3D Graphics and Cameras

Kari Pulli
Senior Director
NVIDIA Research
Overview

- OpenGL ES 1.x
- OpenGL ES 2.0
- WebGL
- OpenCV
- FCam

All examples on this session on Android
OpenGL: Project vertices to camera...
... connect them to triangles ...
... shade them based on lights ...
... add texture maps, shadows, ...

NVIDIA Research
What is OpenGL ES?

- OpenGL was too big for Embedded Systems
  - memory footprint, floating point HW

- OpenGL ES: a more compact API
  - mostly a subset of OpenGL
  - that can still do almost all OpenGL can
  - standardized at Khronos

- Nokia prototyped such a subset in 2002
OpenGL ES 1.0

- Simplify OpenGL 1.3
  - remove old complex functionality
  - `glBegin` – `glEnd` (OUT); vertex arrays (IN)
  - keep almost all the functionality
    - lights, projections, 2D textures, ...

Shipped in 2004 (mostly SW engines)

```c
static const GLbyte verts[4 * 3] =
{ -1, 1, 1, 1, 1, 1,
  1, -1, 1, -1, -1, 1 };  
static const GLubyte colors[4 * 3] =
{ 255, 0, 0, 255, 0, 0,
  0,255, 0, 0,255, 0,0 };  
glVertexPointer( 3,GL_BYTE,0, verts );
glColorPointerf( 3,GL_UNSIGNED_BYTE,
  0, colors );
glDrawArrays( GL_TRIANGLE_STRIP,
  0, 4 );
```
OpenGL ES 1.1

- Add features such as bump maps
- and skinning (deformable animation)

- Shipped in 2005
  - begin of HW accelerated mobile 3D graphics
  - Symbian, iOS, Android, Meego, …
The main Android SDK is based on Java
- the language is very similar to Sun’s original Java
  - UI libraries are different from Java SE
  - a different VM (called Dalvik)

http://developer.android.com/
- one-stop-shop for Android tools and documentation

The easiest way to install all the needed tools is
- JDK, SDK, NDK, Eclipse, …
- plus Tegra tools you can ignore if you don’t have a Tegra
Many Android versions (1 June 2011)

<table>
<thead>
<tr>
<th>Android Version</th>
<th>Market Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>Honeycomb (3.0, 3.1)</td>
<td>0.6%</td>
</tr>
<tr>
<td>Gingerbread (2.3 - 2.3.3)</td>
<td>9.2%</td>
</tr>
<tr>
<td>Froyo (2.2)</td>
<td>64.6%</td>
</tr>
<tr>
<td>Eclair (2.0/2.1)</td>
<td>21.2%</td>
</tr>
<tr>
<td>Donut (1.6)</td>
<td>2.5%</td>
</tr>
<tr>
<td>Cupcake (1.5)</td>
<td>1.9%</td>
</tr>
</tbody>
</table>

^ "Platform Versions". Android Developers. http://developer.android.com/resources/dashboard/platform-versions.html. "based on the number of Android devices that have accessed Android Market within a 14-day period ending on the data collection date noted below"
Quickly evolving platform (5 July 2011)

<table>
<thead>
<tr>
<th>Android Version</th>
<th>Market Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>Honeycomb (3.0, 3.1)</td>
<td>0.9%</td>
</tr>
<tr>
<td>Gingerbread (2.3 - 2.3.4)</td>
<td>18.6%</td>
</tr>
<tr>
<td>Froyo (2.2)</td>
<td>59.4%</td>
</tr>
<tr>
<td>Eclair (2.0/2.1)</td>
<td>17.5%</td>
</tr>
<tr>
<td>Donut (1.6)</td>
<td>2.2%</td>
</tr>
<tr>
<td>Cupcake (1.5)</td>
<td>1.4%</td>
</tr>
</tbody>
</table>

![Platform Versions](http://developer.android.com/resources/dashboard/platform-versions.html). "Based on the number of Android devices that have accessed Android Market within a 14-day period ending on the data collection date noted below"
Hello OpenGL ES 1.0 on Android SDK

The following OpenGL ES tutorial is available at

Create a project (on Eclipse)

New Android Project

Creates a new Android Project resource.

