Introduction to OpenGL ES 1.0

This article aims at providing an introduction to OpenGL ES for someone who is already experienced with desktop OpenGL, and shows how to use it with S60.

Introduction

OpenGL ES is a lightweight 2D/3D graphics library designed for embedded and mobile devices, based on the original OpenGL API. OpenGL ES version 1.0 is based on OpenGL 1.3, whereas OpenGL ES 1.1 is based on OpenGL 1.5. Currently, the Khronos Group is responsible for maintaining the OpenGL ES specifications.

OpenGL ES defines a concept named **profile**, that defines a subset of functionalities from the original OpenGL, plus features specific to OpenGL ES, such as extensions. Here they are:

- Common Profile: This profile is implemented in mobile devices such as cell phones and PDAs;
- Common-Lite Profile: This one is more restricted, and specifies the minimum required functionality to run 3D graphics applications. It is targeted at safety-critical devices, where reliability is a primary concern.

OpenGL ES also features an interface with the window system defined as EGL. The EGL API is common for all devices using OpenGL ES, but its implementation is hardware-dependent.

Required Files

SDKs from S60 2nd Edition FP2 onwards already have the required files to write OpenGL ES 1.0 applications. OpenGL ES 1.1 is available in the S60 3rd Edition FP1 SDK. The header files for OpenGL ES are:

```
#include <GLES/egl.h>
#include <GLES/gl.h>
```

OpenGL ES is implemented as a DLL, so it is necessary to use these import library files:

```
libgles_cm.lib
ws32.lib
```

The first file corresponds to OpenGL ES and EGL, Common Profile. The Common-Lite Profiles is not supported in S60 devices. The second file relates to the Symbian OS window server. For Carbide.c++, these files should have the `.dso` extension when building for the phone.

Tutorial: Starting up OpenGL ES

The following steps are required to start up OpenGL ES:

1. Retrieve the default display device;
2. Initialize OpenGL ES;
3. Choose an OpenGL ES configuration;
4. Create an OpenGL ES context;
5. Create a drawing surface;
6. Activate the OpenGL ES context.

As an example, we could use a class like the following one to accomplish these tasks:

```
#include <e32base.h>
#include <w32std.h>

#include "GLES/egl.h"
#include "GLES/gl.h"

class CGLRender: public CBase
{
  public:

    // method to create an instance of this class
    static CGLRender* NewL (RWindow & aWindow);

  public:

    // destructor
    ~CGLRender ();

    // double buffering, more on this later
    void SwapBuffers ();

  private:

   // constructor
   CGLRender (RWindow& aWindow);

   // second part of the two-phase constructor, where
   // OpenGL ES is initialized
   void ConstructL();

  private:
```


(C) Copyright Nokia 2013. All rights reserved.

```
    RWindow     iWindow;

    EGLDisplay  iEglDisplay;
    EGLConfig   iEglConfig;
    EGLContext  iEglContext;
    EGLSurface  iEglSurface;
};
```

The `iEglDisplay` variable represents the device screen. The OpenGL ES configuration is stored in `iEglConfig`. The `iEglContext` variable represents the OpenGL ES context. Finally, `iEglSurface` represents the drawing surface.

Retrieving the device screen

```
iEglDisplay = eglGetDisplay (EGL_DEFAULT_DISPLAY);
if (iEglDisplay == EGL_NO_DISPLAY)
 User::Panic (_L("Unable to find a  suitable EGLDisplay"), 0);
```

The `EGL_DEFAULT_DISPLAY` constant refers to the default device screen (in most cases there is only one). If the operation fails, the function returns `EGL_NO_DISPLAY`.

Initializing OpenGL ES

```
if (!eglInitialize (iEglDisplay, 0, 0) )
 User::Panic (_L("Unable to initialize EGL"), 0);
```

The last two parameters in this function correspond to the EGL implementation version. If you are not interested in it, just leave them as zeros. Otherwise, they are retrieved as this:

```
EGLint major, minor;

eglInitialize (iEglDisplay, & major, &minor);
```

For example, if the version is 1.0, major would be 1 and minor, 0.

