2D and 3D Graphics in Freescale Devices

AMF-CON-T1025

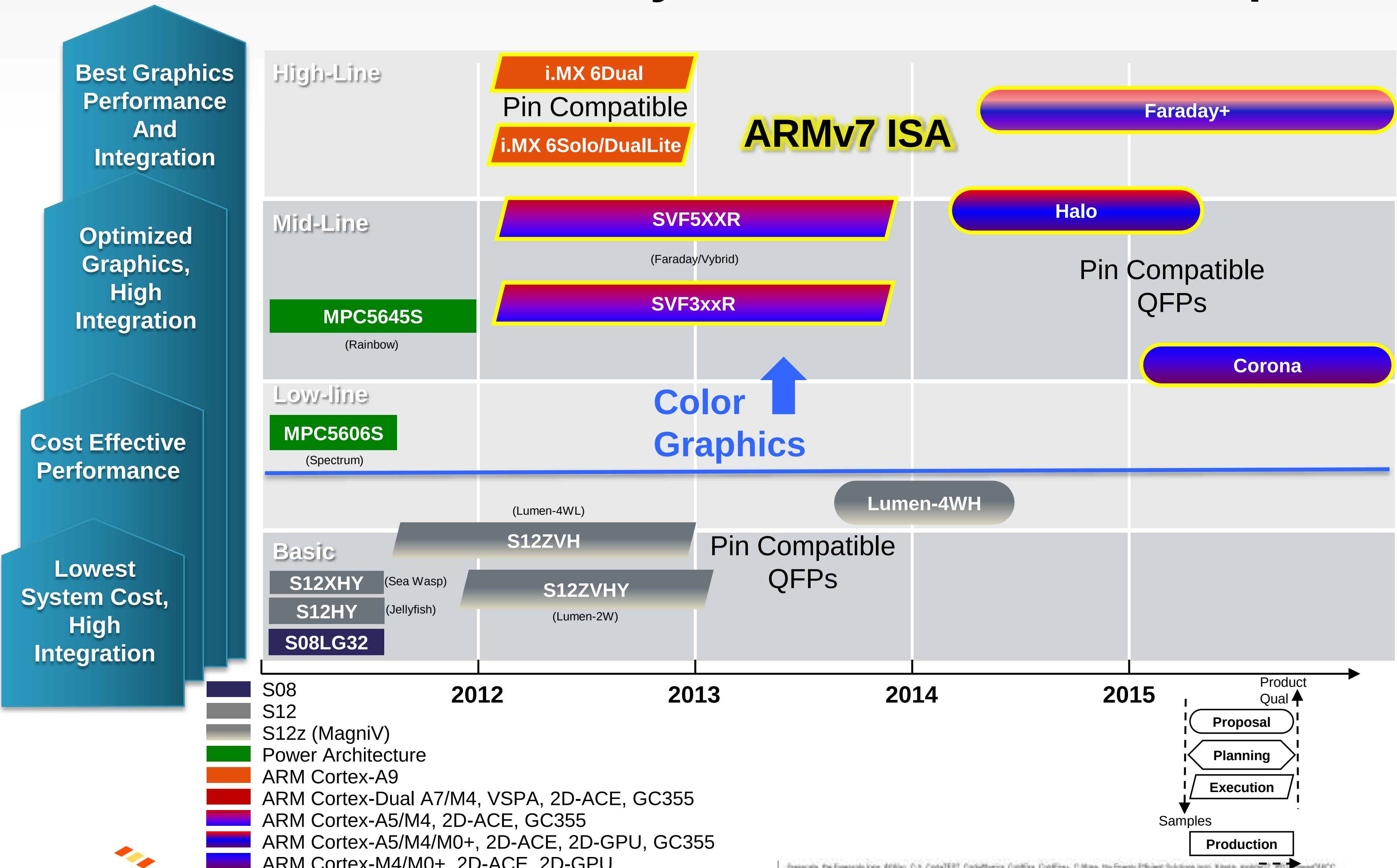
Francisco Sandoval
Field Applications Engineer

September 2013

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



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

GC2000
GC880
Z430

OpenGL ES 1.1
OpenGL ES 2.0
OpenGL ES 3.0

3D

GC355
Z160

OpenVG 1.1

2D/Vector

GC320
IPU
2D-ACE/DCUx

DirectFB
2D-ACE gfxlibs

Composition

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

freescale™

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

Layer/	Example
IPU	Frame buffer
EGL	Qt, DirectFB
OGLES/OVG	glClear();
Engine	Unity3D, EBGuide
Application	3D cluster

Vybrid graphics layers

Layer/	Example
DCU	Configure DCU
Scheduler/RTOS	MQX
OVG/GFXLibs	vgClear(); Altia
Application	2D cluster

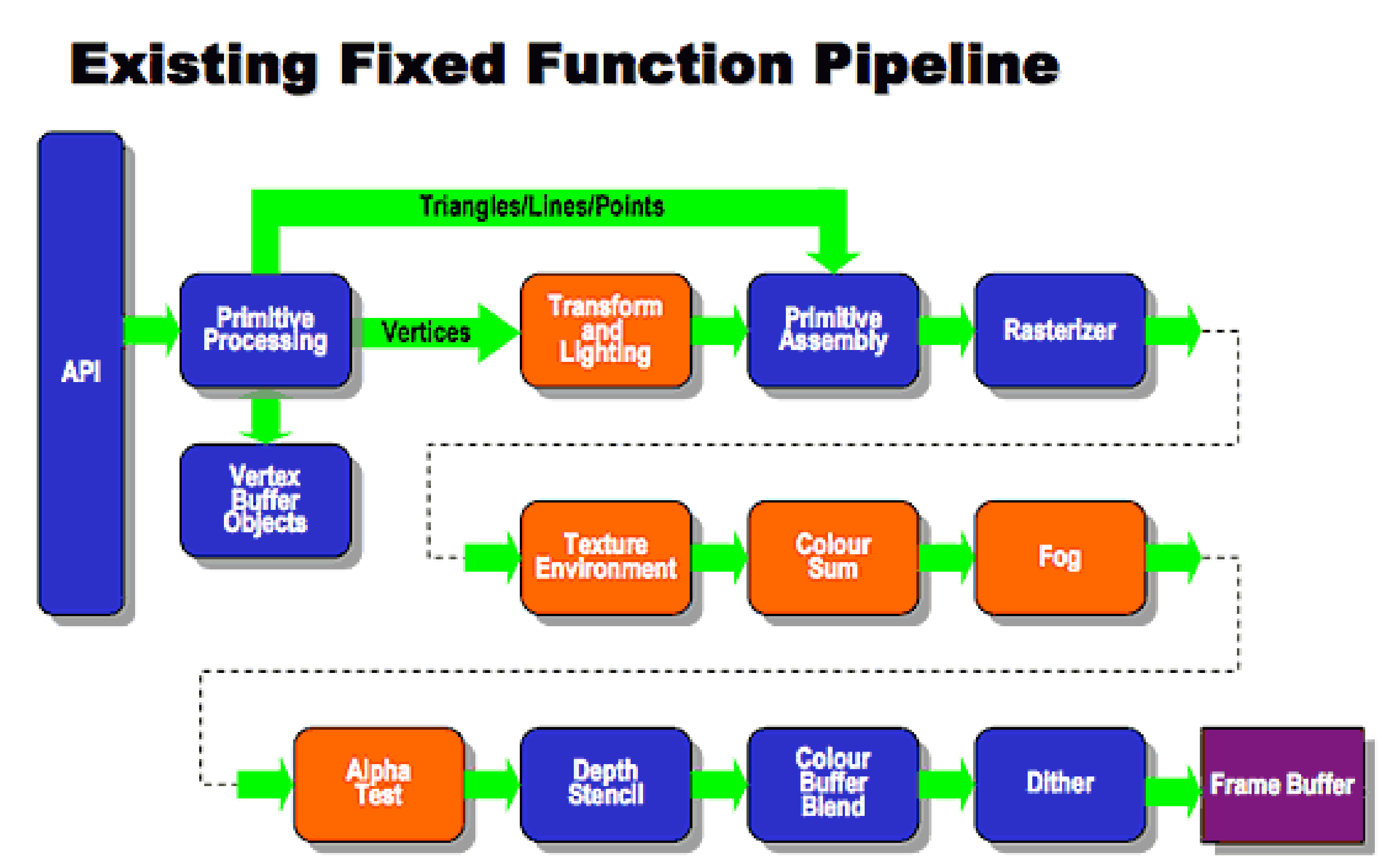
2.5D? 3D-like?