Project name: HelloOpenGLES10

Contents

Build Target

- Target Name: HelloOpenGLES10Activity.java
- Vendor: Android Open Source Project
- Platform: API 1.6

Properties

- Application name: HelloOpenGLES10
- Package name: com.example.gles10
- Create Activity: HelloOpenGLES10Activity
- Min SDK Version: 4
Default *.java file

```java
package com.example.ogles10;

import android.app.Activity;
import android.os.Bundle;

public class HelloOpenGLES10Activity extends Activity {

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```
A couple of new headers

Create a GLSurfaceView and set it as the main content view

Handle Android framework application state changes

Create a renderer for your surface view (in another *.java file we need to create)
Create the file / class

```java
package com.example.ogles10;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;

import android.opengl.GLSurfaceView;

public class HelloOpenGLES10Renderer implements GLSurfaceView.Renderer {

    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        // Set the background frame color
        gl.glClearColor(0.5f, 0.5f, 0.5f, 1.0f);
    }

    public void onDrawFrame(GL10 gl) {
        // Redraw background color
        gl.glClear((GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
    }

    public void onSurfaceChanged(GL10 gl, int width, int height) {
        gl.glViewport(0, 0, width, height);
    }
}
```

- Called once, to set things up
- Called for each redraw event
- Called whenever surface changes (e.g., rotate device from portrait to landscape)
Now we can run the app in emulator!
Let's draw a triangle: initialization

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;

private FloatBuffer triangleVB;

public void onSurfaceCreated(GL10 gl, EGLConfig config) {
    // Set the background frame color
    gl.glClearColor(0.5f, 0.5f, 0.5f, 1.0f);

    // Define a triangle, by default screen center is [0,0,0],
    // and extends from [-1,-1,0] (lower left) to [1,1,0] (upper right)
    float triangleCoords[] = {
        // X, Y, Z
        -0.5f, -0.25f, 0,
        0.5f, -0.25f, 0,
        0.0f, 0.559016994f, 0
    };

    // initialize vertex Buffer for triangle (# of coordinate values * 4 bytes per float)
    ByteBuffer vbb = ByteBuffer.allocateDirect(triangleCoords.length * 4);
    vbb.order(ByteOrder.nativeOrder()); // use the device hardware's native byte order
    triangleVB = vbb.asFloatBuffer(); // create a floating point buffer from the ByteBuffer
    triangleVB.put(triangleCoords); // add the coordinates to the FloatBuffer
    triangleVB.position(0); // set the buffer to read the first coordinate

    // Enable use of vertex arrays
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
}
(Re)Draw

```java
public void onDrawFrame(GL10 gl) {
    // Redraw background color
    gl.glClearColor(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
    // Draw the triangle
    gl.glColor4f(0.63671875f, 0.76953125f, 0.22265625f, 0.0f);
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, triangleVB);
    gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);
}
```

// Define a triangle, by default screen center is [0,0,0], and extends from [-1,-1,0] (lower left) to [1,1,0] (upper right)

```java
float triangleCoords[] = {
    // X, Y, Z
    -0.5f, -0.25f, 0,
    0.5f, -0.25f, 0,
    0.0f, 0.559016994f, 0
};
```
Projection matrix for camera type

```java
public void onSurfaceChanged(GL10 gl, int width, int height) {
    gl.glViewport(0, 0, width, height);
    // make adjustments for screen ratio
    float ratio = (float) width / height;
    gl.glMatrixMode(GL10.GL_PROJECTION);
    gl.glLoadIdentity(); // set matrix to projection mode
    gl.glFrustumf(-ratio, ratio, -1, 1, 3, 7); // apply the projection matrix
}
```
Modelview matrix for placing things

```java
public void onDrawFrame(GL10 gl) {
    // Redraw background color
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

    // Set GL_MODELVIEW transformation mode
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    gl.glLoadIdentity(); // reset the matrix to its default state

