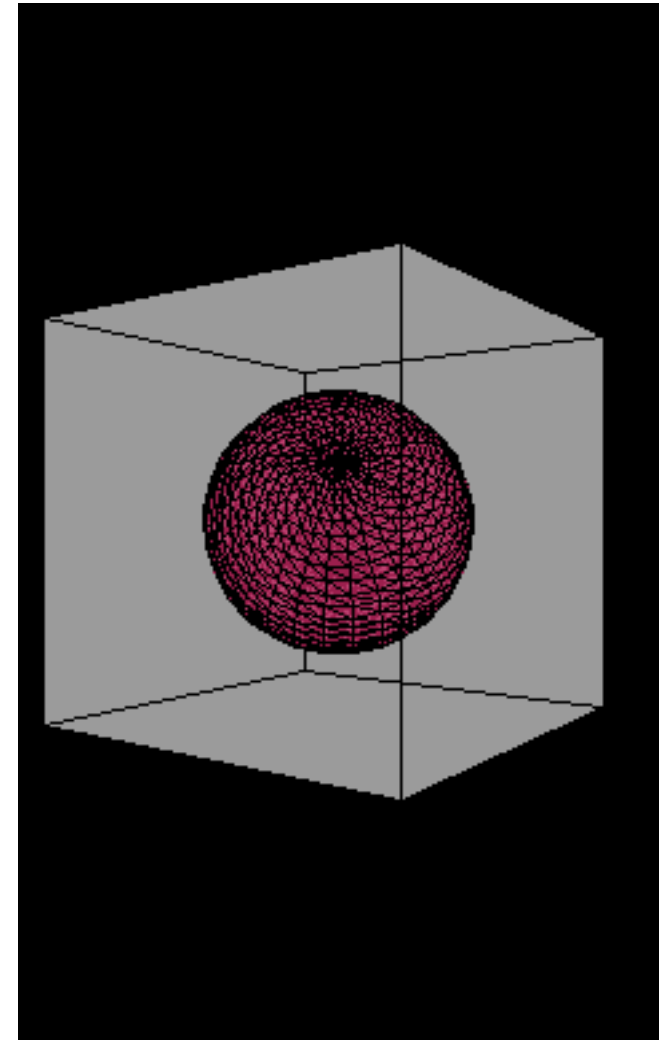
```



10. Creating 3D objects

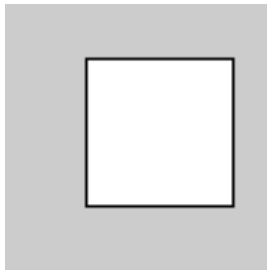
A3D provides some functions for drawing predefined 3D primitives:
sphere(r), box(w, h, d)