Choosing an OpenGL ES configuration

Next, it is necessary to specify the minimum required configuration for the application.

```
EGLint numConfigs;

if (!eglChooseConfig (iEglDisplay, attribList, &iEglConfig, 1, &numConfigs) )
  User::Panic (_L("Unable to choose EGL config"), 0);
```

The `attribList` parameter represents an attribute list that the application requires. The function will return in the `iEglConfig` parameter a list of all the available configurations that match the attribute list. The size of this list is limited by the fourth parameter (in this case, we want only one configuration). The `numConfigs` parameter will inform the number of matching configurations, after the function returns. The attribute list defines a sequence of [attribute, value] pairs, as an array. The EGL specification defines constants for all supported attributes. For example, we will choose the color depth and z-buffer size:

```
// attribute list
EGLint attribList [] =
{
 EGL_BUFFER_SIZE, 0,  // color depth
 EGL_DEPTH_SIZE, 15,  // z-buffer
 EGL_NONE
};


// here we use the same color depth as the device
// iWindow is a RWindow object
switch (iWindow.DisplayMode() )
{
 case EColor4K:
  attribList [1] = 12;
  break;
 case EColor64K:
  attribList [1] = 16;
  break;

 case EColor16M:
  attribList [1] = 24;
  break;
 default:
  attribList [1] = 32;
}
```

The list should end with the `EGL_NONE` constant.

Creating the OpenGL ES context

```
iEglContext = eglCreateContext (iEglDisplay, iEglConfig, EGL_NO_CONTEXT, 0);

if (iEglContext == 0)
 User::Panic (_L("Unable to create EGL context"), 0);
```

The third parameter indicates a context to share texture objects. Here, we use `EGL_NO_CONTEXT` stating that there is no such context. The last parameter

(C) Copyright Nokia 2013. All rights reserved.

represents an attribute list to be mapped to the new context, and in this case there is no such list.

Activating the context

For OpenGL ES commands to take effect, it is necessary to activate the context, making it current. In OpenGL ES, only one context can be current at a time.

```
eglMakeCurrent (iEglDisplay, iEglSurface, iEglSurface, iEglContext);
```

Shutting down OpenGL ES

After using OpenGL ES, it is necessary to release all resources. Remember, this is **very** important!

```
CGLRender::~CGLRender()
{
 eglMakeCurrent (iEglDisplay, EGL_NO_SURFACE, EGL_NO_SURFACE, EGL_NO_CONTEXT);
 eglDestroySurface (iEglDisplay, iEglSurface);
 eglDestroyContext (iEglDisplay, iEglContext);
 eglTerminate (iEglDisplay);
}
```

The first line deactivates the current context. Then, the surface and the context are destroyed. The last line finishes OpenGL ES.

Some differences between OpenGL and OpenGL ES

By default, OpenGL ES uses double buffering. Here is how to perform this:

```
void CGLRender::SwapBuffers ()
{
  eglSwapBuffers (iEglDisplay, iEglSurface);
}
```

Due to limitations of embedded devices, OpenGL ES does not include many redundant operations from OpenGL. For example, it is not possible to use immediate mode for specifying geometry in OpenGL ES. Hence, code like this one is not valid in OpenGL ES:

```
glBegin (GL_TRIANGLES);
  glVertex3f (0,1,0);
  glVertex3f (-1,0,0);
  glVertex3f (1,0,0);
glEnd();
```

OpenGL ES renders all geometry from vertex arrays. So, this is the way to render a triangle in OpenGL ES, for example:

```
const GLbyte KVertices []=
{
 0,1,0,
 -1,0,0,
 1,0,0
};

...

glEnableClientState (GL_VERTEX_ARRAY);
glVertexPointer (3, GL_BYTE , 0, KVertices);
glDrawArrays (GL_TRIANGLES, 0, 3);
```

As many devices do not have a FPU (floating point unit), OpenGL ES profiles define functions that accept fixed-point values. Fixed-point math is a technique to encode floating point numbers using only integers. When using fixed-point numbers, a integer is divided in two parts: a bit range is used for storing the integer part, and the remaining bit range is used to store the real part. OpenGL ES works with 16:16 fixed-point numbers, which means that it uses 16 bits to represent the integer part and other 16 to represent the fractional part. More information can be found here.

An example of using the translation function as fixed-point:

```
glTranslatex (20 << 16, 0, 0, 1 << 16);

// same as
// glTranslatef (20.0f, 0.0f, 0.0f, 1.0f);
```

The functions that accept fixed-point parameters have a 'x' suffix. Additionally, OpenGL ES introduces the `GLfixed` type to represent fixed-point numbers. Some other differences worth to mention:

- Almost all functions that do the same thing, but have different signatures (like `glVertex3{fsidv}`) do not exist in OpenGL ES. However, some of them like `glColor4{fx}` are there;
- OpenGL ES supports only the RGBA color mode (you do not have to choose it);
- OpenGL ES does not render polygons as wireframe or points (only solid);
- It is not possible to query dynamic states in OpenGL ES 1.0 (for example, the current color);
- There is no GLU (OpenGL Utility Library). However, it is possible to find on the internet implementations of GLU functions, suitable for OpenGL ES;
- The `GL_QUADS`, `GL_QUAD_STRIP`, and `GL_POLYGON` primitives are not supported.

