2D and 3D Graphics in Freescale Devices
AMF-CON-T1025

Francisco Sandoval
Field Applications Engineer
Agenda

- Graphics in Freescale
- Introduction to graphics standards
- OpenGL ES vs OpenGL ES 2.0 vs OpenGL ES 3.0 vs OpenGL
- Freescale GPU_SDK
- Boundaries and relationship between layers and tools
- Benchmarks
Driver Information Systems Product Roadmap

**Best Graphics Performance and Integration**

**Optimized Graphics, High Integration**

**Cost Effective Performance**

**Lowest System Cost, High Integration**

---

**High-Line**
- **i.MX 6Dual**
  - Pin Compatible
  - i.MX 6Solo/DualLite

**Mid-Line**
- **SVF5XXR**
  - (Faraday/Vybrid)
- **MPC5645S**
  - (Rainbow)
- **SVF3xxR**

**Low-line**
- **MPC5606S**
  - (Spectrum)
- **S12ZVHY**
  - (Lumen-4WL)

**Basic**
- **S12XHY**
  - (Sea Wasp)
- **S12HY**
  - (Jellyfish)
- **S12ZVHY**
  - (Lumen-2W)
- **S08LG32**

---

**ARMv7 ISA**

- **Faraday+**
- **Halo**
- **Corona**

---

**Pin Compatible QFPs**

---

**Lowest System Cost, High Integration**

---

**Cost Effective Performance**

---

**Best Graphics Performance and Integration**

---

**Optimized Graphics, High Integration**

---

**Freescale**

---

**ARM Cortex-A9**
- ARM Cortex-Dual A7/M4, VSPA, 2D-ACE, GC355
- ARM Cortex-A5/M4, 2D-ACE, GC355
- ARM Cortex-A5/M4/M0+, 2D-ACE, 2D-GPU, GC355
- ARM Cortex-M4/M0+, 2D-ACE, 2D-GPU
Graphics cores

GC2000
GC880
GC355
GC255

Z430
Z160

GC320
2D-ACE
IPU*

VPU
VIU
etc

GCxxx cores are Vivante’s IP, newer
Zxxx cores are AMD’s, older
2D-ACE (DCUx), IPU are Freescale’s

Video input
encoding
decoding
*ipu CSI

Content creation
2D/3D Real time rendering

Composition
Memory manipulation
# Graphics cores

<table>
<thead>
<tr>
<th>Device</th>
<th>2D/Vector</th>
<th>3D</th>
<th>Composition</th>
</tr>
</thead>
<tbody>
<tr>
<td>GC2000</td>
<td>OpenGL ES 1.1</td>
<td>OpenGL ES 2.0</td>
<td>OpenGL ES 3.0</td>
</tr>
<tr>
<td>GC880</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z430</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GC355</td>
<td></td>
<td>OpenVG 1.1</td>
<td></td>
</tr>
<tr>
<td>Z160</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GC320</td>
<td>DirectFB 2D-ACE gfxlibs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2D-ACE/DCUx</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Freescale Devices

- i.MX6x – Most of them have 2D and 3D capabilities
  - DQ – GC2000, GC355, GC320
  - DL, S – GC880, GC320
  - SL – GC355
  - SLX - GC400T

- Spectrum – PowerPC core, has DCU
- Rainbow – PowerPC core, has DCU and z160
- Vybrid – ARM core, has GC355, 2D-ACE
Key concepts

- OpenGL – Open Graphics Library
- OpenGL ES – OpenGL for Embedded Systems
- EGL – Embedded-systems Graphics Library
- OpenCL – Open Computing Library
- OpenVG – Open Vector Graphics-library
- 2D-ACE – 2D Animation and Composition Engine (sexy for DCU)
- DCU – Display Controller Unit
- IPU – Image processing Unit
- Wayland – computer display server protocol, like X
- Weston – reference implementation of a wayland compositor
- Frame buffer – contiguous chunk of memory that stores a frame
## i.MX6 Linux graphics layers

<table>
<thead>
<tr>
<th>Layer</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPU</td>
<td>Frame buffer</td>
</tr>
<tr>
<td>EGL</td>
<td>Qt, DirectFB</td>
</tr>
<tr>
<td>OGLES/OVG</td>
<td>glClear();</td>
</tr>
<tr>
<td>Engine</td>
<td>Unity3D, EBGuide</td>
</tr>
<tr>
<td>Application</td>
<td>3D cluster</td>
</tr>
</tbody>
</table>
# Hybrid graphics layers

<table>
<thead>
<tr>
<th>Layer/</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCU</td>
<td>Configure DCU</td>
</tr>
<tr>
<td>Scheduler/RTOS</td>
<td>MQX</td>
</tr>
<tr>
<td>OVG/GFXLibs</td>
<td>vgClear(); Altia</td>
</tr>
<tr>
<td>Application</td>
<td>2D cluster</td>
</tr>
</tbody>
</table>
2.5D? 3D-like?

