Agenda

- Introduction to computer graphics and the 2D-ACE
  - Hands-on 1
- Creating graphics using vectors
  - Hands-on 2
- Fills and strokes
  - Hands-on 3
- Transformations
  - Hands-on 4
- Combining techniques
  - Hands-on 5 and 6
Introduction to Computer Graphics

- Computer graphics consist of a data set stored in memory which contains information about appearance of colored pixels on a display panel.
- Source data sets can be stored in various different ways including vector graphics and compressed formats (MPEG).
- In the vast majority of cases, the final data set is stored as a raster graphic.
- In this session we will only create these raster graphics by drawing them using the OpenVG vector language.
• Each **location** in memory contains information for a single pixel on the display panel.

• The size of the *location* in memory can vary from a single bit to a full 32-bit word.
  - This “bits per pixel” value determines how much memory is required to store the graphic.
The 2D-ACE Displays Raster Graphics

Memory → DCU → Graphic Display
Example 1 – Getting Started

Requirements:

Software (already installed on PC)
- IAR Compiler v6.70
- Example projects and debugger scripts (found at “C:\FTF_AUT_F0342”)

Hardware
- Vybrid Tower Card (TWR-VF65GS10)
- Tower Elevator Card (TWR-ELEV)
- LCD Display Card (TWR-LCD-RGB)
- Segger Jlink-Lite debugger

The tower card is powered by via USB. Cable should be connected to the micro USB connector (J3) closest to the elevator card.
Example 1 – Launching Project

1. C:\FTF-AUT-F0342\HandsOnProjects\HandsOnn\build-twrf600-mqx-a5\iar\ contains the project file: twrf600.eww. This will launch the IAR IDE which will be used for compiling and debugging for this class. A screen shot of IAR in compiler mode can be seen below:

- Files used in project
- Make
- Download and Debug
- Set breakpoint by clicking in margin
Example 1 – Launching Project

2. Click the “Download and Debug” button to launch the debug interface.

If changes have been made to code, press to recompile and d/load.

Use view to open memory/registers etc.

Quit Debugger!

Step/Over /Into etc

Break

Reset

Memory Window
Hands-On 1
Hands-on 1: Task list

- Download code and run
- Stop execution and bring up DCU0 Layers 20 – 22
- Enable layer 22 by setting EN bit
- Enable layer 20
- Move planet by setting DCU0_CTRLDESCL20_2 to 0x00000000
- Adjust position of planet by adjusting POSX and POSY fields
- Enable layer 21
- Adjust layer 21 TRANS value between 0 and 255 (0xFF)
- Bring up DCU0 layers 10-19
- Enable layer 10
- Move layer 10 around
- Finish by executing code - this performs the animation by doing exactly what we did by hand
Introducing OpenVG

• OpenVG is a standard API for hardware-accelerated 2D graphics
  – Managed by not-for-profit Khronos Group (also OpenGL)
  – Royalty-free, open standard
  – Designed for embedded systems
  – Draws vector graphics and manipulate 2D images

• The OpenVG API natively supports
  – Lines, curves, paths
  – Images
  – Filters, masks
  – Paint (gradients & textures)
  – Blending
  – Transformations
OpenVG Programming Model

- OpenVG is designed around a state machine-based client-server model
- User ‘sets’ and ‘gets’ variables in the machine (enable/disable, bind, etc.)

![Diagram]

- OpenVG: handles paints, paths, gradients, etc., to avoid re-preprocessing each frame
- Data types in a function are determined by i, f, or v postfix
  - Integer, float, or vector
- Execution decoupled
  - Execution is only guaranteed when user blocks (vgFinish) or flips the drawing surfaces (eglSwapBuffers)
Creating Drawing Surfaces Using EGL

• OpenVG needs a surface to draw on, and EGL provides that
• EGL defines three types of surfaces:
  – Windows: used for onscreen rendering
  – Pbuffers: used for offscreen rendering
  – Pixmaps: used for offscreen rendering into buffers that can be used by other native APIs
• The EGL API is also defined by the Khronos group
Creating paths

• Create a window using EGL
• Create an empty Path that will accept segment data – vgCreatePath.
• Fill Segment and point data to the path – vgAppendPathData.
• Once created draw the path using vgDrawPath.