    // When using GL_MODELVIEW, you must set the view point
    GLU.gluLookAt(gl, 0, 0, -5, 0F, 0F, 0F, 0F, 1.0F, 0.0F);
}
```
No more stretching and squeezing based on orientation change
On OpenGL transformations

Instances and transformations

- An instance of a house is transformed by an instance transformation
  
  ```
  glMatrixMode( GL_MODELVIEW )
  glLoadIdentity()
  glTranslatef( ... )
  glRotatef( ... )
  glScalef( ... )
  house()
  ```

Local, changing coordinate system

- Another way to view transformations is as affecting a **local coordinate system** that the primitive is drawn in
- Now the transforms appear in the “right” order

Global, fixed coordinate system

- OpenGL’s transforms, logical as they may be, still seem backwards
  
  ```
  glLoadIdentity()
  glTranslatef( ... )
  glRotatef( ... )
  glScalef( ... )
  house()
  ```

They are, if you think of them as transforming the object in a **fixed coordinate system**
OpenGL ES 1.x Fixed Function pipe

API

Primitive Processing

Vertices

Transform and Lighting

Primitive Assembly

Rasterizer

Triangles/Lines/Points

Texture Environment

Color Sum

Fog

Vertex Buffer Objects

Alpha Test

DepthStencil

Color Buffer Blend

Dither

Frame Buffer
Fixed-function graphics in games
OpenGL ES 2.0 Programmable pipe (introduced in 2007)
Shaders in games
The difference?

- **Fixed-function graphics**
  - simple illumination models
  - evaluate at vertices, interpolate in between

- **Shaders**
  - vertex shader
    - smooth deformations and morphing
  - fragment shader
    - per-pixel illumination effects
    - arbitrary illumination models
Not backwards compatible

- ES 2.0 is a very bare-bones API

**Setup**

- Input of per-object constants (uniforms)
  - no matrix calculation support in the API
    - do it on CPU with other utility APIs

- Input of per-vertex data (attributes)
  - no special property types
    - normals, texture coordinates, …
  - it’s up to the shaders to interpret what the numbers mean

- And the shaders of course
  - sources as strings, compiled and linked on the fly
  - connect CPU variables with shader variables
Programmer’s model

- **Vertex Uniforms** (128 * vec4)
- **Attributes** (8 * vec4)
- **Vertex Shader**
- **Primitive Assembly & Rasterize**
- **Varyings** (8 * vec4)
- **Fragment Shader**
- **Fragment Uniforms** (16 * vec4)
- **Textures**
- **Per-Sample Operations**
The Vertex Shader can do:

- Modify any of the attributes of a vertex

Examples (old stuff, ES 1.1 could do):
  - Transformation of position and normals
    - using model-view and projection matrices
  - Texture coordinate generation and transformation
  - Per-vertex lighting

Examples (new stuff):
  - Arbitrary vertex morphing and skinning
  - Advanced lighting
    - per-vertex more complex lighting models
    - calculation of values used later for per-pixel lighting
  - Other per vertex data (density for physics, ...)

NVIDIA Research
The Vertex Shader cannot do:

- Anything that
  - requires information from more than one vertex
  - for example
    - any triangle operations (e.g., clipping, culling)
    - tessellation of meshes to denser meshes
  - accesses color buffer

That is: “One comes in, one goes out”
The Fragment Shader can do:

- Calculate new colors per pixel
  - texture blending
  - fog
- Dependent textures
- Pixel discard
  - alpha testing
- Bump and environment mapping
- Render to multiple render targets (MTR) at once
  - think of running the vertex part only once, but for the same geometry, producing different intermediate results to different textures at once
The Fragment Shader cannot do:

- Read from any buffer
  - including the previous color

- Write to any buffer
  - except provide the new color
  - which is blended by the later stages of fixed-function pipe

Why the limitations?

Now the implementation can
- rasterize different elements in parallel
- hide the latency of frame buffer access for blending
Android and Native Code

- Native code adds some complexity
  - but unlocks peak performance (games, image processing)
  - makes porting existing (C/C++) code easier
  - reduces portability

- A set of cross-compiler toolchains
  - gcc for ARM
  - hosted on Windows, Linux, or Mac

- JNI, the Java/Native Interface
  - connecting native code to Java

- Gingerbread (Android 2.3) introduced NativeActivity
  - avoids the need for any Java code
Android 2.2 (Froyo) introduced ES 2.0 on Java
but still not on emulators, only on device

We show a C example using NDK

The following example comes from
using its app_create.sh script
Tegra pack examples

- Import all sample projects to Eclipse workspace
  - set up environment variables, etc., following instructions on the pdf file
  - build all, try on a device
    here is es2_globe

- Create your own sample app
  
  ```
  karip@mac:~/NVPACK/TKD_Samples/Android_NVIDIA_samples_2_20110315/tools/app_create$
  ./app_create.sh my_app MyApp basic
  INFO: Java/native app being placed in apps
  creating destination directory tree ../../apps/my_app
  copying files to destination tree
  ```

- Build and try it
The project

Also need a library project for various utilities
package com.nvidia.devtech.MyApp;

import com.nvidia.devtech.*;
import android.view.MotionEvent;
import android.view.KeyEvent;

public class MyApp extends NvGLES2Activity {

    @Override
    public native boolean init();

    @Override
    public native boolean render(float drawTime, int drawWidth, int drawHeight, boolean forceRedraw);

    @Override
    public native void cleanup();

    @Override
    public native boolean inputEvent(int action, float x, float y, MotionEvent e);

    @Override
    public native boolean keyEvent(int action, int keyCode, KeyEvent e);

    static {
        System.loadLibrary("my_app");
    }
}
Matching part from C side