- 2D-ACE is very powerful, i.MX’s gpus are not always better:
  - Locked framerate
  - Hardware layers
  - On the fly HUD warping (coming soon)
  - Cheaper solution
  - Can look as good, sometimes even better frame rate. Example:
Differences between OpenGLs

• OpenGL – Desktop
• OpenGL ES 1.0, 1.1 (i.mx35, mpc5121, etc)
  - Fixed pipeline
  - Not compatible with 2.0
• OpenGL ES 2.0, 3.0 (i.mx5x, 6x)
  - Programmable pipeline (shader programs)
  - Compatible with 1.1 by a shader program
Fixed (1.0) Pipeline

Existing Fixed Function Pipeline

- API
  - Primitive Processing
    - Vertices
      - Transform and Lighting
      - Primitive Assembly
      - Rasterizer
    - Triangles/Lines/Points
      - Vertex Buffer Objects
      - Texture Environment
        - Colour Sum
          - Fog
            - Alpha Test
              - Depth Stencil
                - Colour Buffer Blend
                  - Dither
                    - Frame Buffer
The OpenGL ES 2.0 Pipeline
Shaders
Graphics process
3D objects are built from one or more series or “strips” of triangles.

- **GL_TRIANGLES**
- **GL_TRIANGLE_STRIP**
- **GL_TRIANGLE_FAN**
- **GL_QUADS**
- **GL_QUAD_STRIP**
How to place this in a 3D world?

• Everything is referenced using three coordinate systems:

  - Global space
  - Model space
  - Camera space

• The Matrix

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
x & y & z & 1
\end{bmatrix}
\]
The 3D world (Global space)

- The 3D Cartesian coordinates that define the triangle vertices are multiplied by a matrix to move it from the object’s space to the “screen” space.
Frustum
Transformations (Modelview Matrix)

Translation

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
x & y & z & 1
\end{bmatrix}
\]

Scale

\[
\begin{bmatrix}
x & 0 & 0 & 0 \\
0 & y & 0 & 0 \\
0 & 0 & z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Rotation (y)

\[
\begin{bmatrix}
\cos a & 0 & \sin a & 0 \\
0 & 1 & 0 & 0 \\
-\sin a & 0 & \cos a & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
3D models

- 3D models are made from a series of triangles and UV textures
Texture Mapping

• If a texture (2D still image) is being applied to the surface, the interpolated texture coordinates are used to derive what color should be sampled from the texture.
Texture Mapping

- To every vertex in our mesh, a pair of 2D coordinates (UV) is assigned and pixels are interpolated.

UV Textures go from (0.0,0.0) to (1.0, 1.0)
Text-tures

• Render string to texture

• PROS
  - Easiest method

• CONS
  - Least flexible method
  - Texture space in memory
  - Not scalable (resolution)
Text-tures

- Create a sprite-based alphabet

**PROS**
- More efficient
- Good for plain text

**CONS**
- Complexity ++
- Check alignment, spacing, etc
Text-tures

• Use library

• PROS
  – Most flexible, robust

• CONS
  – Most complex
  – Port engine to system
  – Third party SW
OpenGL Lighting

- Lighting is calculated at each vertex

- Phong Lighting 3 components:
  - Ambient.
    - Do not depend of the light source.
  - Diffuse.
    - The light is evenly reflected in all direction (think flashlight).
  - Specular.
    - The reflection is predominant in a given direction.
      - The specular coefficient can be varied.

- The light intensity can decrease according the distance (constant, linear or quadratic).
OpenGL Lighting

• By combining all the lighting components, we can create very realistic real-time renders

- Ambient Only
- Ambient + Diffuse
- Ambient + Diffuse + Specular
Shading Modes (ES 1.0)

- A normal vector defines the orientation of a surface relative to a light source
Shading Modes (ES 2.0)

Flat color, vertex position only

Color information

Moving the vertices around

Cartoony look

Per pixel lighting

Textures
Rasterization

- After the 3 vertex coordinates are transformed into screen space, they are then bilinearly interpolated and aliased to ‘fragments’
- Other data associated with the vertices is also interpolated (colors, normals, texture coordinates, etc.)
Z-test

- After interpolation, fragments have a ‘depth’ into the screen that is tested against the fragments that have already been rasterized.
  - If a fragment has already been produced that is “closer” to the screen, then the new value is not stored.
  - Otherwise, the new color value overwrites the old and the new depth is written to a “Depth Buffer” or “Z Buffer”.