```
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();  
}
```



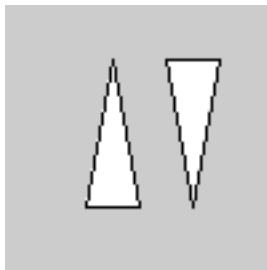
beginShape()/endShape()

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.



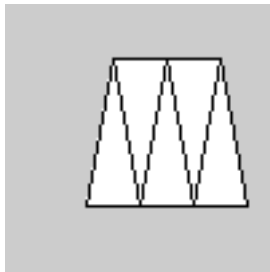
```
beginShape();  
vertex(30, 20, 0);  
vertex(85, 20, 0);  
vertex(85, 75, 0);  
vertex(30, 75, 0);  
endShape(CLOSE);
```

Closed polygon



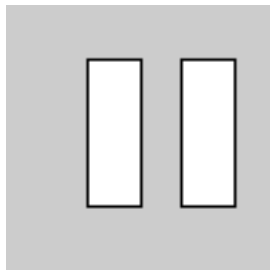
```
beginShape(TRIANGLES);  
vertex(30, 75, 0);  
vertex(40, 20, 0);  
vertex(50, 75, 0);  
vertex(60, 20, 0);  
vertex(70, 75, 0);  
vertex(80, 20, 0);  
endShape();
```

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



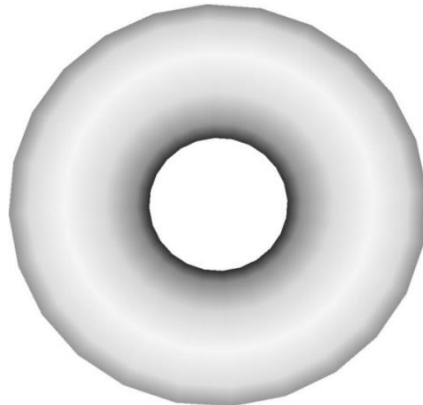
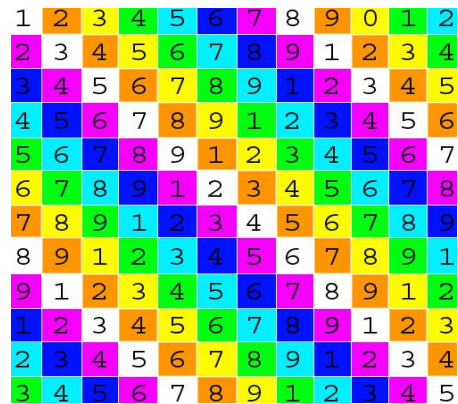
```
beginShape (QUADS);  
vertex(30, 20, 0);  
vertex(30, 75, 0);  
vertex(50, 75, 0);  
vertex(50, 20, 0);  
vertex(65, 20, 0);  