- 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:

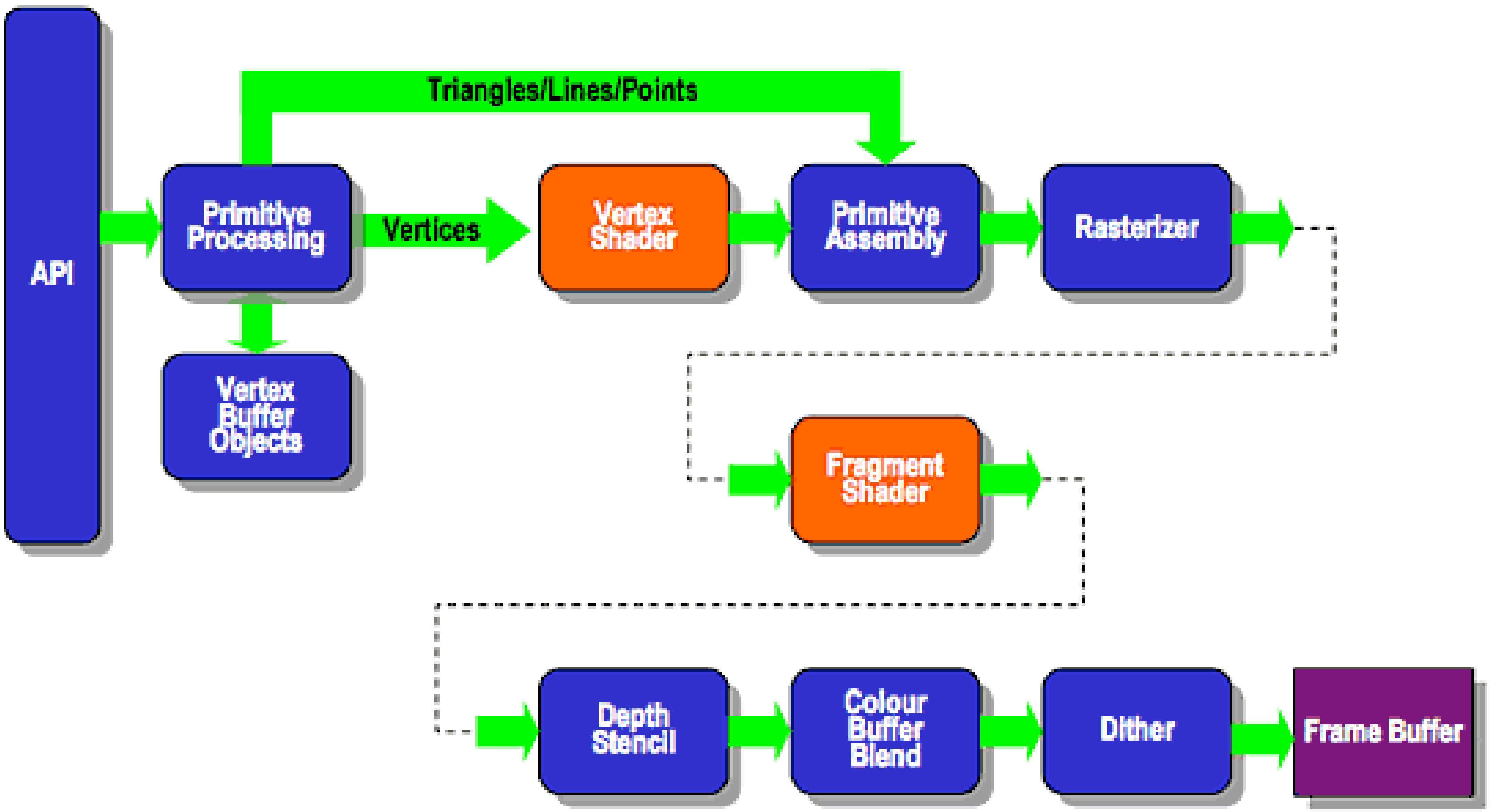
Differences between OpenGLs

- 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

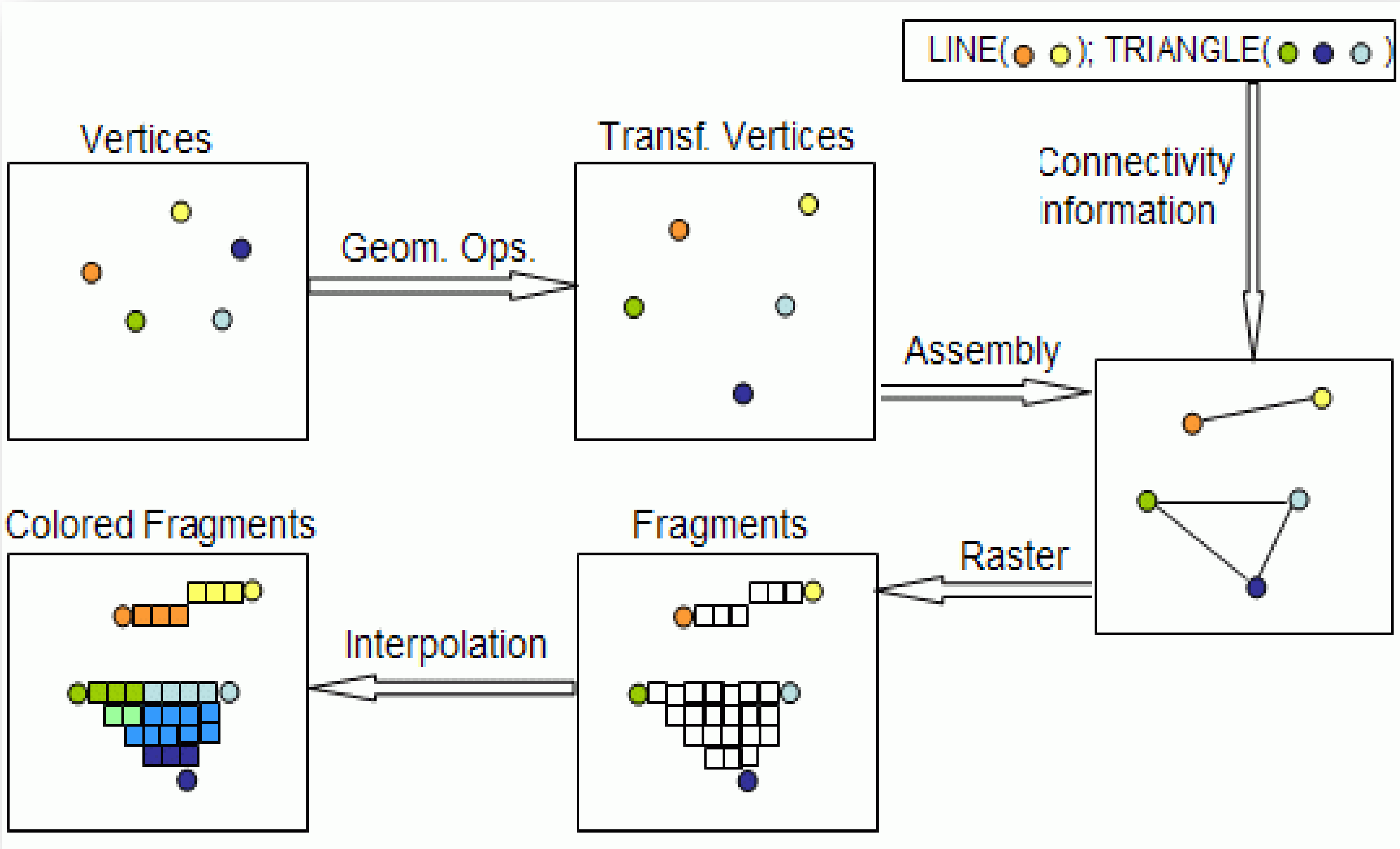


The OpenGL ES 2.0 Pipeline



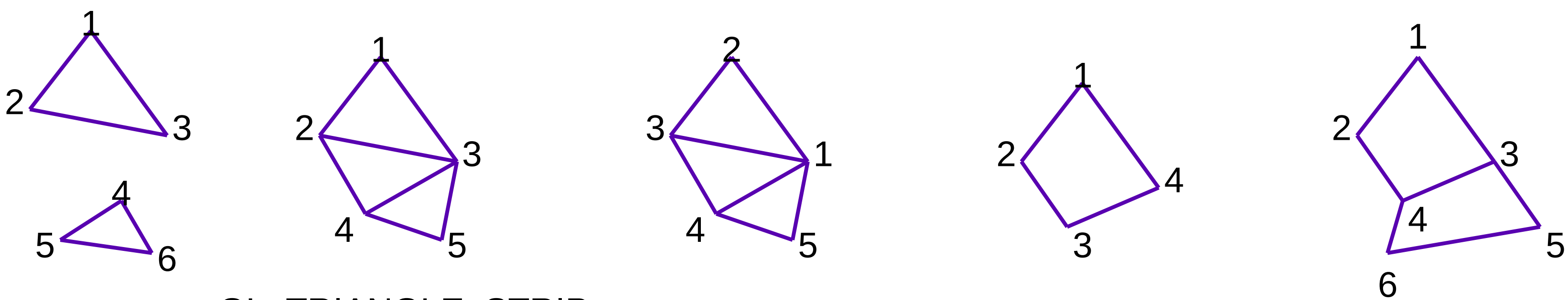
Shaders

Graphics process



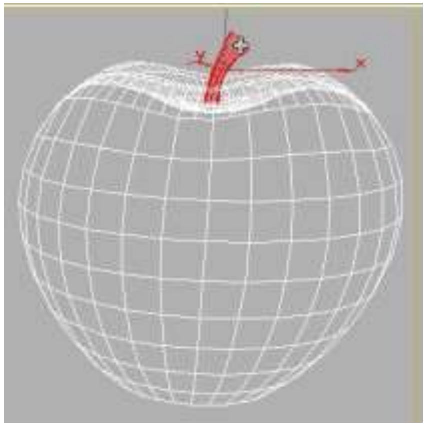
Triangles in OpenGL

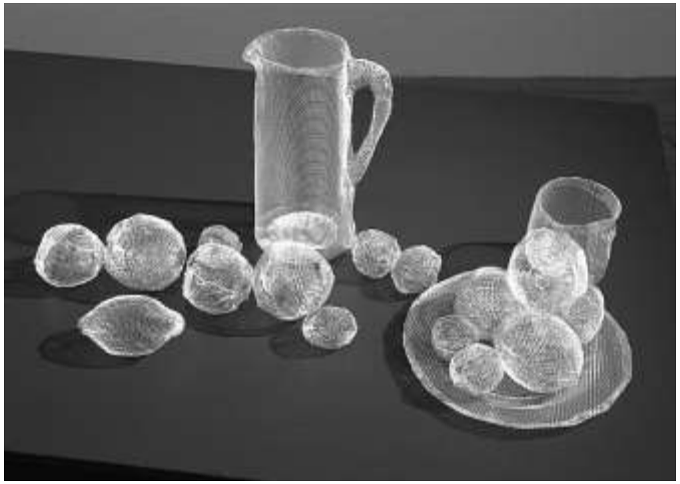
- 3D objects are built from one or more series or “strips” of triangles



GL_TRIANGLE_STRIP

GL_TRIANGLES GL_TRIANGLE_FAN GL_QUADS GL_QUAD_STRIP





How to place this in a 3D world?

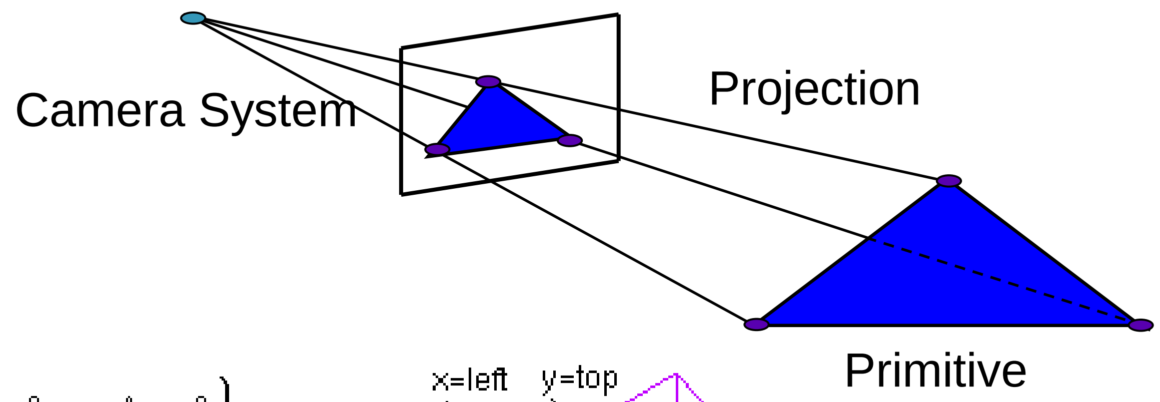
- Everything is referenced using three coordinate systems:
  - Global space
  - Model space
  - Camera space
- The Matrix