• For convenience
  – We perform the path creation tasks in OvgApp_Init()
  – We do the drawing in OvgApp_Draw()
Hands-On 2
Converting to a diamond

VGbyte vgTriangleSegments[] =
{
    VG_MOVE_TO_ABS,
    VG_LINE_TO_ABS,
    VG_LINE_TO_ABS,
    VG_CLOSE_PATH,
};

VGfloat vgTrianglePoints[] =
{
    /* VG_MOVE_TO_ABS consumes 2 */
    60.0f,  50.0f,
    /* VG_LINE_TO_ABS consumes 2 */
    180.0f, 100.0f,
    /* VG_LINE_TO_ABS consumes 2 */
    300.0f, 50.0f,
};

vgAppendPathData(vgTriangle, 4, ...
Fills and strokes

- Paints are used to define the color and pattern that the stroke and interior of a path will have
  - They define a color and an alpha value for each pixel being drawn
- The steps to use a paint are similar than those of a Path
  - Create an empty Paint that will accept paint parameters information – vgCreatePaint.
  - Set parameters for the paint type.
    vgSetParameteri(#NAME_OF_PAINT#, VG_PAINT_TYPE…).
  - Set the parameter data for the selected paint type (In this case the type paint is color): vgSetColor((#NAME_OF_PAINT#, 0x0000FFFF);
- The vgDrawPath command can draw path outlines as we’ve seen or the path filled (or both)
Hands-On 3
Hands-on 3: Adjusting strokes and fills

• Swap vgTriangle and vgTriangle2 drawing order
  – See how the drawing order affects the final image
• Reconfigure strokePaint to draw each figure with a different colored and sized outline
• Give the figures different fill colors
Transformations

• Up to now we have been using a 1 to 1 relationship between OpenVG coordinates and the drawing surface
• In fact all OpenVG drawing operations pass through a transformation matrix
• In the examples so far the matrix is set to the identity matrix to give us the 1 to 1 translation
• Paths, strokes, fills, images and glyphs have their own transformation matrices
• There are special concatenated matrix combinations for common transformations vgRotate, vgScale, vgShear, vgTranslate
• Transformations are essential if you plan to include animate using OpenVG within your application
Hands-On 4
Hands-on 4: Transformations

• Apply the transformations to the triangles to see the effect
  – vgScale(float x, float y)
    ▪ Scales x and y by their respective factors
  – vgRotate(float angle)
    ▪ Rotates counter-clockwise by angle degrees
  – vgTranslate(float x, float y)
    ▪ Shifts the origin by (x,y)
  – vgShear(float x, float y)
    ▪ Applies a shear in the respective direction
  – vgLoadIdentity()
    ▪ Restore the transformation to 1:1
Combining techniques

- The final examples combine the various techniques
- We will use pre-rendered graphics alongside OpenVG generated graphics to create an optimised output
  - This approach is beneficial because it reduces the processing load on the OpenVG system with no loss in quality at the output
Hands-On 5 and 6
Hands-on 5: Slide status and graphics

- The application displays a slider which moves from minimum to maximum
- The value of the slider is passed to the OvgApp_Draw function as the parameter `adjust`
- Use this value to process one or more of the simple graphics on the left hand side of the screen

- In this example the application is forced to use a single buffer for all OpenVG processing
  - Output becomes very “choppy”
  - Disable this restriction by commenting out this line in ovgmain.c
    - `fb_info1.BUF_COUNT = 1;`
Hands-on 6: Album art

• This example has a more complex OpenVG manipulation of pre-rendered graphics.
  – Using vgScale, vgRotate and vgTranslate

• The background to the image is removed using alpha blending which allows the 2D-ACE to control what is in the background
  – The project simply changes the color

• Additionally the 2D-ACE modifies the alpha channel of the OpenVG graphics
  – Independently to the OpenVG animation

• The 2D-ACE also captures the currently selected album art
  – This would allow the image of the playing song to be displayed by the 2D-ACE while having an independent animation for selection
Summary

• The OpenVG standard allows graphics to be created using a vector language
• Complex graphics can be created by combing paths, strokes and fills
• Raster graphics can also be manipulated by the OpenVG GPU
• Combining the 2D-ACE capability with the OpenVG process allows for a wider range of graphic capability and better use of processing power
  – There is little point in using OpenVG to render a scene that is static