vertex(65, 75, 0);  
vertex(85, 75, 0);  
vertex(85, 20, 0);  
endShape();
```

Individual quads

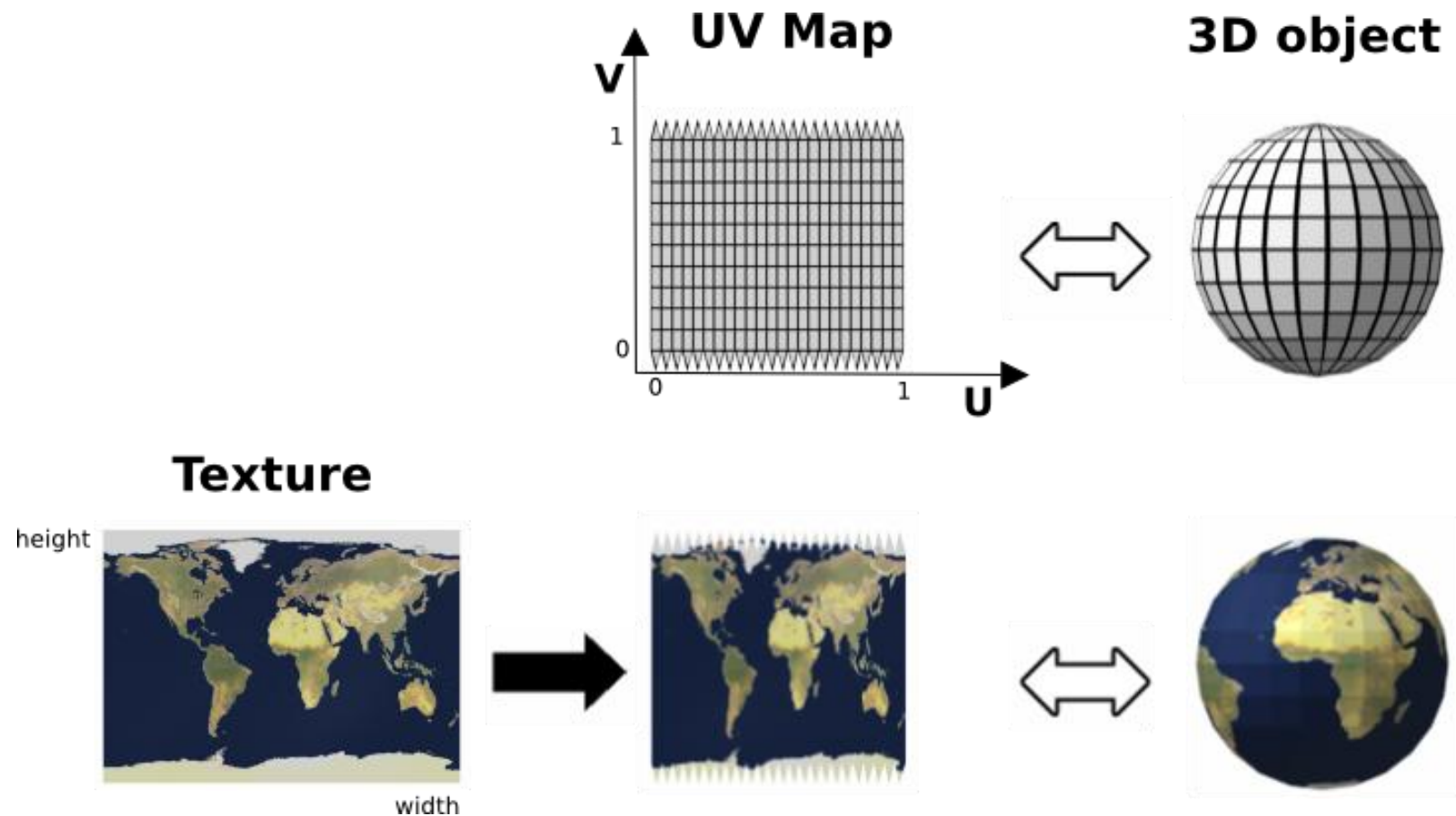
Check the Processing reference for more details:
http://processing.org/reference/beginShape_.html

Texturing

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



Basic texture mapping:



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

Texture mapping becomes a very complex problem when we need to texture complicated tridimensional shapes (organic forms).

Finding the correct mapping from 2D image to 3D shape requires mathematical techniques that takes into account edges, folds, etc.

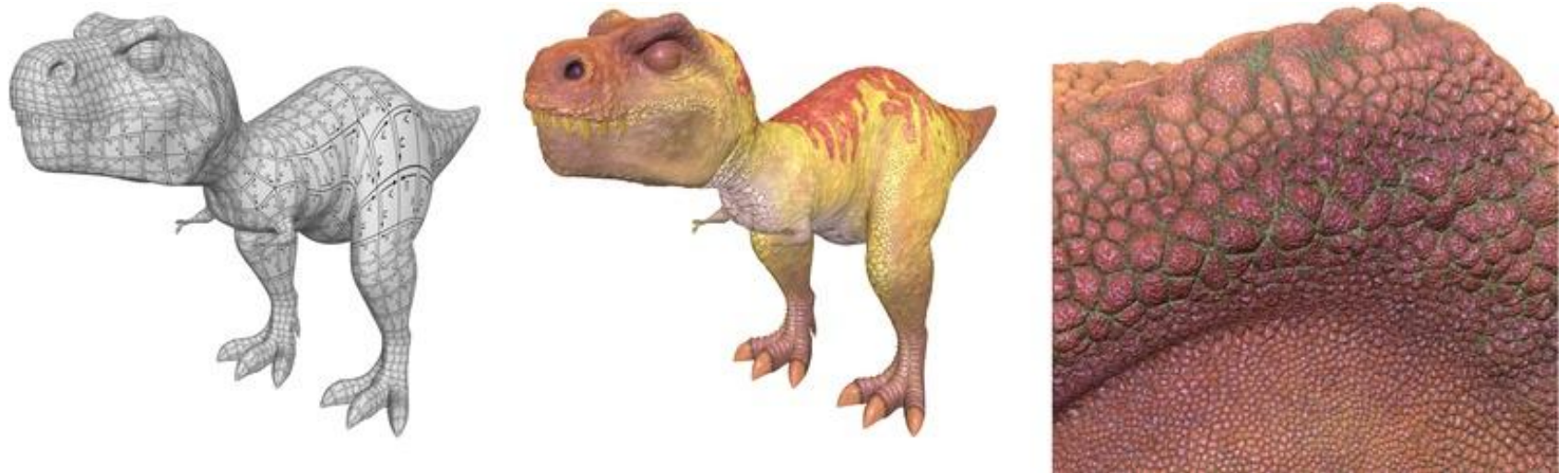


Image from: Ptex: Per-Face Texture Mapping for Production Rendering, by Brent Burley and Dylan Lacewell

Simple shape texturing

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