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

MATRICES!! MATH!!

The 3D world (Global space)

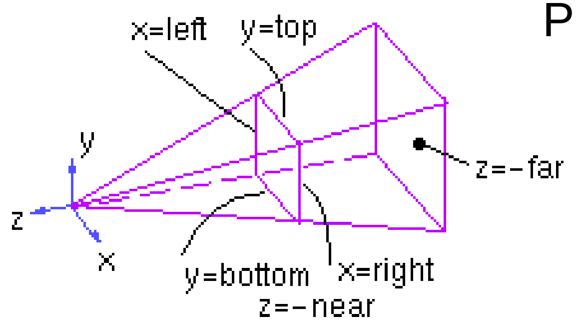
- The 3D Cartesian coordinates that define the triangle vertices are multiplied by a matrix to move it from the object's space to the "screen" space



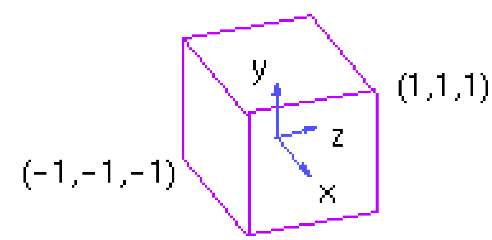
$$\begin{pmatrix} \frac{2\,near}{right-left} & 0 & A & 0 \\ 0 & \frac{2\,near}{top-bottom} & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$A = \frac{right+left}{right-left}$$

$$B = \frac{top+bottom}{top-bottom}$$

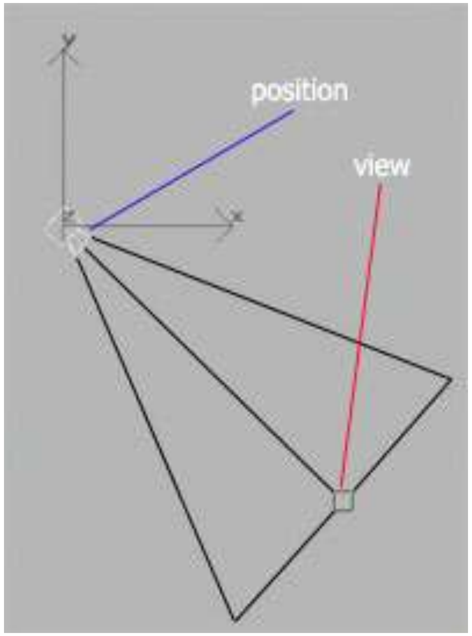


VCS

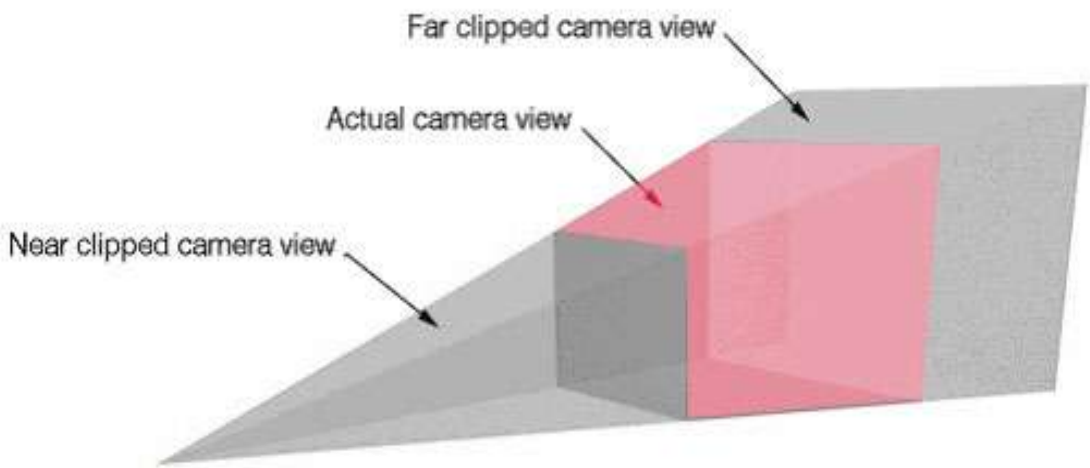


NDCS

Frustum

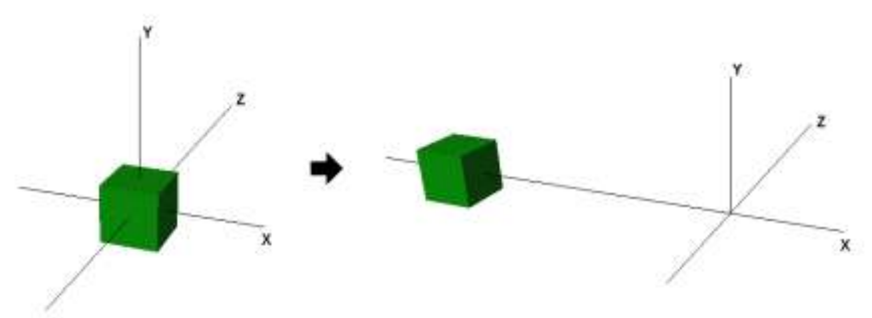


position
view



Far clipped camera view
Actual camera view
Near clipped camera view

Transformations (Modelview Matrix)