Final image

Final z-buffer
GLSL

- GL Shading Language
- Supported in OpenGL ES 2.0

- Lets you define specific parts of the graphics pipeline
  - Vertex shader
  - Fragment shader

- C like language to write shaders

- Shaders have to be written, compiled and linked
GLSL overview

• Data types

  • float
  
  float a, b; // two vectors

  • bool
  
  int c = 2; // c is initialized with 2

  • Int
  
  bool d = true; // d is true

  • vec{2,3,4}

  • bvec{2,3,4}

  • ivec{2,3,4}

  • mat{2,3,4}

  • sampler{1,2,3}D

  • samplerCube

  • sampler{1D,2D}Shadow

  • arrays

  • structs

vec3 direction;
bvec2 lightFlags;
ivec3 color;
square matrices

for textures

for cube map textures

for shadow maps

struct dirlight { // type definition
vec3 direction; vec3 color; };
GLSL overview

- Variable qualifiers
  - **const**
    - The declaration is of a compile time constant
  - **attribute**
    - Global variables
    - May change per vertex
    - Passed from the application to vertex shaders.
    - Can only be used in vertex shaders.
    - Read-only variable.
  - **uniform**
    - Global variables
    - May change per primitive
    - Passed from the application to the shaders
    - Read-only variable.
  - **varying**
    - Used to pass data between vertex and fragment shader
    - write in vertex shader, and read-only in a fragment shader.
GLSL overview

• Functions

• At least a main per shader
• Functions can’t return arrays
• Function overloading is possible (different parameters)

```glsl
vec4 toonify(in float intensity)
{
    vec4 color;
    if (intensity > 0.98)
        color = vec4(0.8,0.8,0.8,1.0);
    else if (intensity > 0.5)
        color = vec4(0.4,0.4,0.8,1.0);
    else if (intensity > 0.25)
        color = vec4(0.2,0.2,0.4,1.0);
    else
        color = vec4(0.1,0.1,0.1,1.0);
    return(color);
}
```
Vertex Shader Functions

- Receives vertex data as input (position, colors, normals…)
  - The vertex shader can do:
    - Transformation of position using model-view and projection matrices
    - Transformation of normals, including renormalization
    - Texture coordinate generation and transformation
    - Per-vertex lighting
    - Color computation
  - The vertex shader cannot do:
    - Anything that requires information from more than one vertex
    - Anything that depends on connectivity.
    - Any triangle operations (e.g. clipping, culling)
    - Access color buffer
- Responsible for writing AT LEAST `gl_Position`
Sample Vertex Shader

uniform   mat4 g_matModelView;
uniform   mat4 g_matProj;
attribute vec4 g_vPosition;
attribute vec3 g_vColor;
varying   vec3 g_vVSColor;

void main()
{
    vec4 vPositionES = g_matModelView * g_vPosition;
    gl_Position   = g_matProj * vPositionES;
    g_vVSColor    = g_vColor;
}
Fragment Shader Functions

• Receives vertex shader output (varying variables) as input
  - The fragment shader can do:
    ▪ Texture blending
    ▪ Fog
    ▪ Alpha testing
    ▪ Dependent textures
    ▪ Pixel discarding
    ▪ Bump and environment mapping
  - The fragment shader cannot do:
    ▪ Blending with colour buffer
    ▪ ROP operations
    ▪ Depth or stencil tests
    ▪ Write depth
• Responsible of writing gl_FragColor (fragment color)
Sample Fragment Shader

precision highp float;

varying vec3 g_vVSCColor;

void main()
{
    gl_FragColor = vec4( g_vVSCColor, 1.0 );
}

Compile the shader program

```c
// Compile the shaders
GLuint hVertexShader = glCreateShader( GL_VERTEX_SHADER );
glShaderSource( hVertexShader, 1, &g_strVertexShader, NULL );
glCompileShader( hVertexShader );
// Check for compile success with glGetShaderiv

GLuint hFragmentShader = glCreateShader( GL_FRAGMENT_SHADER );
glShaderSource( hFragmentShader, 1, &g_strFragmentShader, NULL );
glCompileShader( hFragmentShader );
// Check for compile success with glGetShaderiv()

// Attach the individual shaders to the common shader program
g_hShaderProgram = glCreateProgram();
glAttachShader( g_hShaderProgram, hVertexShader );
glAttachShader( g_hShaderProgram, hFragmentShader );

// Init attributes BEFORE linking
glBindAttribLocation( g_hShaderProgram, g_hVertexLoc, "g_vPosition" );
gBindAttribLocation( g_hShaderProgram, g_hColorLoc, "g_vColor" );

// Link the vertex shader and fragment shader together
glLinkProgram( g_hShaderProgram );
// Check for link success with glGetProgramiv()

// Get uniform locations
g_hModelViewMatrixLoc = glGetUniformLocation( g_hShaderProgram, "g_matModelView" );
g_hProjMatrixLoc = glGetUniformLocation( g_hShaderProgram, "g_matProj" );
gDeleteShader( hVertexShader ); gDeleteShader( hFragmentShader );
```
How it looks in a sample application