```c
jint JNI_OnLoad(JavaVM* vm, void* reserved)
{
    JNIEnv *env;

    NVThreadInit(vm);

    if (vm->GetEnv((void**) &env, JNI_VERSION_1_4) != JNI_OK) {
        DEBUG("Failed to get the environment using GetEnv()"));
        return -1;
    }

    JNINativeMethod methods[] = {
        { "init", "()Z", (void *) init },
        { "render", "(FIIZ)Z", (void *) render },
        { "inputEvent", "(Ljava/android/view/MotionEvent;)Z", (void *) inputEvent },
        { "keyEvent", "(Ljava/android/view/KeyEvent;)Z", (void *) keyEvent },
        { "cleanup", "()V", (void *) cleanup };
    }

    jclass k = (env)->FindClass("com/nvidia/devtech/MyApp/MyApp");
    (env)->RegisterNatives(k, methods, 5);

    NVTimeInit();

    return JNI_VERSION_1_4;
}
```

Match the C functions, with their types, to Java functions
Initialize shaders and attributes

```c
#define ROOT_3_OVER_2 0.8660254f
#define ROOT_3_OVER_6 (ROOT_3_OVER_2 / 3.0f)

static GLfloat s_vert[6] = { 0.5f, -ROOT_3_OVER_6,
                            -0.5f, -ROOT_3_OVER_6,
                            0.0f,  ROOT_3_OVER_2 - ROOT_3_OVER_6 }
;

static GLfloat s_col [12] = { 1.0, 0.0, 0.0, 1.0,
                             0.0, 1.0, 0.0, 1.0,
                             0.0, 0.0, 1.0, 1.0};

jboolean init(JNIEnv*  env, jobject thiz)
{
    nv_shader_init();
    DEBUG("GL_VENDOR: %s",getString(GL_VENDOR));
    DEBUG("GL_VERSION: %s",getString(GL_VERSION));
    DEBUG("GL_RENDERER: %s",getString(GL_RENDERER));

    s_prog = nv_load_program("demo");
    s_angle = 0;
    s_translationX = 0;
    s_translationY = 0;

    return JNI_TRUE;
}
```

- Vertex position and color data
- NV utilities help in getting started
Really simple shaders

Per object consts

Per vertex data

Outputs from vertex to fragment shader
```c
jint render(JNIEnv* env, jobject thiz, jfloat javatime,
            jint drawWidth, jint drawHeight, jboolean forceRedraw) {
    s_winW = (float)drawWidth;
    s_winH = (float)drawHeight;
    s_aspect = s_winW / s_winH;

    if (s_paused)
        return 0;

    s_angle += s_delta;

    glClear(GL_COLOR_BUFFER_BIT);

    glUseProgram(s_prog);

    nv_set_attrib_by_name(s_prog, "pos_attr", 2, GL_FLOAT, 0, 0, s_vert);
    nv_set_attrib_by_name(s_prog, "col_attr", 4, GL_FLOAT, 0, 0, s_col);

    float rad = (float)(s_angle * 0.0174532924F);
    glUniform2f(glGetUniformLocation(s_prog, "rot"), (float)cos(rad), (float)sin(rad));
    glUniform2f(glGetUniformLocation(s_prog, "translation"), s_translationX, s_translationY);
    glUniform1f(glGetUniformLocation(s_prog, "aspect"), s_aspect);

    glDrawElements(GL_TRIANGLES, 3, GL_UNSIGNED_BYTE, s_ind);

    return 1; // return true to update screen
```
Cleanup, and touch input processing