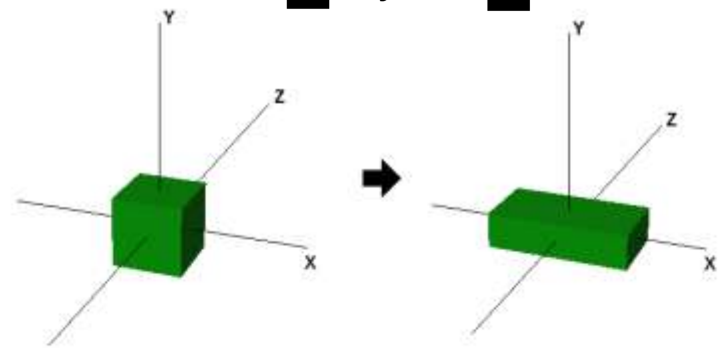


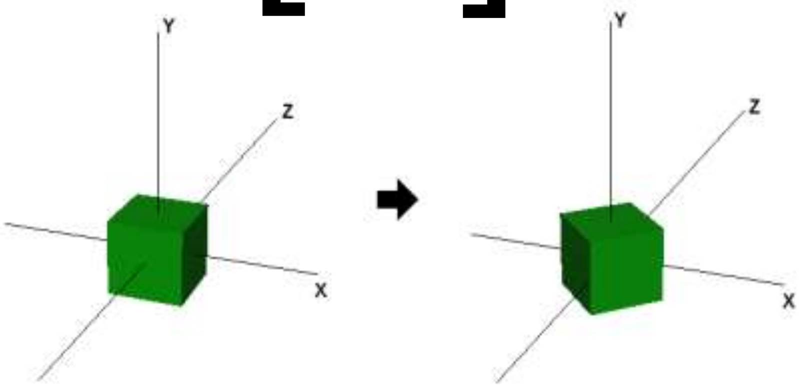
Translation

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

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

Scale



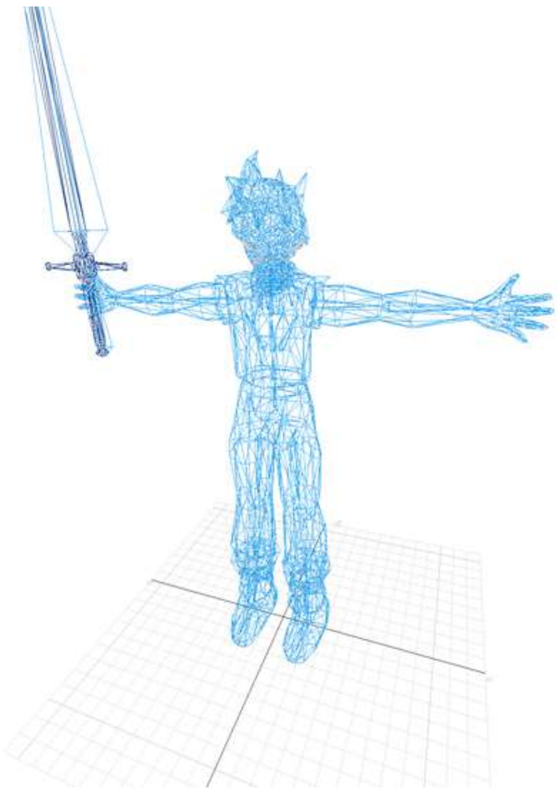


Rotation (y)

$$\begin{bmatrix} \cos a & 0 & \sin a & 0 \\ 0 & 1 & 0 & 0 \\ -\sin a & 0 & \cos a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

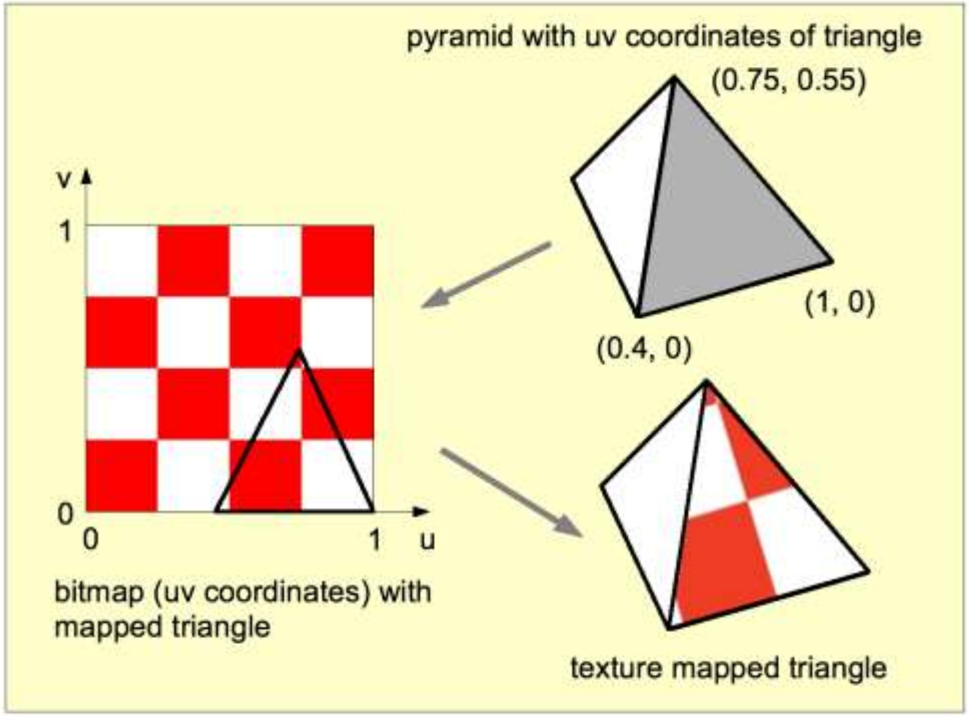
3D models

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



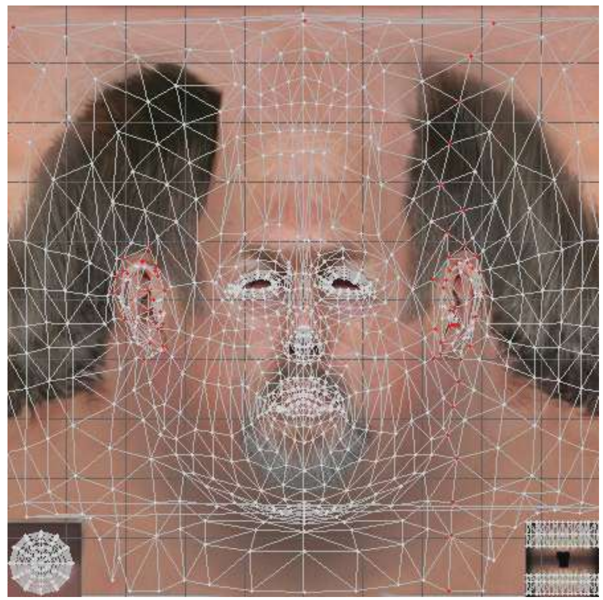
Texture Mapping

- 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



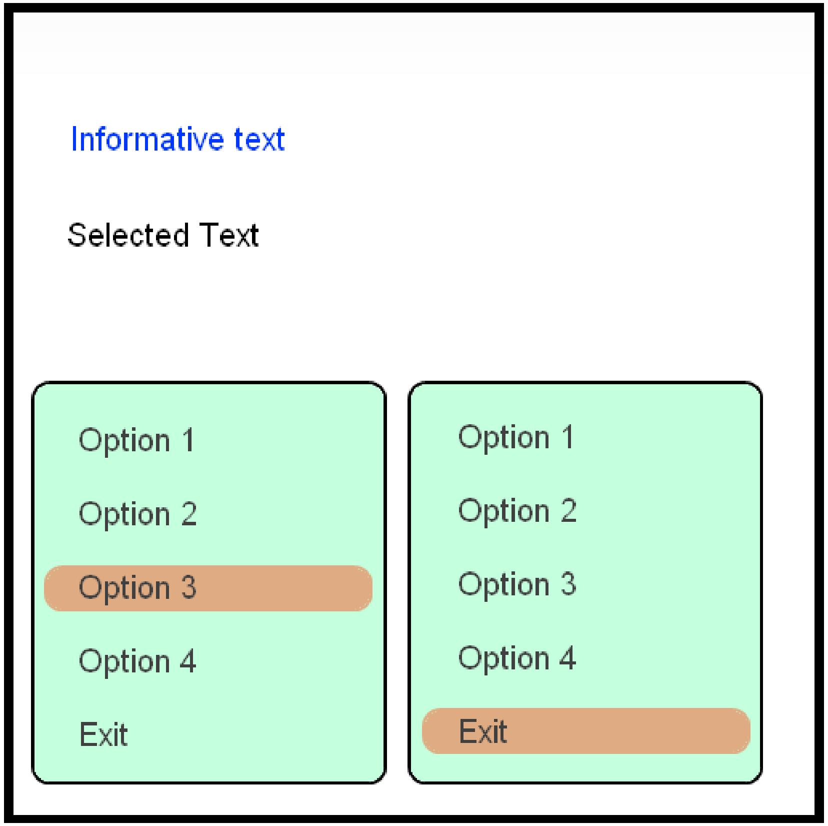
UV Textures go from (0.0,0.0) to (1.0, 1.0)