```
PImage img;
```

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

The texture mode can be
NORMAL or IMAGE

```
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();  
}
```

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("pebbles.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>

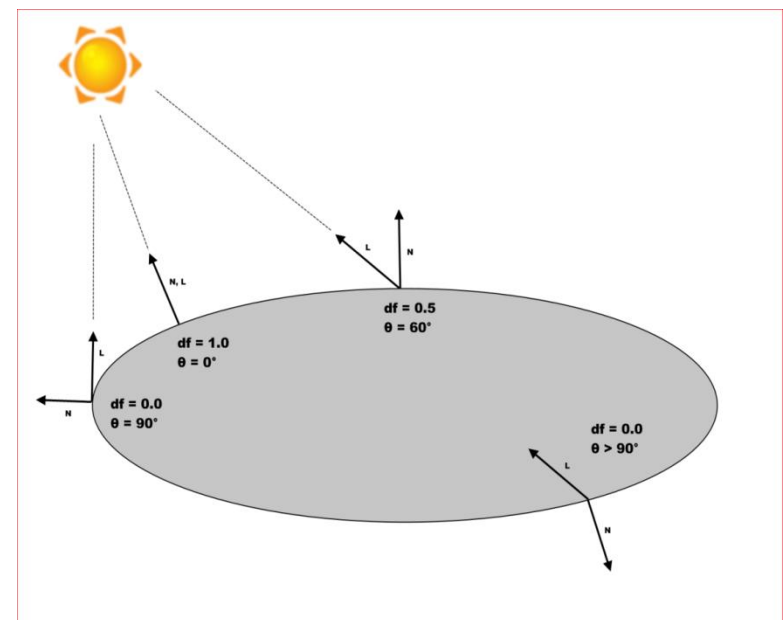
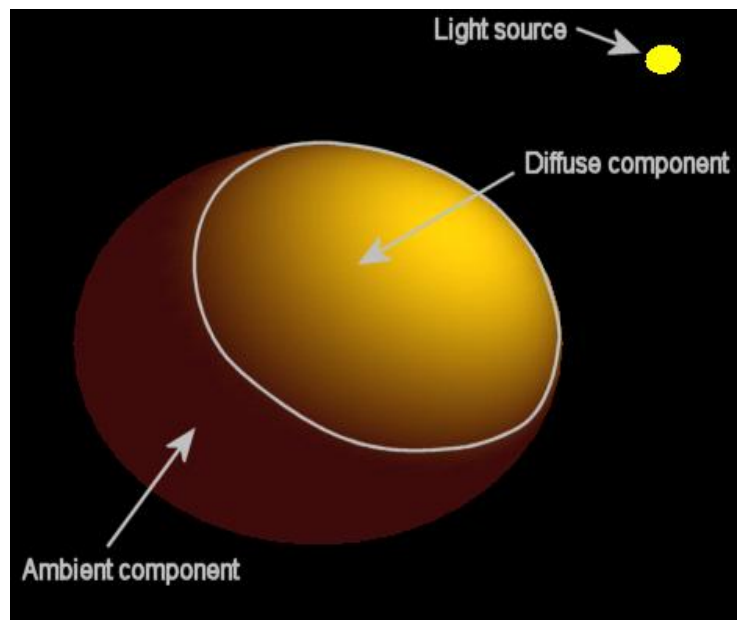
Some more good resources about lights in OpenGL:

<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

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