- `glUseProgram()`
- Shader program handle
- MVM Loc handle
- PM Loc handle
- Color Loc handle
- Vertex Loc handle
- `gl_Position`
- MVM uniform
- PM uniform
- Color attribute
- Position attribute
- VSCColor varying
- `gl_FragColor`
- `glBindAttribLocation()`
- `glGetUniformLocation()`
void render(float w, float h) {
    // Rotate and translate the model view matrix
    float matModelView[16] = {0};
    fslLoadIdentity(matModelView);
    fslTranslate('z', -4, matModelView);
    // Build a perspective projection matrix
    float matProj[16] = {0};
    loadFrustum(matProj, 120, 1.33);
    // Clear the colorbuffer and depth-buffer
    glClearColor(0.0f, 0.0f, 0.5f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // Set the shader program
    glUseProgram(g_hShaderProgram);
    glUniformMatrix4fv(g_hModelViewMatrixLoc, 1, 0, matModelView);
    glUniformMatrix4fv(g_hProjMatrixLoc, 1, 0, matProj);
    // Bind the vertex attributes
    glVertexAttribPointer(g_hVertexLoc, 3, GL_FLOAT, 0, 0, VertexPositions);
    glEnableVertexAttribArray(g_hVertexLoc);
    glVertexAttribPointer(g_hColorLoc, 4, GL_FLOAT, 0, 0, VertexColors);
    glEnableVertexAttribArray(g_hColorLoc);
    // Draw
    glDrawArrays(GL_TRIANGLES, 0, 3);
    // Cleanup
    glDisableVertexAttribArray(g_hVertexLoc);
    glDisableVertexAttribArray(g_hColorLoc);
}
GPU_SDK

• i.MX6 Graphics SDK – Includes sample, demo code, and documentation for working with the i.MX6X family graphics cores. Includes OpenVG, OpenGL ES, and GAL2D reference files.

• Found in i.MX6 Software & Tools tab, under “Software Development Tools” -> “Snippets, Boot Code, Headers, Monitors, etc.”

• Released and maintained by the Virtual Graphics Core Team
GPU_SDK – contents
GPU_SDK – requirements

- A working LTIB with required libraries
- Export the following variables from console:
  - export `ROOTFS=/home/paco/ltib/121218/ltib/rootfs`
- Cd to directory of sample/demo/tutorial, type
  - `$make -f Makefile.fb`
- Copy generated binary and required resources (images, shader files, etc) to target then run.
Resources

- https://community.freescale.com/community/imx
- imxgpu mailing list (imxgpu@freescale.com)
Benchmark Comparisons
i.MX 6 Series

Kyle Fox
Product Manager – i.MX 6 and i.MX 7

Summary by Paco
## i.MX 6Quad/6Dual 1Ghz and 1.2Ghz Benchmarking Results

<table>
<thead>
<tr>
<th>Device</th>
<th>i. MX6Q Sbera SD VerC</th>
<th>i. MX6Q Sbera SD VerC</th>
<th>Media Tek Nexus 7</th>
<th>ASUS TF201</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS Version</td>
<td>R13.4.1-RC4</td>
<td>R13.4.1-RC4</td>
<td>Android ??</td>
<td>Android 4.1.1</td>
</tr>
<tr>
<td>CPU</td>
<td>i.MX6Quad 1G</td>
<td>i.MX6Quad 1.2G</td>
<td>1.2Ghz Quad A7</td>
<td>Tegra 3 1.3Ghz</td>
</tr>
<tr>
<td>GPU</td>
<td>Vivante GC2000</td>
<td>Vivante GC2000</td>
<td>Unknown</td>
<td>Tegra 3 GPU</td>
</tr>
<tr>
<td>Memory</td>
<td>1G DDR3</td>
<td>1G DDR3</td>
<td>Unknown</td>
<td>1GB LPDDR2</td>
</tr>
<tr>
<td>Screen Resolution</td>
<td>1024*728</td>
<td>1024*728</td>
<td>1280*800</td>
<td>1280*800</td>
</tr>
</tbody>
</table>

### Quadrant 2.0

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>CPU</th>
<th>Memory</th>
<th>I/O</th>
<th>2D</th>
<th>3D</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.MX6Q/6D</td>
<td>4028</td>
<td>9554</td>
<td>2796</td>
<td>4354</td>
<td>990</td>
<td>2445</td>
</tr>
<tr>
<td>Tegra3</td>
<td>4828</td>
<td>11069</td>
<td>2943</td>
<td>1259</td>
<td>990</td>
<td>2444</td>
</tr>
</tbody>
</table>

### AntTutu Benchmark 2.9.4

<table>
<thead>
<tr>
<th></th>
<th>Total Scores</th>
<th>Memory</th>
<th>CPU Integer</th>
<th>CPU Float</th>
<th>2D Graphics</th>
<th>3D Graphics</th>
<th>Database IO</th>
<th>SD Card Write</th>
<th>SD Card Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.MX6Q/6D</td>
<td>9605</td>
<td>9150</td>
<td>3057</td>
<td>2312</td>
<td>294</td>
<td>1248</td>
<td>400</td>
<td>150</td>
<td>194</td>
</tr>
<tr>
<td>Tegra3</td>
<td>11159</td>
<td>2348</td>
<td>3665</td>
<td>2782</td>
<td>294</td>
<td>1249</td>
<td>480</td>
<td>137</td>
<td>204</td>
</tr>
</tbody>
</table>