Text-tures

- Render string to texture

- PROS
  - Easiest method

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

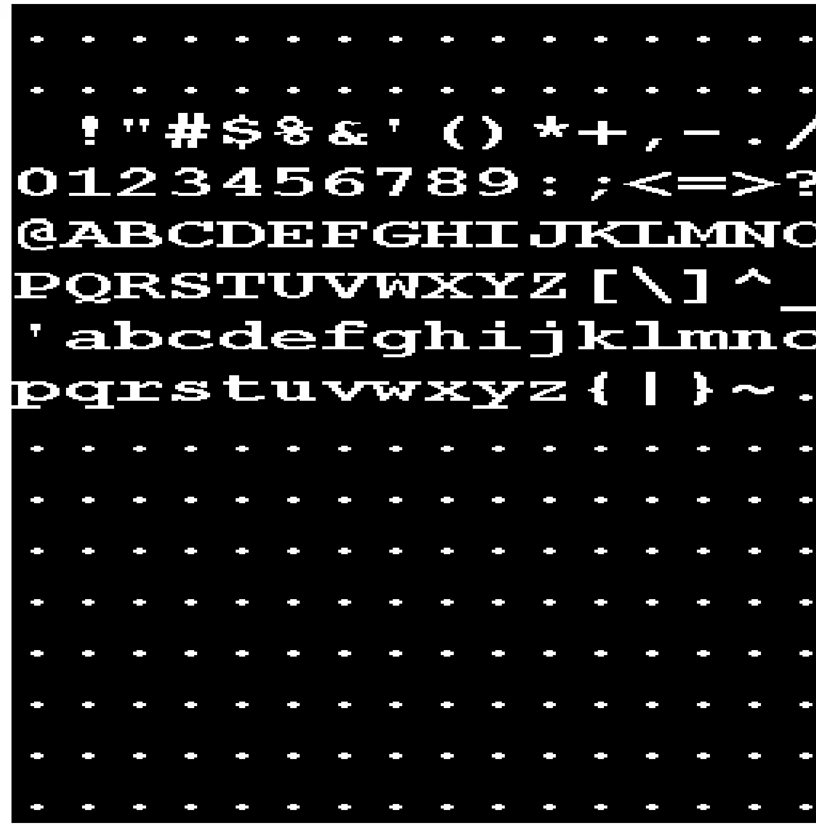


Text-tures

- Create a sprite-based alphabet

- PROS
  - More efficient
  - Good for plain text

- CONS
  - Complexity ++
  - Check alignment, spacing, etc



Text-tures

- Use library

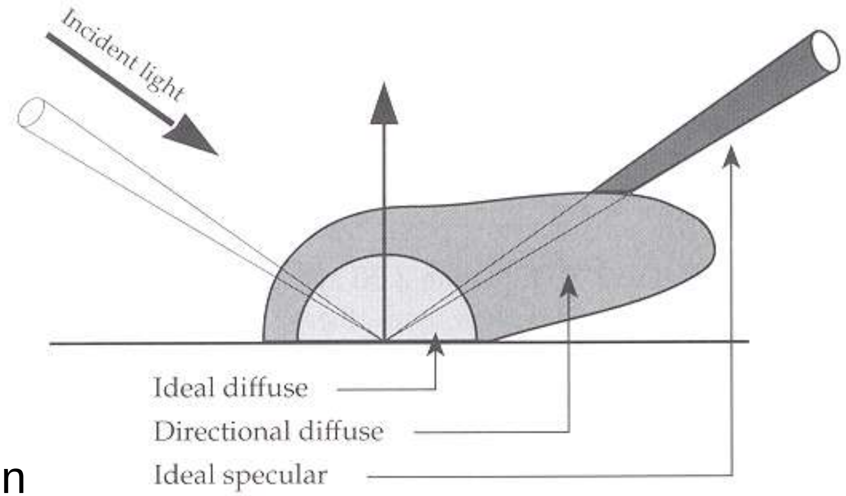
- PROS
  - Most flexible, robust

- 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

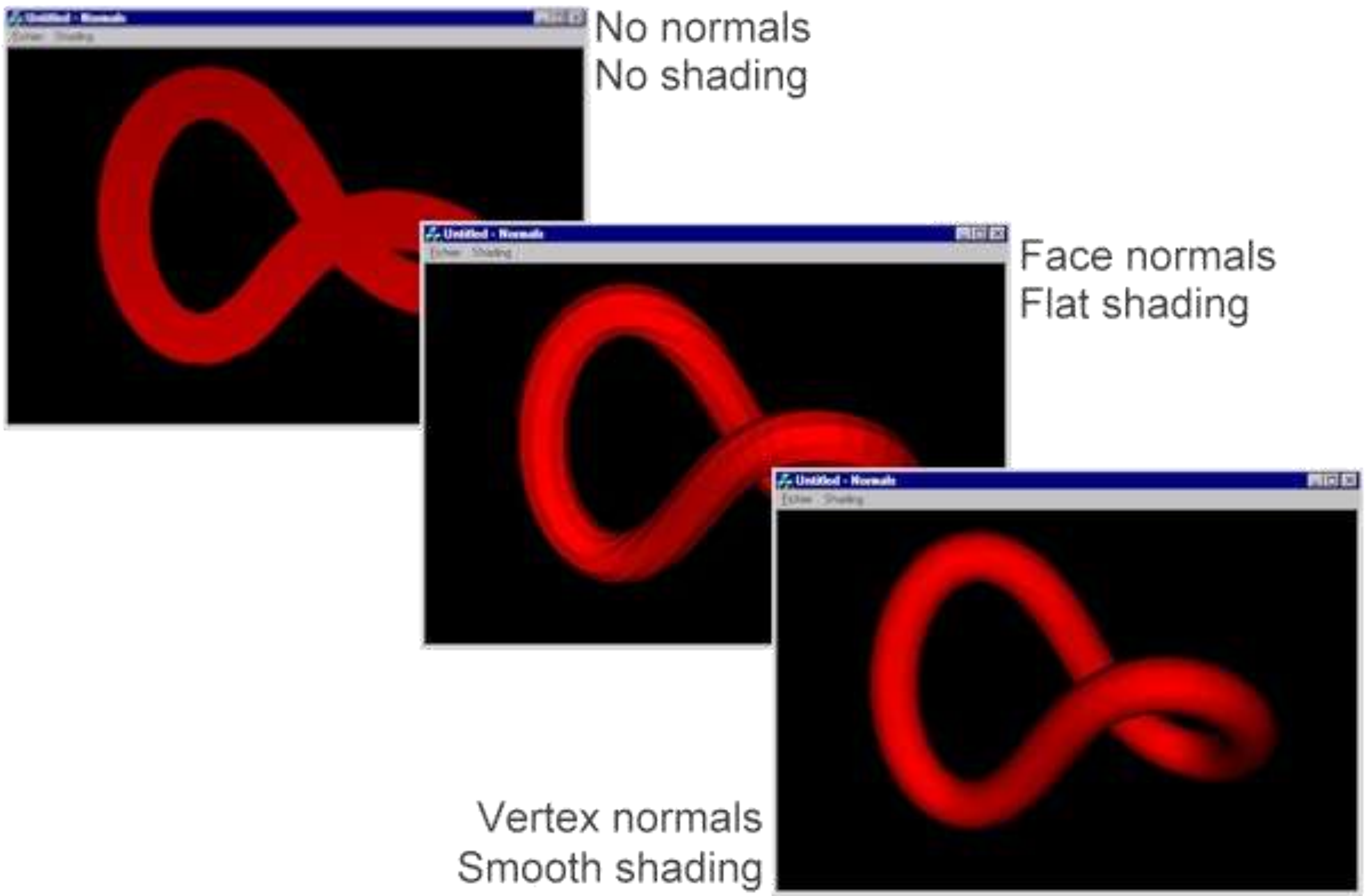
Ambient Only

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

Cartoony look

Color information

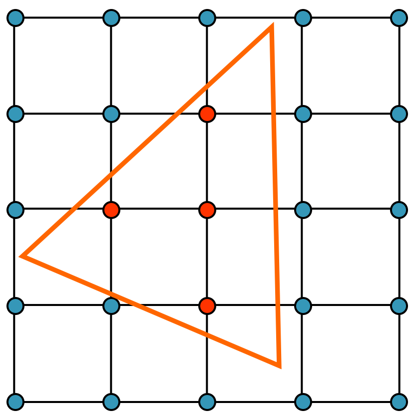
Per pixel lighting

Moving the vertices around

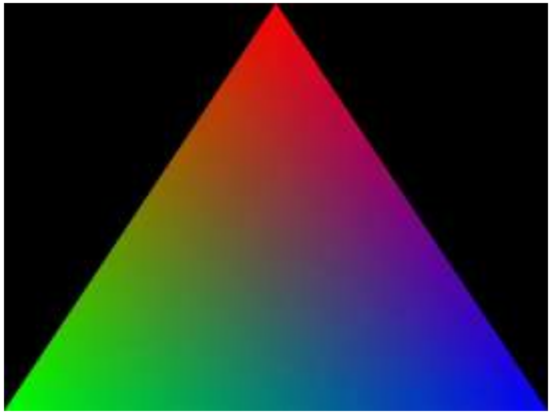
Textures

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.)