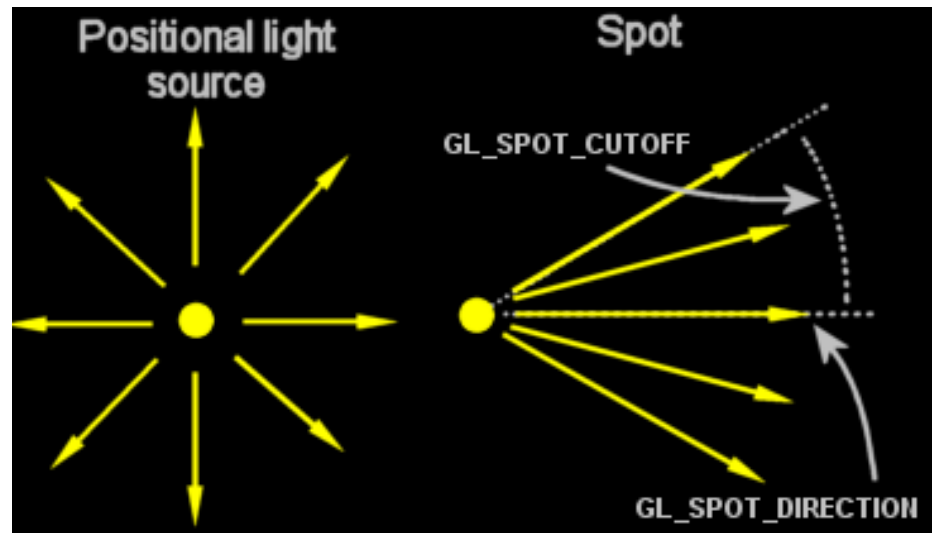
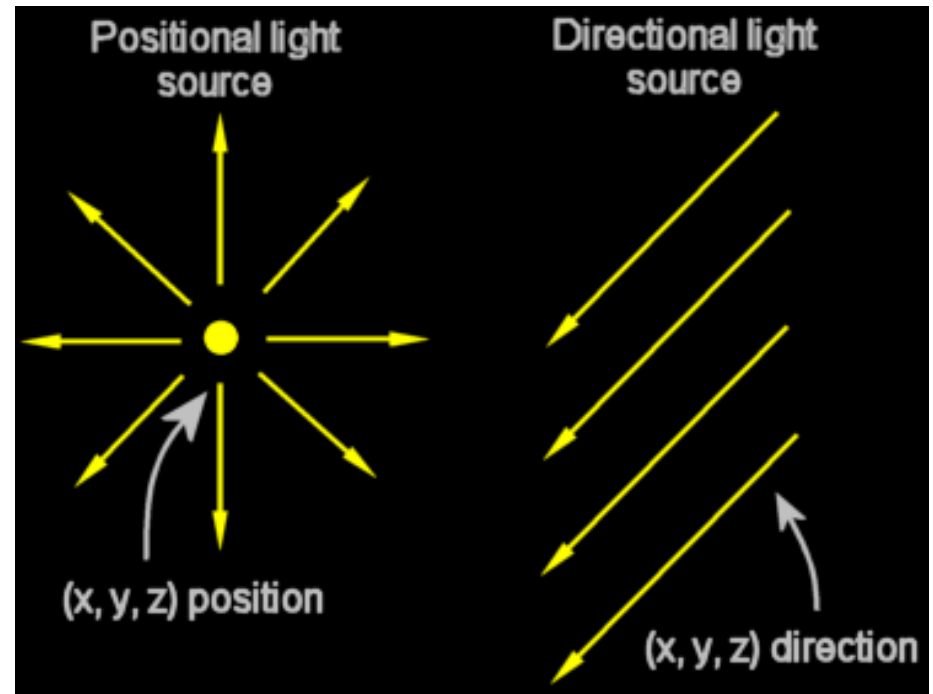
`x, y, z` position:

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

`angle` float: angle the spotlight cone

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



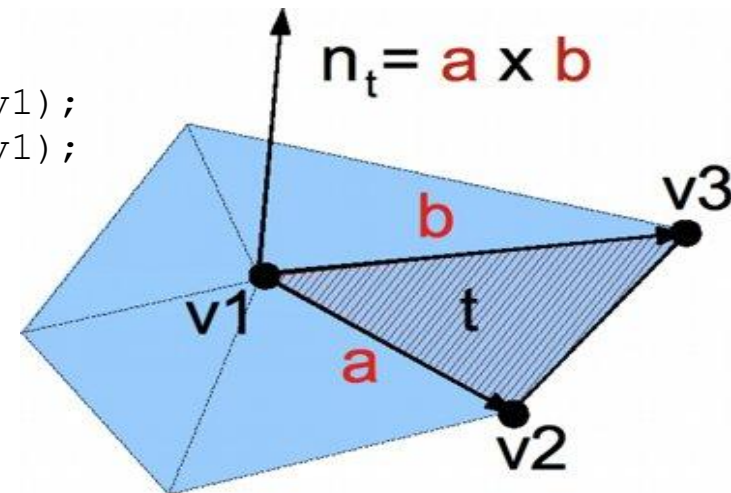


<http://jerome.jouvie.free.fr/OpenGL/Lessons/Lesson6.php>

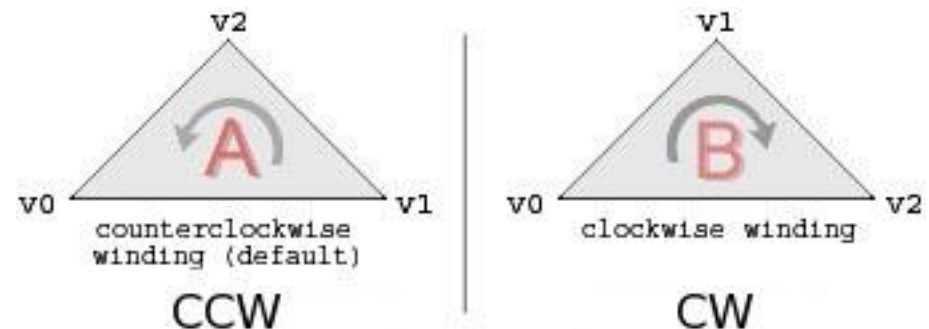
Normals: each vertex needs to have a normal defined so the light calculations can be performed correctly