```c
void cleanup(JNIEnv* env)
{
    DEBUG("cleanup!!!");

    if(s_prog)
    {
        glDeleteProgram(s_prog);
        s_prog = 0;
    }
}

jboolean inputEvent(JNIEnv* env, jobject thiz, jint action, jfloat mx, jfloat my)
{
    DEBUG("inputEvent!!!");
    s_translationX = mx * 2.0f / s_winW - 1.0f;
    s_translationY = (1.0f - my * 2.0f / s_winH) / s_aspect;

    return JNI_TRUE;
}
```
WebGL: OpenGL ES 2.0 in JavaScript

- The only truly cross-platform way to create applications is to use web technologies
- WebGL provides OpenGL ES 2.0 in a browser
  - Supported by many desktop browsers
    - Chrome, Firefox, Safari, Opera
  - Coming to also on smartphones
    - Firefox on Android runs WebGL
<html>  <head>
<title>Learning WebGL &mdash; lesson 1</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
</head>
<body onlook="webGLStart();">
<canvas id="lesson01-canvas" style="border: none;" width="500" height="500">
</canvas>
</body>  </html>
var gl;

function initGL(canvas) {
    try {
        gl = canvas.getContext("experimental-webgl");
        gl.w = canvas.width;
        gl.h = canvas.height;
    } catch (e) {} 
    if (!gl) {
        alert("Could not initialise WebGL, sorry :-(");
    }
}
Shaders: vertex, fragment

```html
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
    }
</script>

<script id="shader-fs" type="x-shader/x-fragment">
    #ifdef GL_ES
    precision highp float;
    #endif

    void main(void) {
        gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
    }
</script>
```
Compile a shader

```javascript
function getShader(gl, id) {
    var shaderScript = document.getElementById(id);
    if (!shaderScript) { return null; }

    var shader;
    if (shaderScript.type == "x-shader/x-fragment") {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    } else if (shaderScript.type == "x-shader/x-vertex") {
        shader = gl.createShader(gl.VERTEX_SHADER);
    } else { return null; }

    var range = document.createRange();
    range.selectNodeContents(shaderScript);
    gl.shaderSource(shader, range.toString());
    range.detach();
    gl.compileShader(shader);

    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
        alert(gl.getShaderInfoLog(shader));
        return null;
    }

    return shader;
}
```
var prg; // shader program

function initShaders() {
    prg = gl.createProgram();
    gl.attachShader(prg, getShader(gl, "shader-vs"));
    gl.attachShader(prg, getShader(gl, "shader-fs"));
    gl.linkProgram(prg);

    if (!gl.getProgramParameter(prg, gl.LINK_STATUS)) {
        alert("Could not initialise shaders");
    }
}

gl.useProgram(prg);

prg.vtxPos = gl.getAttribLocation(prg, "aVertexPosition");
gl.enableVertexAttribArray(prg.vtxPos);

prg.pMatrixUniform = gl.getUniformLocation(prg, "uPMatrix");
prg.mvMatrixUniform = gl.getUniformLocation(prg, "uMVMMatrix");
Initialize vertex position buffers

```javascript
var tri; // triangleVertexPositionBuffer
var sqr; // squareVertexPositionBuffer

function initBuffers() {
    tri = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, tri);
    var vertices = [ 0.0, 1.0, 0.0,
                     -1.0, -1.0, 0.0,
                     1.0, -1.0, 0.0 ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
    tri.itemSize = 3;  tri.numItems = 3;

    sqr = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, sqr);
    vertices = [ 1.0, 1.0, -1.0,
                 -1.0, 1.0, -1.0,
                 -1.0, -1.0, -1.0,
                 1.0, -1.0, -1.0 ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
    sqr.itemSize = 3;  sqr.numItems = 4;
}
```
Include a simple matrix utility library

- OpenGL ES 2.0 does not include any matrix utilities
  - idea is that matrix math happens on CPU
  - OpenGL ES 2.0 concentrates on things running on GPU
  - it’s easy to write such utility libraries

- Include such a library from a file in the same folder