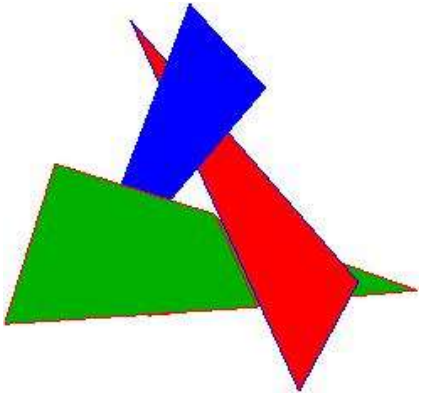
Rasterization

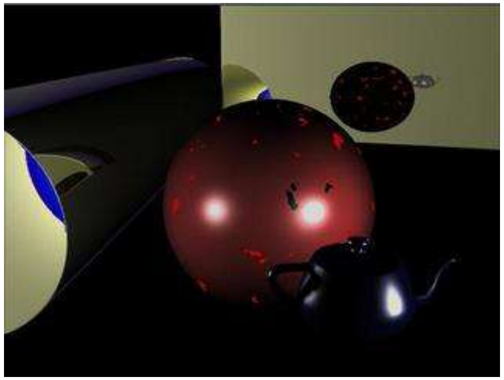


Linear Color Interpolation

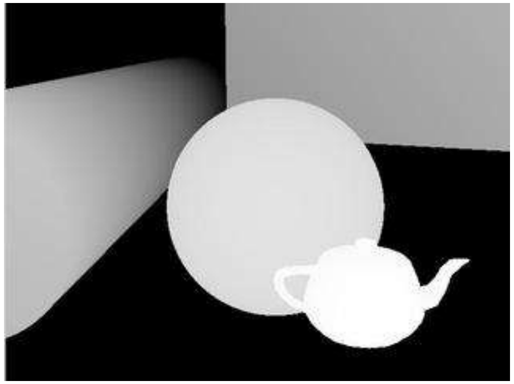
Z-test

- 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

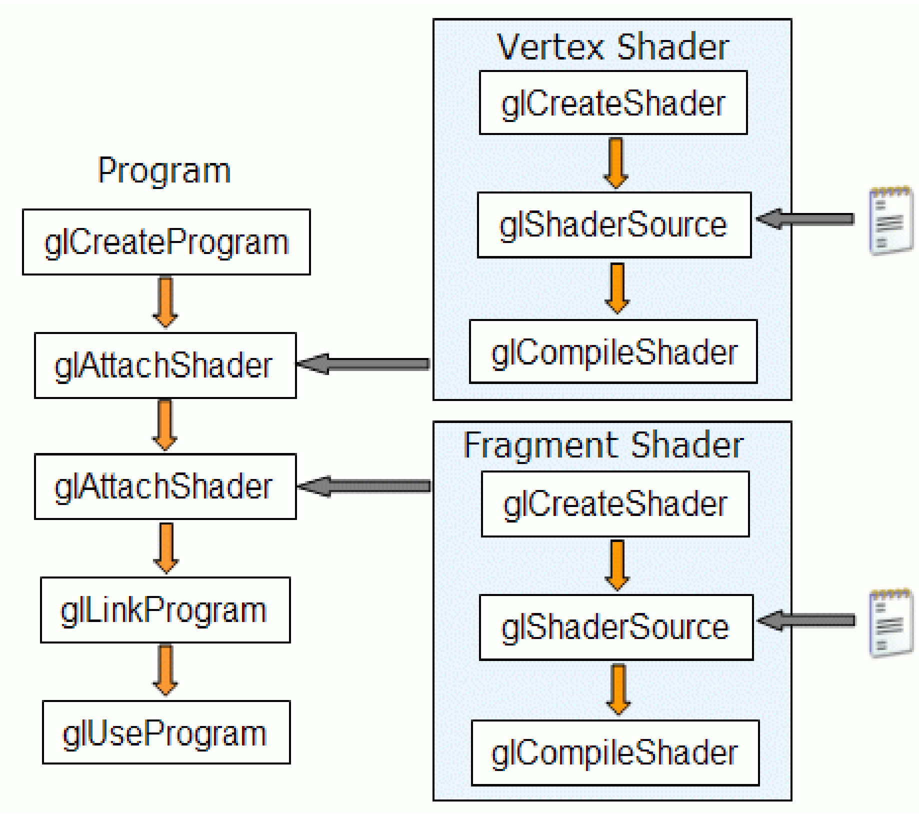
GLSL

- 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} `vec3 direction;`
- bvec{2,3,4} `bvec2 lightFlags;`
- ivec{2,3,4} `ivec3 color;`

- mat{2,3,4} square matrices

- sampler{1,2,3}D for textures
- samplerCube for cube map textures
- sampler{1D,2D}Shadow for shadow maps

- arrays
- structs

```
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)

```
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)

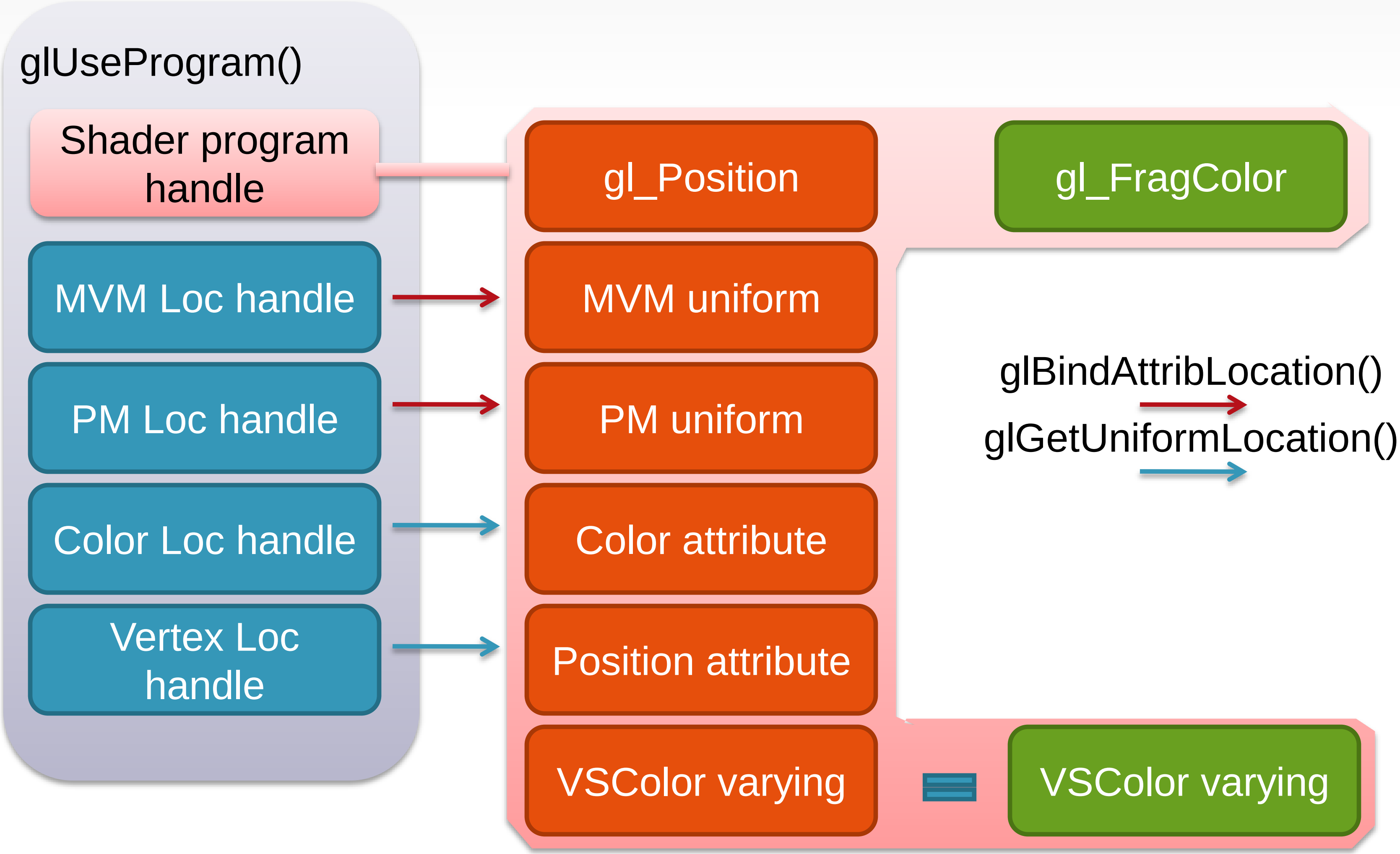
Sample Fragment Shader

```
precision highp float;
varying   vec3  g_vVSColor;
void main()
{
      gl_FragColor = vec4( g_vVSColor, 1.0 );
}
```