```
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.



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();
  }
}

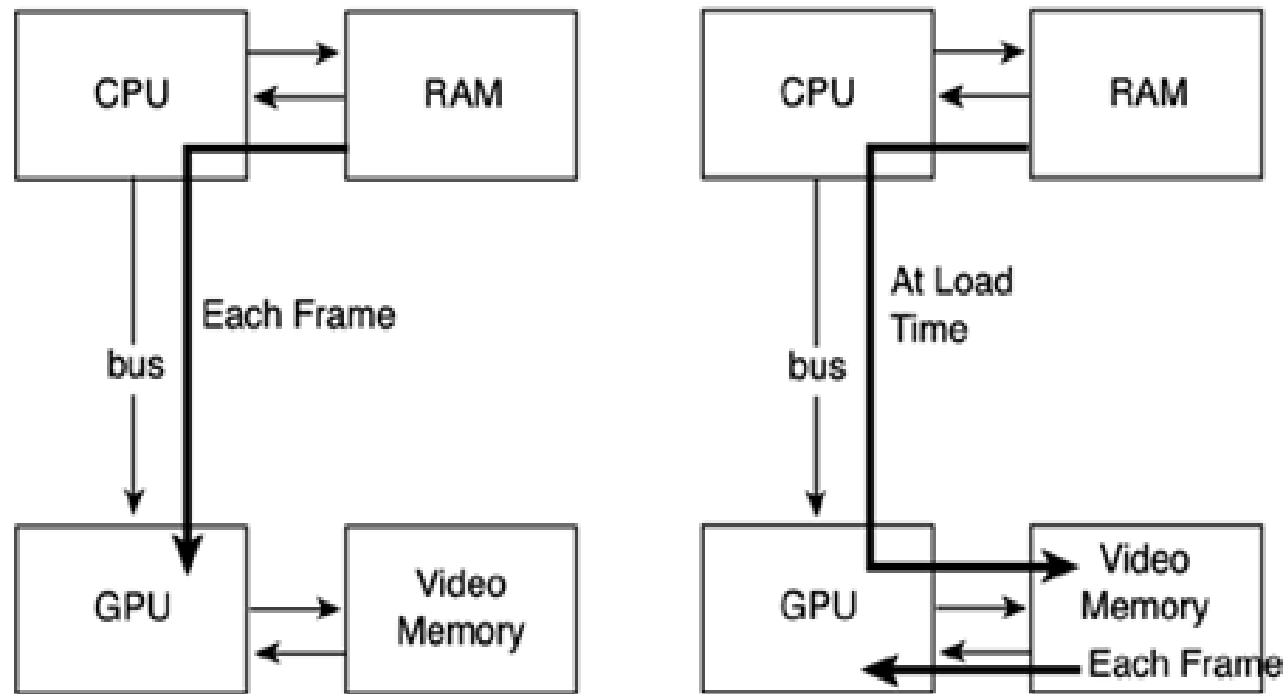
```



12 Models: the Pshape3D class

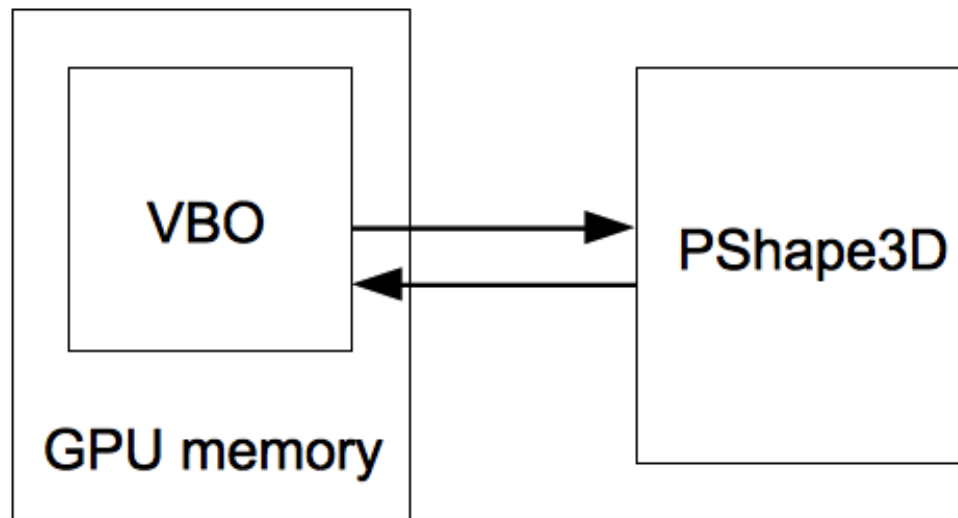
Vertex Buffer Objects

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

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 divided in groups, to facilitate normal and color assignment, texturing and creation of complex shapes with multiple geometry types (line, triangles, etc).

Manual creation of Pshape3D models

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 CCW vertex ordering.

Creation

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

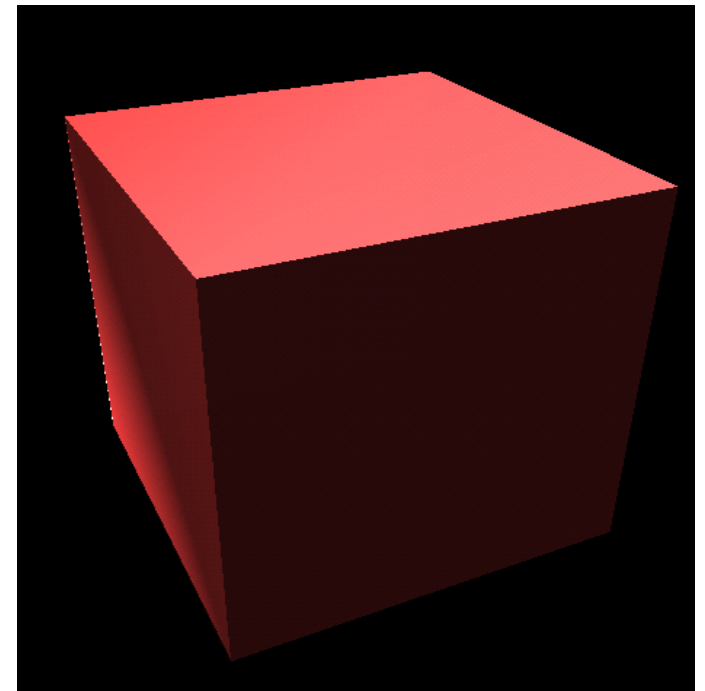
cube.beginUpdate(VERTICES);
cube.setVertex(0, -100, +100, -100);
cube.setVertex(1, -100, -100, -100);
...
cube.endUpdate();

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

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

Drawing

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

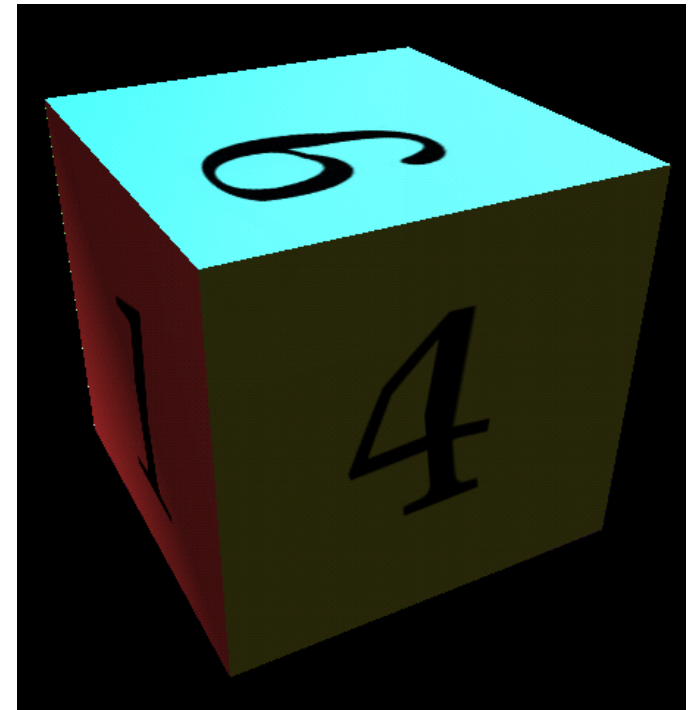


1. A PShape3D can be textured with one or more images.
2. The vertices can be organized in groups, which allows to assign a different texture to each group.
3. Groups also facilitate the assignment of colors and normals.

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

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

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



OBJ loading

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);  
}
```