```html
<script type="text/javascript"
    src="glMatrix-0.9.5.min.js">
</script>
```
Finally, draw the scene

```javascript
function drawScene() {
  gl.viewport(0, 0, gl.w, gl.h);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

  var pMatrix = mat4.create();
  mat4.perspective(45, gl.w / gl.h, 0.1, 100.0, pMatrix);
  gl.uniformMatrix4fv(prg.pMatrixUniform, false, pMatrix);

  var mvMatrix = mat4.create(); mat4.identity(mvMatrix);
  mat4.translate(mvMatrix, [-1.5, 0.0, -7.0]);
  gl.uniformMatrix4fv(prg.mvMatrixUniform, false, mvMatrix);

  gl.bindBuffer(gl.ARRAY_BUFFER, tri);
  gl.vertexAttribPointer(prg.vtxPos, tri.itemSize, gl.FLOAT, false, 0, 0);
  gl.drawArrays(gl.TRIANGLES, 0, tri.numItems);

  mat4.translate(mvMatrix, [3.0, 0.0, 0.0]);
  gl.uniformMatrix4fv(prg.mvMatrixUniform, false, mvMatrix);
  gl.bindBuffer(gl.ARRAY_BUFFER, sqr);
  gl.vertexAttribPointer(prg.vtxPos, sqr.itemSize, gl.FLOAT, false, 0, 0);
  gl.drawArrays(gl.TRIANGLE_STRIP, 0, sqr.numItems);
}
```
Camera: the “I” of visual I/O

- Graphics is one half of the visual I/O
  - the output

- Camera provides the input
  - capture textures
  - affect the application

- New Tegra board supports cameras
  - Back: Stereo 5 MP
  - Front: 2 MP
  - SW: OpenCV & FCam
OpenCV
Thousands of Developers, Cross Platform API

- Open standard for Computer Vision
- 12 years old, professionally developed
  - Over 3 Million Downloads!
- > 500 Algorithms

Common API for Server, Workstation, Desktop and now Mobile Platforms!
Computer Vision = New Applications

Augmented Reality

Gesture interfaces

Augmented Reality Ghost Hunter (Argh)

Wordlens

Google Goggles

NVIDIA Research
OpenCV functionality overview

General Image Processing
Segmentation
Machine Learning, Detection
Image Pyramids
Transforms
Fitting

Camera Calibration
Features
Depth Maps
Optical Flow
Inpainting
Tracking
OpenCV for Android
Optimized for ARM, Tegra, & Android

Install from
Tutorial: Android Camera

- Part of the Android OpenCV distribution
- Get camera image
- Display it
Tutorial: Add OpenCV

- The second part of the tutorial adds OpenCV functionality
  - real-time Canny edge detector from the input image
OpenCV supports Java and Native
Basic Android camera is limited

- Android provides an easy-to-use camera API
  - most low-level functionality is automated and hidden

- Not sufficient if you need accurate camera control
  - computational photography
    - quickly taking images with different settings, combine to better pictures
  - many computer vision tasks
    - the system tries to optimize the image overall, without understanding the task
    - e.g., task: recognize a register plate of a car
      - the parameters should be set so that the plate looks good, other parts of the image can look arbitrarily bad
The **fCam** Architecture

- A software architecture for programmable cameras
  - that attempts to expose the maximum device capabilities
  - while remaining easy to program (C++)
  - modeling sensor as a pipeline allows rapid state changes
Applications

HDR
Low-light imaging

Creative use of flash

All-in-Focus Imaging
FCam: Open Source Project

What is it?

FCam is an open-source C++ API for easy and precise control of digital cameras. It allows full low-level control of all camera parameters on a per-frame basis, making it easy to rewrite the camera's autofocus routine, to capture a burst of images all with different parameters, and to synchronize the operation of the camera lens and flash with all of the above.

FCam is the result of the Camera 2.0 joint research project on programmable cameras and computational photography between Marc Levoy's group in the Stanford Computer Graphics Laboratory and Kari Pulli's team at Nokia Research Center Palo Alto. A paper describing the FCam architecture, the motivation behind it, and some applications, was presented at SIGGRAPH 2010.
FCam tutorial on Wed afternoon!
3D Graphics and Cameras

Kari Pulli
Senior Director
NVIDIA Research