Compile the shader program

```
// 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" );
glBindAttribLocation( 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" );

glDeleteShader( hVertexShader ); glDeleteShader( hFragmentShader );
```

How it looks in a sample application



freescale™

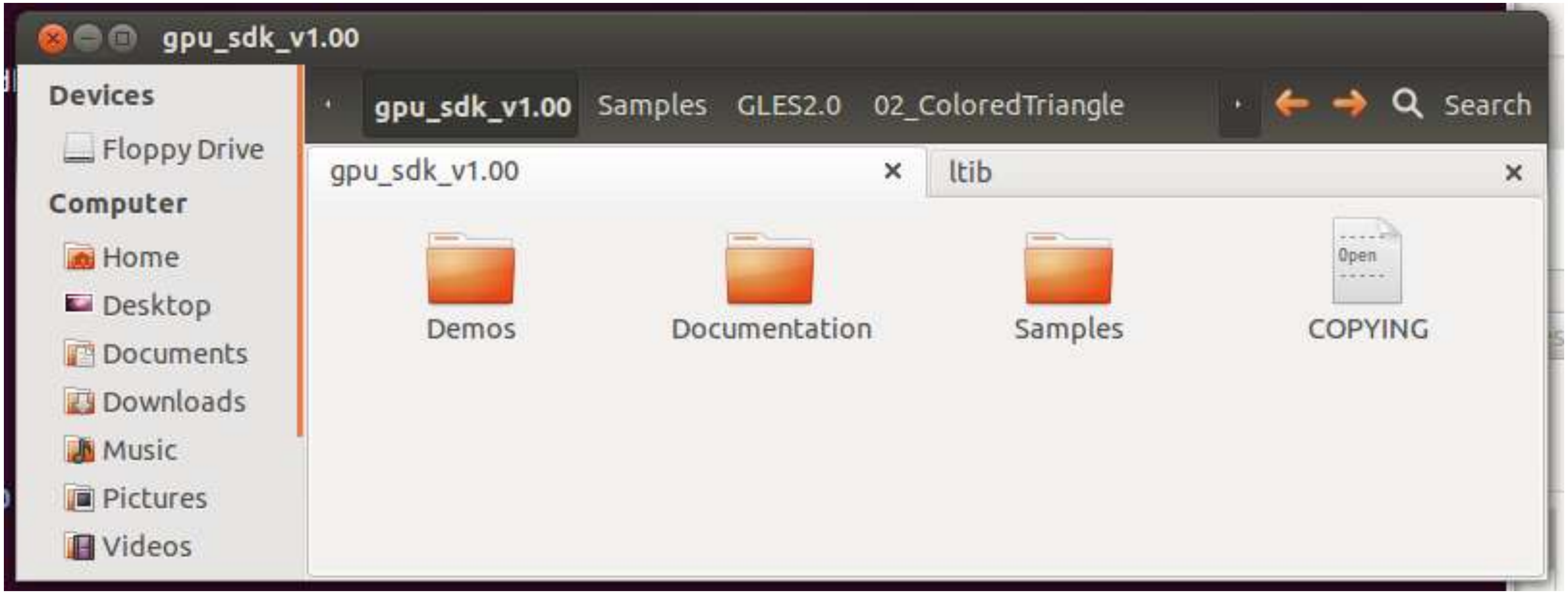
Render Loop

```
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



tree

```
.
├── Demos
│   ├── GLES2.0
│   │   └── simple_gpu_player
│   │       ├── bin
│   │       └── src
│   └── MultiAPI
│       ├── clusterDemo
│       ├── GIFT
│       │   └── resources
│       │       ├── required_libs
│       │       │   ├── assimp-2.0.863
│       │       │   └── DevIL
│       │       └── shaders
│       └── virtual_fb_driver
├── Documentation
│   ├── Tools
│   └── Tutorials
└── Samples
    ├── GLES1.1
    │   ├── 01_SimpleTriangle
    │   ├── 02_VertexColors
    │   ├── 03_VertexTransformation
    │   ├── 04_ColoredVerticesInterpolation
    │   ├── 05_GeometricObjects
    │   ├── 06_Projection
    │   ├── 07_BasicTexturing
    │   ├── 08_Multitexturing
    │   │   └── data
    │   ├── 09_Alphablending
    │   ├── 10_FilteringLights
    │   ├── 11_LightingFog
    │   │   └── data
    │   ├── 12_Stencil
    │   ├── 13_3DFonts
    │   │   └── data
    │   ├── 14_ParticlesSpritesAnimation
    │   ├── 15_ParticleAccelerator
    │   ├── 16_VertexBufferObjects
    │   │   └── data
    │   ├── 17_Beizer
    │   └── common
    │       ├── inc
    │       │   ├── FSL
    │       │   └── GLU3
    │       └── src
    ├── GLES2.0
    │   ├── 01_SimpleTriangle
    │   ├── 02_ColoredTriangle
    │   ├── 03_Transform
    │   ├── 04_Projection
    │   ├── 05_PrecompiledShader
    │   │   └── shaders
    │   ├── 06_Texturing
    │   ├── 07_EnvironmentMapping
    │   ├── 08_EnvironmentMappingRefraction
    │   ├── 09_VIV_direct_texture
    │   └── common
    │       ├── inc
    │       │   └── FSL
    │       └── src
    └── OpenVG
        ├── common
        │   ├── inc
        │   │   └── FSL
        │   └── src
        ├── CoverFlow
        ├── Example1
        ├── Example2
        └── Example3