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. There are two ways of using shape recording: one is saving a single shape with beginShapeRecorder/endShapeRecorder, and the second allows saving multiple shapes with beginShapesRecorder/endShapesRecorder

```
PShape recShape;

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

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

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

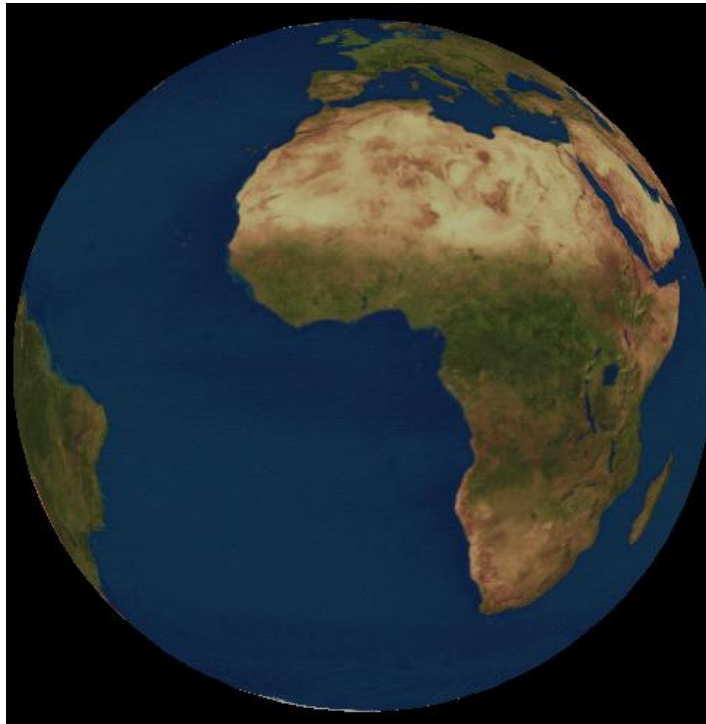
```
objects recShape;