### AntTutu Benchmark 3.0

<table>
<thead>
<tr>
<th></th>
<th>Total Scores</th>
<th>Memory</th>
<th>CPU Integer</th>
<th>CPU Float</th>
<th>2D Graphics</th>
<th>3D Graphics</th>
<th>Database IO</th>
<th>SD Card Write</th>
<th>SD Card Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.MX6Q/6D</td>
<td>12603</td>
<td>2326</td>
<td>3747</td>
<td>2784</td>
<td>768</td>
<td>2191</td>
<td>460</td>
<td>123</td>
<td>204</td>
</tr>
<tr>
<td>Tegra3</td>
<td>NT</td>
<td>NT</td>
<td>NT</td>
<td>NT</td>
<td>NT</td>
<td>NT</td>
<td>NT</td>
<td>NT</td>
<td>NT</td>
</tr>
</tbody>
</table>

### BaseMark ES2.0 (Taiji)

<table>
<thead>
<tr>
<th></th>
<th>FPS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i.MX6Q/6D</td>
<td>36</td>
<td>NT</td>
</tr>
<tr>
<td>Tegra3</td>
<td>49.2</td>
<td>NT</td>
</tr>
</tbody>
</table>

### NenaMark2

<table>
<thead>
<tr>
<th></th>
<th>FPS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i.MX6Q/6D</td>
<td>17.93</td>
<td>14.65</td>
</tr>
<tr>
<td>Tegra3</td>
<td>46.5</td>
<td>46.4</td>
</tr>
</tbody>
</table>

### M3D

<table>
<thead>
<tr>
<th></th>
<th>FPS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i.MX6Q/6D</td>
<td>23.57921</td>
<td>Can’t run</td>
</tr>
<tr>
<td>Tegra3</td>
<td>26.32444</td>
<td>Can’t run</td>
</tr>
</tbody>
</table>

### Kernel Bootup Time

<table>
<thead>
<tr>
<th></th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.MX6Q/6D</td>
<td>20</td>
</tr>
</tbody>
</table>

---

- i.MX6Q/6D wins with Quadrant total score
- i.MX 6Q/6D wins with Antutu (normalized to screen size)
- i.MX 6Q wins with BaseMark ES2.0.
  On par with NenaMark due to LCD size
- i.MX6Q wins with M3D
## MX 6Quad/6Dual, 6DualLite, 6Solo vs Competition