70 directories
paco@ubuntu:~/vgct/external/gpu_sdk_v1.00$
```

GPU_SDK – requirements

- A working LTIB with required libraries
- Export the following variables from console:
  - export **CROSS_COMPILE**=/opt/freescale/usr/local/gcc-4.6.2-glibc-2.13-linaro-multilib-2011.12/fsl-linaro-toolchain/bin/arm-fsl-linux-gnueabi-
  - 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

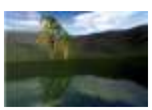
Product Manager – i.MX 6 and i.MX 7

Summary by Paco

i.MX 6Quad/6Dual 1Ghz and 1.2Ghz Benchmarking Results

Tegra3

	Device	i. MX6Q Sbera SD VerC	i. MX6Q Sbera SD VerC	Media Tek	Nexus 7	ASUS TF201
	OS Version	R13.4.1-RC4	R13.4.1-RC4	Android ??	Android 4.1.1	Android 4.0.3
	CPU	i.MX6Quad 1G	i.MX6Quad 1.2G	1.2Ghz Quad A7	Tegra 3 1.3Ghz	Tegra 3 1.2Ghz
	GPU	Vivante GC2000	Vivante GC2000	Unknown	Tegra 3 GPU	Tegra 3 GPU
	Memory	1G DDR3	1G DDR3	Unknown	1GB LPDDR2	1GB LPDDR2
	Screen Resolution	1024*728	1024*728	Unknown	1280*800	1280*800
Quadrant 2.0	Total	4028	4828	3384	3741	3978
	CPU	9554	11407		11069	10414
	Memory	2796	3327		2943	2983
	I/O	4354	5693		1259	3714
	2D	990	1000		990	314
	3D	2445	2445		2444	2464
AnTutu Benchmark 2.9.4	Total Scores	9605	11159	10429	10520	10011
	Memory	1950	2348	2131	2309	1996
	CPU Integer	3057	3665	3614	3391	3273
	CPU Float	2312	2782	2270	2746	2596
	2D Graphics	294	294	298	324	295
	3D Graphics	1248	1249	1255	1307	1189
	Database IO	400	480	520	85	325
	SD Card Write	150	137	150	148	140
	SD Card Read	194	204	191	210	197
AnTutu Benchmark 3.0	Total Scores		12603	NT	NT	NT
	Memory		2326	NT		
	CPU Integer		3747	NT		
	CPU Float		2784	NT		
	2D Graphics		768	NT		
	3D Graphics		2191	NT		
	Database IO		460	NT		
	SD Card Write		123	NT		
	SD Card Read		204	NT		
BaseMark ES20 (Taiji)	FPS	36		NT	17.93	14.65
NenaMark2	FPS	49.2		NT	46.5	46.4
M3D	Dynamic Lighting Benchmark	23.57921		NT	Can't run	16.73816
	Particle Systems Benchmark	26.32444		NT	Can't run	18.76312
	Triangles Based Benchmark	40.87296		NT	Can't run	23.14258
	Physics Based Benchmark	53.72527		NT	Can't run	35.68423
Kernel Bootup Time	seconds	20		NT	36	NT



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

freescale™

i.MX 6Quad/6Dual, 6DualLite, 6Solo vs Competition

	Device	SABRE SD	SABRE SD	SABRE SD	SABRE SD	Nexus 7	ASUS Eee PAD TF201	Galaxy Tab P7500
	OS Version	Android R13.4.1-RC4	Android R13.4.1-RC4	Android R13.4.1-RC4	Android R13.4.1-RC4	Android 4.1.1	Android 4.0.3	Android 3.1
	CPU	i.MX6Quad 1G	i.MX6Quad 1.2G	i.MX6 DL 1G	i.MX 6S 1Ghz	Tegra 3 1.3Ghz	Tegra 3 1.2Ghz	Tegra 2 1Ghz
	GPU	GC2000 GC320	GC2000 GC320	GC880 GC320	GC880 GC320	Tegra 3 GPU	Tegra 3 GPU	Tegra 2 GPU
	Memory	64bit DDR3	64bit DDR3	64bit DDR3	32bit DDR3	LPDDR2	LPDDR2	DDR2
	Screen Resolution	1024*728	1024*728	1024*728	1024*728	1280*800	1280*800	1280*800
Quadrant 2.0	Total	4011	4828	3005	2414	3741	3978	2559
	CPU	9435	11407	4813	2411	11069	10414	3141
	Memory	2809	3327	2807	2726	2943	2983	2584
	I/O	4371	5693	4166	3973	1259	3714	4988
	2D	1000	1000	817	608	990	314	280
	3D	2441	2445	2420	2352	2444	2464	1804
AnTutu 2.9.4	Total Scores	9605	11159	5583		10707	10675	
	Memory	1950	2348	971		2051	2226	
	CPU Integer	3057	3665	1527		3594	3648	
	CPU Float	2312	2782	1148		2694	2729	
	2D Graphics	294	294	294		298	293	
	3D Graphics	1248	1249	927		1242	1198	
	Database IO	400	480	385		500	300	
	SD Card Write	150	137	127		150	89	
	SD Card Read	194	204	204		178	192	
AnTutu 3.0.3	Total Scores	11126	12603		4531	12972	12161	
	Memory	1992	2326		672	2033	2121	
	CPU Integer	3133	3747		775	3714	3621	
	CPU Float	2303	2784		564	2690	2503	
	2D Graphics	765	768		474	797	730	
	3D Graphics	2212	2191		1423	3003	2643	
	Database IO	430	460		350	395	310	
	SD Card Write	134	123		139	150	90	
	SD Card Read	157	204		134	190	143	
BaseMark ES20 (Taiji)	FPS	25.65	25.63	7.78	7.67	17.93	14.65	
NenaMark2	FPS	49.2	49.1	30.5	27.2	46.5	46.4	
M3D	Dynamic Lighting	23.57921	23.54141	7.302674	7.254386	Can't run	16.73816	5.979404
	Particle Systems	26.32444	26.29946	10.13034	10.08059		18.76312	8.139534
	Triangles Based	40.87296	43.22133	19.66406	19.00992		23.14258	13.9976
	Physics Based	53.72527	60.34217	43.28051	23.14033		35.68423	22.77416

freescale™

AnTuTu v2.9.3, v2.9.4 and v3.0 Comparison (6Quad) vs MediaTek

	Device	i. MX6Q Sbera SD VerC
	OS Version	R13.4.1-RC4
	CPU	i.MX6Quad 1.2G
	GPU	Vivante GC2000
	Memory	1G DDR3
AnTutu Benchmark 2.9.4	Total Scores	11159
	Memory	2348
	CPU Integer	3665
	CPU Float	2782
	2D Graphics	294
	3D Graphics	1249
	Database IO	480
	SD Card Write	137
	SD Card Read	204
AnTutu Benchmark 3.0	Total Scores	12603
	Memory	2326
	CPU Integer	3747
	CPU Float	2784
	2D Graphics	768
	3D Graphics	2191
	Database IO	460
	SD Card Write	123
	SD Card Read	204

freescale™

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

Benchmark Suite	Item	i.MX6Quad 1Ghz 1024x768	i.MX6Dual 1Ghz 1024x768	Archos 101x Tablet TI OMAP 4470 1.5Ghz 1280x800 lcd	Archos 101 G9 Tablet OMAP 4460 1.2Ghz 1280x800 lcd
Quadrant	Total	4636	3300	3557	2622
Antutu	Total	10488	6911	5874	7835
Linpack	Multithread	136.71	87	94.9	82.6
Velamo		1086	752	1374	1325

freescale™

Relevant Benchmarks

i.MX 6 vastly outperforms Tegra3 and 2

UI Category	Benchmark	Unit	i.MX 6	Competition 1	Comme
1) UI's require 'quality' pixels to enhance view 2) 3D performance for User interface manipulation, games, video texturing					
	3DMarkMobile ES 1.1: Proxycon	**FPS**	**108 FPS**	**60 (iPad 2)**	**Older gen benchmark for more basic games**
	3DMarkMobile ES 1.1: Samurai	**FPS**	**103 FPS**	**60 (iPad 2)**	**Older gen benchmark for more basic games**
	3DMarkMobile ES 2.0: Taiji Girl	**FPS**	**35 FPS**	**24 (iPad 2) 5.1 (Tegra2) 15.8 (Tegra3)**	**Updated for latest 3D – tests whether complex games can be run at high FPS**
	3DMarkMobile ES 2.0: Hover Jet	**FPS**	**25 FPS**	**14 (iPad 2)**	**Updated for latest 3D – tests whether complex games can be run at high FPS**
3) UI content is inherently dynamic → Cannot predict content so need High speed, wide memory bus					
LMBENCH	Mem Read (MB/s) (higher is better)	**MB/s**	**264**	**163 (OMAP 4)**	**Ability to read content fast during decode or 3D operations**
LMBENCH	Mem Write (MB/s) (higher is better	**MB/s**	**2280**	**2221 (OMAP 4)**	**Ability to write data during decode or 3D operations**
LMBENCH	BCopy (libc) (MB/s) (higher is better)	**MB/s**	**590**	**312.8 (OMAP 4)**	**Ability to replicate frame buffers or other content especially during UI**
LMBENCH	L1 Latency (ns) (lower is better)	**ns**	**3.93**	**3.974 (OMAP 4)**	**How fast memory roundtrip is to CPU**
LMBENCH	L2 Latency (ns) (lower is better)	**ns**	**27.2**	**29.6 (OMAP 4)**	**How fast memory roundtrip is to CPU**
LMBENCH	Main Mem Latency (ns) (lower is better)	**ns**	**113.8**	**194.1 (OMAP 4)**	**How fast memory roundtrip is to CPU**
LMBENCH	Rand Mem Latency (ns) (lower is better)	**ns**	**172**	**265 (OMAP 4)**	**How fast memory roundtrip is to CPU**
Linux MemCopy		**MB/s**	**8K copies: 6975 64K copies: 4542 1M copies: 2674 16M copies: 1097**	**TI OMAP4 1Ghz on PANDA board 8K copies: 6510 64K copies: 3906 1M copies: 737 16M copies: 595**	**Determines efficiency of the memory controller for different size 'chunks' of memory. Video files tend to be large chunks. User data tends to be smaller chunks. Need to be good at all ranges**

Relevant Benchmarks – User Interface – Cont

UI Category	Benchmark	Unit	i.MX 6	Competition 1	Comments
4) UI requires high resolution support 1080p TV or LCD is now the norm					
	Run two 1080p h.264 high profile video clips on two different 1080p HDMI TVs	**FPS**	**30 fps per screen decode**	**Not possible on Tegra 2. unlikely to be capable on Tegra 3 due to single memory channel**	**Measures ability of chip to drive higher resolution displays than just 1080p. Plus ability to do picture in picture video playback simultaneously**
	CPU Core Utilization during Playback of video content	**% of max**	**4 %**	**Not measured**	**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**
5) Access to fast CPU MIPS → used for complicated transforms to augment visual experience					
	Dhrystone	**DMIPS**	2.45 DMIPS/Mhz	TI PANDA Board 2.2 DMIPS/MHz	**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.**
	CoreMark	**Score**	i.MX 6Quad processor at 1Ghz, 4 cores: 11147 coremark;	Nvidia Tegra 3 with **5 cores:** 11352	**Better score for CPU than Dhrystone. Much more complete set of tests to measure CPU performance**
	SunSpider (0.9.1) Java	**Sec**	run on i.MX 6Quad core 1: 1830; core 2: 1827; core 3: 1836; core 4: 1849	**OMAP4 @1Ghz on Panada Board: 1756 with two cores active**	**Browser test with broad support. Also a good measure for the CPU's ability to handle JAVA**
	Threading test	**Tbd**	Tbd	Tbd	**Measure ability to hit multiple threads and allocate effectively across multiple CPUs**

Tegra 3 limited by single memory channel

NXP

freescale™