OpenGL ES tips

(C) Copyright Nokia 2013. All rights reserved.

- The `eglSwapBuffers` function should not be called inside the Draw method of the View class. The actual rendering should be performed by a timer method. A timer class or active object should call this method, possibly in a callback function;
- It is advisable to use fullscreen mode for OpenGL ES applications. Sometimes, when not using it, the application may end up updating just half of it (on the phone). An application can set fullscreen mode by using the `ApplicationRect()` method from the application UI when creating the view;
- Specify geometry using fixed-point numbers, because many devices lack FPUs;
- It is important to avoid many state changes. For example, mixing texture-mapped polygons with ones not using textures decreases performance. In this case, a better solution is to create two groups of polygons and render them separately;
- Disable perspective correction for distant objects (because the effect will not be very noticeable);
- Disable or reduce effects that are too far to be noticeable;
- Lighting is cool, but it can increase processing requirements, so take care;
- If possible, try to group several images into one texture, so texture changes are minimized;
- If possible, try to submit many polygons in a single call, rather than doing many calls with few polygons.

Appendix: Complete class

Here is the implementation file.

```
#include "CGLRender.h"

//_____________________________________________________________________________

CGLRender::CGLRender (RWindow & aWindow)
 : iWindow (aWindow)
{}

//_____________________________________________________________________________

CGLRender::~CGLRender()
{
 eglMakeCurrent (iEglDisplay, EGL_NO_SURFACE, EGL_NO_SURFACE, EGL_NO_CONTEXT);
 eglDestroySurface (iEglDisplay, iEglSurface);
 eglDestroyContext (iEglDisplay, iEglContext);
 eglTerminate (iEglDisplay);
}

//_____________________________________________________________________________

CGLRender* CGLRender::NewL (RWindow & aWindow)
{
 CGLRender* instance = new (ELeave) CGLRender (aWindow);
 CleanupStack::PushL (instance);

 instance->ConstructL();
 CleanupStack::Pop();

 return instance;
}

//_____________________________________________________________________________

void CGLRender::ConstructL()
{
 // attribute list
 EGLint attribList [] =
 {
  EGL_BUFFER_SIZE, 0,
  EGL_DEPTH_SIZE, 15,
  EGL_NONE
 };


 // get device color depth
 switch (iWindow.DisplayMode() )
 {
  case EColor4K:
   attribList [1] = 12;
   break;
  case EColor64K:
   attribList [1] = 16;
   break;

  case EColor16M:
   attribList [1] = 24;
   break;
  default:
   attribList [1] = 32;
 }

 // step 1
 iEglDisplay = eglGetDisplay (EGL_DEFAULT_DISPLAY);

 if (iEglDisplay == EGL_NO_DISPLAY)
  User::Panic (_L("Unable to find a  suitable EGLDisplay"), 0);

 // step 2
 if (!eglInitialize (iEglDisplay, 0, 0) )
  User::Panic (_L("Unable to initialize EGL"), 0);

        // step 3
 EGLint numConfigs;

 if (!eglChooseConfig (iEglDisplay, attribList, &iEglConfig, 1, &numConfigs) )
  User::Panic (_L("Unable to choose EGL config"), 0);

 // step 4
 iEglContext = eglCreateContext (iEglDisplay, iEglConfig, EGL_NO_CONTEXT, 0);

 if (iEglContext == 0)
  User::Panic (_L("Unable to create EGL context"), 0);


 // step 5
 iEglSurface = eglCreateWindowSurface (iEglDisplay, iEglConfig, &iWindow, 0);

 if (iEglSurface == NULL)
 User::Panic (_L("Unable to create EGL surface"), 0);

 // step 6
 eglMakeCurrent (iEglDisplay, iEglSurface, iEglSurface, iEglContext);
```


(C) Copyright Nokia 2013. All rights reserved.

```
}

//_____________________________________________________________________________

void CGLRender::EnforceContext ()
{
 eglMakeCurrent (iEglDisplay, iEglSurface, iEglSurface, iEglContext);
}

//_____________________________________________________________________________

void CGLRender::SwapBuffers ()
{
 eglSwapBuffers (iEglDisplay, iEglSurface);
}
```


(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.