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

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

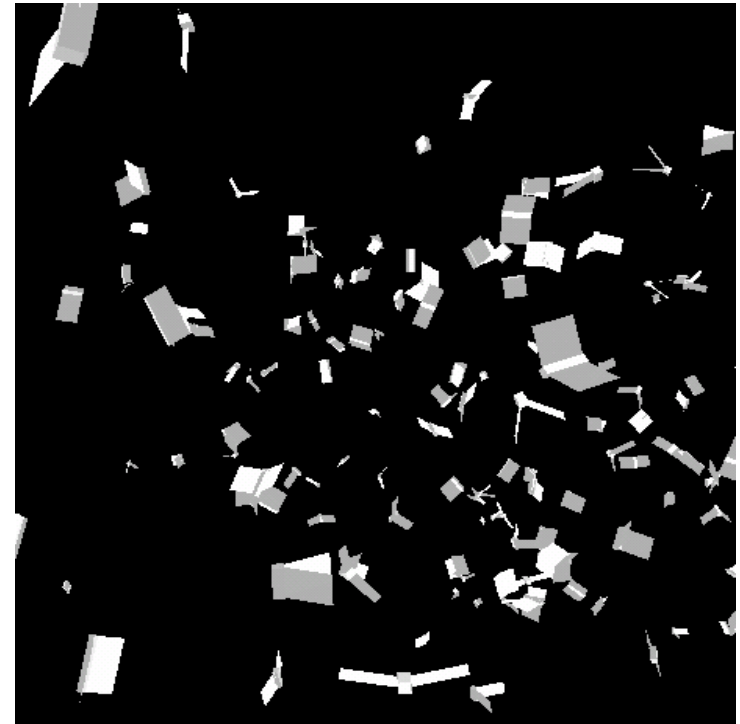

The performance gains of using shape recording are quite substantial.
It usually increases the rendering framerate by 100% or more.

Textured sphere



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

Birds flock



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

Particle systems

PShape3D allows to create models with the POINT_SPRITES geometry type.

With this type, each vertex is treated as a textured point sprite.

At least one texture must be attached to the model, in order to texture the sprites.

More than sprite texture can be attached, by dividing the vertices in groups.

The position and color of the vertices can be updated in the draw loop in order to simulate motion (we have to create the shape as DYNAMIC).

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

Creation/initialization

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

Dynamic update

Wrapping up...

Lets look an example where we do:

1. offscreen rendering to texture
2. particle system (rendered to offscreen surface)
3. dynamic texturing of a 3D shape using the result of the offscreen drawing

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/>

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

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

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

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

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**
SIGGRAPHASIA2010

Andres Colubri (andres.colubri@gmail.com)
Jihyun Kim (kimjihyunn@gmail.com)