# Chapter

Purpose

* Self-Education
* Broader Educational

Status

* Describe Functionality

Nature

* Image Recognition is lion-share
* Rather easy compared to more real world situations.
* However, this app not ready for broad use:
  + Variation of Rubik Cubes (Shape and Colors)
  + Variation of Luminous
  + Variation of Devices : libraries, cpu’s, gpu’s
  + Variation of User Guesters
* Real target is Smart Glasses: i.e., Hands Free

Product Possibilities

* Party Gadget, or Conference Gadget
* Really serves as demo
* However, truly demonstrates potential

Bigger Markets

* Production Assembly
* Military equipment repair
* Emergency Medical Care
* Ikea Furniture Assembly
* Christmas Bicycle Assembly

This Presentation

* Walk Through the anatomy of an AR app.
* Discuss deficiencies

Design

* Model View Controller
* Image Recognition
* Gesture Recognition

Core Technologies:

* OpenCV (Khrnoous Java API)
* OpenGL (Google API wrapper)
* OpenCL (no official java wrapper, JNI to C++ used).
* Android

Comments on Technologies

* OpenCV
  + Several language APIs
  + org.opencv
  + Class Mat everywhere
  + Auto generated Java API ??
  + Matrix storage Row-Column layout ??
* OpenGL
  + Seems like written for C++ (using operator overloading)
  + javax.microedition.khronos.opengles
  + All matrices represented by float[]
  + Matrix storage Column-Row layout.
  + Auto generated Java API?
* Java APIs for both reflect history
* Neither is very strong typed
* Some overlap between the two
* Substantial “impedance matching”
  + Float vs Double
  + Degrees vs radians
  + X-Y-Z Coordinate definitions
  + OpenGL and OpenCV not “super polished:” i.e., as Android Studio is to Eclipse.
* Opportunity for a strong-java unification layer, but is there yet sufficient wisdom?
  + Look for convergence platform to be a battleground of the Android AR vendors.
* Like Unity, software platforms that unifies and absorbs hardware differences among vendors.

Hardware

* Google Cardboard AR demonstrates how hardware agnostic AR is.
* My Fortunetelling is:
  + Smart Glasses are effectively auxiliary cameras and display to smartphone.
  + Expect to simple Smart Glasses with USB3 to smartphone.
  + Leverage users existing smartphone.
  + Hardware race to the bottom: features, quality, cost.
* Extra battery pack for eight hour day probably needed.

Timing

* Image processing technology (in development for past 40 years)
* Portable hand held platforms (from Smart Phone)
* Heavy GPU processing power (from Game Industry)
* Android (bring all together)

# Chapter

Basic Environment Setup and Hello World App

Nature

* Ecosystem defined for a point in time.

Packages

* Android ATD (Eclipse + Android SDK)
* Android NDK
* OpenCV Library
* Appcompat?
* OpenCV (obtain libOpenCL.so from device)

Tools

* ADB
* Eclipse load,run,logcat,debug,ddms

App

* Basic Activity

Soapbox

* Eclipse vs. Intellij
* Complexity

# Chapter

Menu, Parameter, and Diagnostic Modes infrastructure

* Preparation for all to come
* Files: Constants, Activity, and Menu.

# Basis Image Processor

Camera, Display and Frame

* Frame (Camera) Listener
* Return passed in frame arg.
* Represents “segment” approach to IP

## Grey Scale

* What about all that color?

## Gaussian Filtering

* Sigma size?

## Canny Edge Detection

* Threshold coefficients

## Dilation

## Contour Detection

## Polygon Detection

# Parallelogram Detection

* Term Rhombus (singular) Rhombi (plural) used in code.
* Steps:
  + Filter out holes (or keep holes)
  + Filter according to minimum size.
  + Filter xxxxxx?
  + Determine angles
  + Filter w.r.t. mean-set
* Typically, not all nine tiles recognized

# Rubik Face Detection

* What we see verse what the camera sees.
* Objective is 2D 3x3 skewed grid.
* Obtain skew angles from Rhombus set
* Obtain center of each Rhombi
* Normalize to rectangular grid
* 2D Sort
* Least Means Square test and acceptance
* Tile color recording

# CPU Time Consumption

# Projections

* AR will be a heavy CPU and GPU consuming field
  + Algorithms are extensive
  + Lots of search and AI
  + Always on