<table>
<thead>
<tr>
<th>Device</th>
<th>SABRE SD</th>
<th>SABRE SD</th>
<th>SABRE SD</th>
<th>SABRE SD</th>
<th>Nexus 7</th>
<th>ASUS Eee PAD TF201</th>
<th>Galaxy Tab P7500</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OS Version</strong></td>
<td>R13.4.1-RC4</td>
<td>R13.4.1-RC4</td>
<td>R13.4.1-RC4</td>
<td>R13.4.1-RC4</td>
<td>Android 4.1.1</td>
<td>Android 4.0.3</td>
<td>Android 3.1</td>
</tr>
<tr>
<td><strong>CPU</strong></td>
<td>i.MX6Quad 1.2G</td>
<td>i.MX6 Quad 1G</td>
<td>i.MX6 DL 1G</td>
<td>i.MX 6S 1GHz</td>
<td>Tegra 3 1.3GHz</td>
<td>Tegra 3 1.2GHz</td>
<td>Tegra 2 1GHz</td>
</tr>
<tr>
<td><strong>GPU</strong></td>
<td>GC2000</td>
<td>GC320</td>
<td>GC2000</td>
<td>GC320</td>
<td>GC880</td>
<td>GC320</td>
<td>Tegra 3 GPU</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>64bit DDR3</td>
<td>64bit DDR3</td>
<td>64bit DDR3</td>
<td>32bit DDR3</td>
<td>LPDDR2</td>
<td>LPDDR2</td>
<td>DDR2</td>
</tr>
<tr>
<td><strong>Screen Resolution</strong></td>
<td>1024*728</td>
<td>1024*728</td>
<td>1024*728</td>
<td>1024*728</td>
<td>1280*800</td>
<td>1280*800</td>
<td>1280*800</td>
</tr>
<tr>
<td><strong>Quadrant 2.0</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>4011</td>
<td>4828</td>
<td>3005</td>
<td>2414</td>
<td>3741</td>
<td>3978</td>
<td>2559</td>
</tr>
<tr>
<td><strong>CPU</strong></td>
<td>9435</td>
<td>11407</td>
<td>4813</td>
<td>2411</td>
<td>11069</td>
<td>10414</td>
<td>3141</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>2809</td>
<td>3327</td>
<td>2807</td>
<td>2726</td>
<td>2943</td>
<td>2983</td>
<td>2584</td>
</tr>
<tr>
<td><strong>I/O</strong></td>
<td>4771</td>
<td>5693</td>
<td>4166</td>
<td>3973</td>
<td>1259</td>
<td>314</td>
<td>4988</td>
</tr>
<tr>
<td><strong>2D</strong></td>
<td>1000</td>
<td>1000</td>
<td>817</td>
<td>608</td>
<td>990</td>
<td>314</td>
<td>280</td>
</tr>
<tr>
<td><strong>3D</strong></td>
<td>2441</td>
<td>2445</td>
<td>2420</td>
<td>2352</td>
<td>2444</td>
<td>2464</td>
<td>1804</td>
</tr>
<tr>
<td><strong>AnTuTu 2.9.4</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total Scores</strong></td>
<td>9605</td>
<td>11159</td>
<td>5583</td>
<td>10707</td>
<td>10675</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>1950</td>
<td>2348</td>
<td>971</td>
<td>2051</td>
<td>2226</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CPU Integer</strong></td>
<td>3057</td>
<td>3665</td>
<td>1527</td>
<td>3594</td>
<td>3648</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CPU Float</strong></td>
<td>2312</td>
<td>2782</td>
<td>1148</td>
<td>2694</td>
<td>2729</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>2D Graphics</strong></td>
<td>294</td>
<td>294</td>
<td>294</td>
<td>298</td>
<td>293</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>3D Graphics</strong></td>
<td>1248</td>
<td>1249</td>
<td>927</td>
<td>1242</td>
<td>1198</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Database IO</strong></td>
<td>400</td>
<td>480</td>
<td>385</td>
<td>500</td>
<td>300</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SD Card Write</strong></td>
<td>150</td>
<td>137</td>
<td>127</td>
<td>150</td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SD Card Read</strong></td>
<td>194</td>
<td>204</td>
<td>204</td>
<td>178</td>
<td>192</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>AnTuTu 3.0.3</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total Scores</strong></td>
<td>11126</td>
<td>12603</td>
<td>4531</td>
<td>12972</td>
<td>12161</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>1992</td>
<td>2326</td>
<td>672</td>
<td>2033</td>
<td>2121</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CPU Integer</strong></td>
<td>3133</td>
<td>3747</td>
<td>775</td>
<td>3714</td>
<td>3621</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CPU Float</strong></td>
<td>2303</td>
<td>2784</td>
<td>564</td>
<td>2690</td>
<td>2503</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>2D Graphics</strong></td>
<td>765</td>
<td>768</td>
<td>474</td>
<td>797</td>
<td>730</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>3D Graphics</strong></td>
<td>2122</td>
<td>2191</td>
<td>1423</td>
<td>3003</td>
<td>2643</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Database IO</strong></td>
<td>430</td>
<td>460</td>
<td>350</td>
<td>395</td>
<td>310</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SD Card Write</strong></td>
<td>134</td>
<td>123</td>
<td>139</td>
<td>150</td>
<td>90</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SD Card Read</strong></td>
<td>157</td>
<td>204</td>
<td>134</td>
<td>190</td>
<td>143</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BaseMark ES20 (Taiji)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FPS</strong></td>
<td>25.65</td>
<td>25.63</td>
<td>7.78</td>
<td>7.67</td>
<td>17.93</td>
<td>14.65</td>
<td></td>
</tr>
<tr>
<td><strong>NenaMark2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FPS</strong></td>
<td>49.2</td>
<td>49.1</td>
<td>30.5</td>
<td>27.2</td>
<td>46.5</td>
<td>46.4</td>
<td></td>
</tr>
<tr>
<td><strong>M3D</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dynamic Lighting</strong></td>
<td>23.57921</td>
<td>23.54141</td>
<td>7.302674</td>
<td>7.254386</td>
<td>16.73816</td>
<td>5.979404</td>
<td></td>
</tr>
<tr>
<td><strong>Triangles Based</strong></td>
<td>40.87296</td>
<td>43.22133</td>
<td>19.66406</td>
<td>19.00992</td>
<td>23.14258</td>
<td>13.9976</td>
<td></td>
</tr>
<tr>
<td><strong>Physics Based</strong></td>
<td>53.72527</td>
<td>60.34217</td>
<td>43.28051</td>
<td>23.14033</td>
<td>35.68423</td>
<td>22.77416</td>
<td></td>
</tr>
</tbody>
</table>
**AnTuTu v2.9.3, v2.9.4 and v3.0 Comparison (6Quad) vs MediaTek**

