Module 13C:
Using The 3D Graphics APIs
OpenGL ES

BREW™ Developer Training

Module Objectives

- See the steps involved in 3D rendering
- View the 3D graphics capabilities

3D Overview

- The 3D graphics library uses the industry standard OpenGL® ES API
- Supports custom 3D hardware rendering in handsets

OpenGL|ES
The Standard for Embedded 3D Graphics

QUALCOMM 3

OpenGL® ES

- Lightweight API for advanced 3D graphics capabilities on mobile devices
- It is based on well-defined subset profiles of the industry standard OpenGL 3D API available on Unix and the PC

QUALCOMM 4

Strong Industry Support for OpenGL-ES

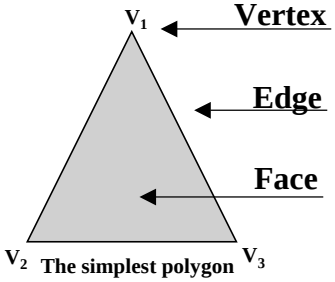
Promoting Members

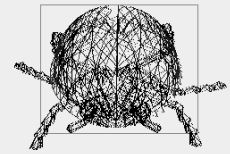
Contributing Members

A Few Definitions

- Rendering
  - The process of creating life-like images on a screen using mathematical models and formulas to add shading, color, lighting, and textures to a 2D or 3D wire frame
- Rendering Engine
  - "Rendering Engine" generically applies to the part of the graphics engine that draws 3D primitives, usually triangles or other simple polygons. In most implementations, the rendering engine is responsible for interpolation of edges and "filling in" the triangle

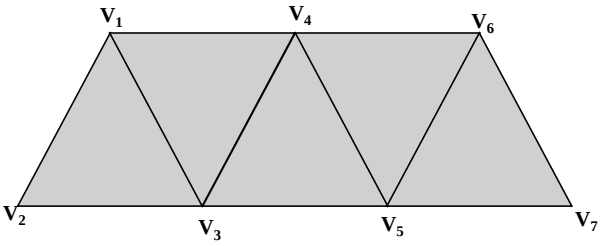
All About Triangles





1,014 vertices
1,816 triangles

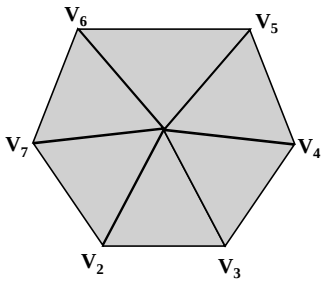
Reducing Vertices: Triangle Strip



Triangle	Vertices
1	1,2,3
2	1,3,4
3	3,5,4
4	4,5,6
5	5,7,6

Example: 5 triangles = 15 vertices, but represented as a strip there is only 7 vertices

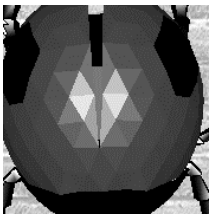
Reducing Vertices: Triangle Fan



Triangle	Vertices
1	1,2,3
2	1,3,4
3	1,4,5
4	1,5,6
5	1,6,7
6	1,7,2

Example: 6 triangles = 18 vertices, but represented as a fan there is only 7 vertices

Shading Triangles

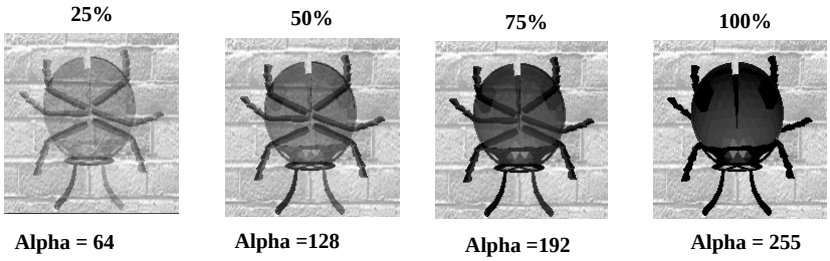


- **Flat Shading**
  - Each triangle is rendered with only one color, the color of the triangle is the average color of the 3 vertices

- **Smooth Shading**
  - Shading is a more complex process using algorithms to create a color gradient across the surface of each triangle

Alpha Blending

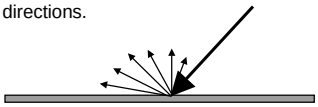
- The alpha value is really a mask -- it specifies how the pixel's colors should be merged with another pixel when the two are overlaid, one on top of the other.
- Rendering overlapping objects that include an alpha value is called *alpha blending.*

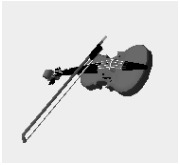


Alpha value is normally provided in the range (0-1), shown here normalized to (0-255)

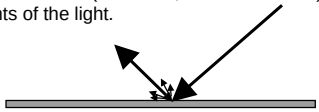
Lighting

- Ambient
  - Equal light in all directions.
  - Background light level caused by multiple reflections from all objects in the scene.
- Diffuse
  - Diffuse light comes from one direction, then as it hits a surface, it is scattered equally in all directions.