<table>
<thead>
<tr>
<th>Device</th>
<th>i. MX6Q Sbera SD VerC</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS Version</td>
<td>R13.4.1-RC4</td>
</tr>
<tr>
<td>CPU</td>
<td>i.MX6Quad 1.2G</td>
</tr>
<tr>
<td>GPU</td>
<td>Vivante GC2000</td>
</tr>
<tr>
<td>Memory</td>
<td>1G DDR3</td>
</tr>
</tbody>
</table>

### AnTuTu Benchmark 2.9.4

<table>
<thead>
<tr>
<th>Sphere</th>
<th>Total Scores</th>
<th>Memory</th>
<th>CPU Integer</th>
<th>CPU Float</th>
<th>2D Graphics</th>
<th>3D Graphics</th>
<th>Database IO</th>
<th>SD Card Write</th>
<th>SD Card Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnTuTu Benchmark 2.9.4</td>
<td>11159</td>
<td>2348</td>
<td>3665</td>
<td>2782</td>
<td>294</td>
<td>1249</td>
<td>480</td>
<td>137</td>
<td>204</td>
</tr>
</tbody>
</table>

### AnTuTu Benchmark 3.0

<table>
<thead>
<tr>
<th>Sphere</th>
<th>Total Scores</th>
<th>Memory</th>
<th>CPU Integer</th>
<th>CPU Float</th>
<th>2D Graphics</th>
<th>3D Graphics</th>
<th>Database IO</th>
<th>SD Card Write</th>
<th>SD Card Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnTuTu Benchmark 3.0</td>
<td>12603</td>
<td>2326</td>
<td>3747</td>
<td>2784</td>
<td>768</td>
<td>2191</td>
<td>460</td>
<td>123</td>
<td>204</td>
</tr>
</tbody>
</table>

![MT6589 1.2G](image1.png) ![MSM8260A 1.5G](image2.png)
Normalizing Antutu results

- Antutu benchmark measures full range of capabilities
  - CPU integer/float, memory b/w, database i/o and storage card
  - 2D and 3D performance metrics are only part of the story
  - Goal is to ensure a more complete view of the performance of the CPU
- Freescale i.MX 6Quad/6Dual Antutu tested on 1024x768 display
  - Mediatek benchmarks are assumed to be 720p (1280x720)
  - Delta is around 14.6% greater pixels in Mediatek screen vs Freescale
  - This will affect the 2D and 3D portions of the Antutu benchmark
- To quickly normalize Freescale’s results to a 720p screen:
  - 1) all scores for i.MX 6Q/6D stay the same except for the 2D/3D scores.
  - 2) the i.MX 6Q/6D scores for 2D/3D are reduced by 14.6% to account for a 720p screen
    - 2D was 294. New score is 250.9
    - 3D was 1249. New score is 1065.81
- Therefore the Freescale i.MX 6Q/6D normalized score would be: 10887.71
  - existing score: 11,159; subtract original 2D/3D: 11,159 - 294 - 1294 = 9571
  - add in normalized 2D/3D #'s: 9571 + 250.9 + 1065.81 = 10887.71
### Benchmark Comparison – OMAP4470 and 4460

<table>
<thead>
<tr>
<th>Benchmark Suite</th>
<th>Item</th>
<th>i.MX6Quad 1Ghz 1024x768</th>
<th>i.MX6Dual 1Ghz 1024x768</th>
<th>Archos 101x Tablet TI OMAP 4470 1.5Ghz 1280x800 lcd</th>
<th>Archos 101 G9 Tablet OMAP 4460 1.2Ghz 1280x800 lcd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quadrant</td>
<td>Total</td>
<td>4636</td>
<td>3300</td>
<td>3557</td>
<td>2622</td>
</tr>
<tr>
<td>Antutu</td>
<td>Total</td>
<td>10488</td>
<td>6911</td>
<td>5874</td>
<td>7835</td>
</tr>
<tr>
<td>Linpack</td>
<td>Multithread</td>
<td>136.71</td>
<td>87</td>
<td>94.9</td>
<td>82.6</td>
</tr>
<tr>
<td>Velamo</td>
<td></td>
<td>1086</td>
<td>752</td>
<td>1374</td>
<td>1325</td>
</tr>
</tbody>
</table>
### Relevant Benchmarks

<table>
<thead>
<tr>
<th>UI Category</th>
<th>Benchmark</th>
<th>Unit</th>
<th>i.MX 6</th>
<th>Competition 1</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) UI’s require ‘quality’ pixels to enhance view</td>
<td>3DMarkMobile ES 1.1: Proxycon</td>
<td>FPS</td>
<td>108 FPS</td>
<td>60 (iPad 2)</td>
<td>Older gen benchmark for more basic games</td>
</tr>
<tr>
<td></td>
<td>3DMarkMobile ES 1.1: Samurai</td>
<td>FPS</td>
<td>103 FPS</td>
<td>60 (iPad 2)</td>
<td>Older gen benchmark for more basic games</td>
</tr>
<tr>
<td></td>
<td>3DMarkMobile ES 2.0: Taiji Girl</td>
<td>FPS</td>
<td>35 FPS</td>
<td>24 (iPad 2)</td>
<td>Updated for latest 3D – tests whether complex games can be run at high FPS</td>
</tr>
<tr>
<td></td>
<td>3DMarkMobile ES 2.0: Hover Jet</td>
<td>FPS</td>
<td>25 FPS</td>
<td>14 (iPad 2)</td>
<td>Updated for latest 3D – tests whether complex games can be run at high FPS</td>
</tr>
<tr>
<td></td>
<td>3) UI content is inherently dynamic → Cannot predict content so need High speed, wide memory bus</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### LMBENCH