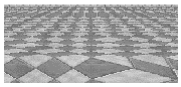


- Specular
  - Light comes from one direction, then, bounces off in a preferred direction as it hits a surface.
  - Material properties of the surface (shininess, emissive values) change the appearance or highlights of the light.

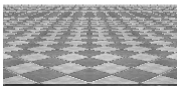


Perspective Correction

- Takes into account the effect of the Z value in a 3D scene while mapping texels (texture mapping pixels) onto the surface of polygons
- Without perspective correction, textured objects will appear to shift and 'tear' in an unrealistic way
- Texture coordinates are interpolated linearly when applied to objects
- If the object is projected into 3D space adjustment needs to be made to the texture coordinates

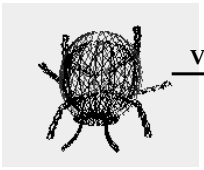


No perspective correction



With perspective correction

A Real Basic 3D Pipeline



Hardware Acceleration

- Hardware acceleration increases the 3D rendering capabilities in various ways
  - Increases number of triangles calculated per second for smother looking images
  - Increases pixel fill rate per second for smoother animation
  - Allows better blending and lighting techniques for more realistic images
- DSP hardware acceleration
- Custom 3D hardware, like on a PC

OpenGL-ES Graphics Available Functions

Geometry Processing
- Vertex Arrays
- Points, Lines, Triangles
- Matrix Stack
- Viewport, DepthRange
- Vertex Lighting
- ShadeModel

Rasterization
- Points & anti-aliased points
- Lines & anti-aliased lines
- Polygons
- Face Culling
- PolygonOffset - fill mode

Texture Mapping
- 2D Textures
- Wrap repeat, edge_clamp
- Compressed Texture
- TexSubImage, CopyTexImage
- Multitexture (1 texture supported)
- RGBA pixel and packed pixel formats, L, LA
- All Filters

Fragment Processing
- Fog
- Scissor Test
- Alpha Test
- Depth Test
- Blending
- Logic Op
- Dither

Framebuffer Operations/Miscellaneous
- Clear
- ReadPixels / Alpha Test / Dither
- Flush/Finish
- Hint
- Get-static state (constants)

OpenGL and BREW

- Download from BREW developer's site
  - BREW SDK Extension for OpenGL ES
- Include files
  - #include <gles/gl.h>
  - #include <gles/egl.h>
  - #include "igl.h"
- 2 BREW Interfaces
  - AEECLSID_GL - Interface to GL
  - AEECLSID_EGL - Interface to platform specific layer sitting between GL and BREW device
- Libraries for Windows and ARM

OpenGL and BREW

```
if( ISHELL_CreateInstance(m_applet.m_pIShell, AEECLSID_GL,
(void **)&m_pIGL) != SUCCESS )
        return FALSE;
else
        IGL_Init(m_pIGL);

if( ISHELL_CreateInstance(m_applet.m_pIShell, AEECLSID_EGL,
(void **)&m_pIEGL) != SUCCESS )
        return FALSE;
else
        IEGL_Init(m_pIEGL);
```

OpenGL and BREW

- Uses IDISPLAY_GetDeviceBitMap() to get pointer to screen's bitmap buffer
- Pointer passed to EGL routines
- EGL routines use dual buffers and manager buffer swapping

```
IBitmap*   pDBitmap;
IDISPLAY_GetDeviceBitMap(pMe->pIDisplay,
  &pMe->pDBitmap);
```


OpenGL and BREW

- BREW applet is predominately a shell
- Game logic isolated from BREW - most of the program consists of calls to gl functions contained within the libraries
- Key Events passed to game logic
- Timers, File IO, Networking handled by BREW

Examples

- BREW OpenGL ES Sample code
  - ogles_demo_02
- *www.gamedev.net*
  - *search "brew"*
- *www.opengl.org*
- *http://www.khronos.org/opengles*