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Unit</th>
<th>i.MX 6</th>
<th>OMAP 4</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mem Read (MB/s) (higher is better)</td>
<td>MB/s</td>
<td>264</td>
<td>163</td>
<td>Ability to read content fast during decode or 3D operations</td>
</tr>
<tr>
<td>Mem Write (MB/s) (higher is better)</td>
<td>MB/s</td>
<td>2280</td>
<td>2221</td>
<td>Ability to write data during decode or 3D operations</td>
</tr>
<tr>
<td>BCopy (libc) (MB/s) (higher is better)</td>
<td>MB/s</td>
<td>590</td>
<td>312.8</td>
<td>Ability to replicate frame buffers or other content especially during UI</td>
</tr>
<tr>
<td>L1 Latency (ns) (lower is better)</td>
<td>ns</td>
<td>3.93</td>
<td>3.974</td>
<td>How fast memory roundtrip is to CPU</td>
</tr>
<tr>
<td>L2 Latency (ns) (lower is better)</td>
<td>ns</td>
<td>27.2</td>
<td>29.6</td>
<td>How fast memory roundtrip is to CPU</td>
</tr>
<tr>
<td>Main Mem Latency (ns) (lower is better)</td>
<td>ns</td>
<td>113.8</td>
<td>194.1</td>
<td>How fast memory roundtrip is to CPU</td>
</tr>
<tr>
<td>Rand Mem Latency (ns) (lower is better)</td>
<td>ns</td>
<td>172</td>
<td>265</td>
<td>How fast memory roundtrip is to CPU</td>
</tr>
<tr>
<td>MemCopy</td>
<td>MB/s</td>
<td>8K copies: 6975</td>
<td>64K copies: 4542</td>
<td>Tegra4 1Ghz on PANDA board 8K copies: 6510</td>
</tr>
</tbody>
</table>

---

**i.MX 6 vastly outperforms Tegra3 and 2**
<table>
<thead>
<tr>
<th>UI Category</th>
<th>Benchmark</th>
<th>Unit</th>
<th>i.MX 6</th>
<th>Competition 1</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4) UI requires high resolution support</td>
<td>Run two 1080p h.264 high profile video clips on two different 1080p HDMI TVs</td>
<td>FPS</td>
<td>30 fps per screen decode</td>
<td>Not possible on Tegra 2. unlikely to be capable on Tegra 3 due to single memory channel</td>
<td>Measures ability of chip to drive higher resolution displays than just 1080p. Plus ability to do picture in picture video playback simultaneously</td>
</tr>
<tr>
<td></td>
<td>CPU Core Utilization during Playback of video content</td>
<td>% of max</td>
<td>4 %</td>
<td>Not measured</td>
<td>Determines if CPU provider is using main core to offload some or most of the decode. Also measures how much CPU is left over for other tasks</td>
</tr>
<tr>
<td>5) Access to fast CPU MIPS</td>
<td>Dhrystone</td>
<td>DMIPS</td>
<td>2.45 DMIPS/MHz</td>
<td>TI PANDA Board 2.2 DMIPS/MHz</td>
<td>Very basic benchmark that is useful to determine if core has been implemented correctly to ARM spec. Lower score means possible implementation problems or slow transistors.</td>
</tr>
<tr>
<td></td>
<td>CoreMark</td>
<td>Score</td>
<td>i.MX 6Quad processor at 1Ghz, 4 cores: 11147 coremark;</td>
<td>Nvidia Tegra 3 with 5 cores: 11352</td>
<td>Better score for CPU than Dhrystone. Much more complete set of tests to measure CPU performance</td>
</tr>
<tr>
<td></td>
<td>SunSpider (0.9.1) Java</td>
<td>Sec</td>
<td>run on i.MX 6Quad core 1: 1830; core 2: 1827; core 3: 1836; core 4: 1849</td>
<td>OMAP4 @1Ghz on Panada Board: 1756 with two cores active</td>
<td>Browser test with broad support. Also a good measure for the CPU’s ability to handle JAVA</td>
</tr>
<tr>
<td></td>
<td>Threading test</td>
<td>Tbd</td>
<td>Tbd</td>
<td>Tbd</td>
<td>Measure ability to hit multiple threads and allocate effectively across multiple CPUs</td>
</tr>
</tbody>
</table>

**Tegra 3 limited by single memory channel**