Influenced by their experiences with smartphones, users expect high-end HMI solutions on any of their devices. Cost-sensitive systems may lack available resources, graphics processing capabilities as well as the HMI framework itself. This paper discusses the main system requirements and introduces an efficient, yet powerful, system solution.

1. Introduction

Driven by smartphones with their high-resolution graphical touch displays, we experience a revolution in the control concepts of embedded devices in all kinds of markets. Intuitive graphical user interfaces find their way into a continuously raising number of products. Although this revolution is already far advanced especially in traditional Computer-On-Module (COM) and Micro Processor Unit (MPU) based products, it has not yet reached its final stage. Usually equipped with a full 3D graphical processing unit COMs and MPUs provide the highest graphics processing power—often by far exceeding the performance requirements of the targeted graphical user interfaces. A large variety of Human-Machine-Interface (HMI) frameworks is available for these platforms integrated in the OS like Silverlight in Windows Embedded Compact or OS independent like QT. Consequently, the introduction of smartphones like user interfaces (“Smart HMIs”) in such products was a rather straightforward process.

However, these “smart” user interface concepts are just starting to find their way into more cost-driven markets such as white goods. Microcontrollers are typically used here. In such processors either no or just comparably simple GPUs are applied. The implementation of graphical user interfaces providing the same (or at least similar) user experience like on smartphones is consequently a by far harder challenge. Graphics rendering engines and HMI frameworks optimized for the requirements of the expected user interfaces while meeting the low—because cost-driven—footprint requirements are needed.

The upcoming chapters of this paper discuss the expected capabilities of “Smart HMIs” and the derived requirements on the HMI framework and the graphics processing capabilities. A cost-efficient system solution consisting of a processor optimized for modern user interfaces in combination with an HMI framework is introduced, designed for embedded devices providing an up-to-date user experience and a rapid design flow.

2. 3D or 2D Graphics

OpenGL ES 2.0 is a full 3D graphics API which has advanced to a de-facto standard, certainly also driven by the high number of software engineers in the 3D game industry. The Android OS and the HMI framework QT5.x—both widely used in embedded devices—require a hardware platform fully compliant to the OpenGL ES 2.0 specification. However, as the silicon footprint of a 3D GPU is by factors higher than the footprint of a 2D GPU, avoiding the necessity to include the OpenGL ES 2.0 API is one of the biggest system cost saving potentials.

The concept of 3D GPUs is usually based on thousands of triangles forming 3D objects. Z-buffer handling in hardware supports computing complete 3D scenes with several objects overlapping each other. The triangles are then filled with textures perceptively transformed into the 3D domain. Such a concept is optimal for graphics applications like 3D games, 3D modeling or terrain rendering.

The concept of 2D GPUs is usually based on a set of basic graphic operations like BLITting (BLIT = Block Level Image Transfer), drawing of lines, filled- or unfilled polygons, ellipses and so on.

It is obvious that such a concept is much more suited to the needs of a graphical user interface with modalities like buttons, icons, sliders, and progress bars. Also modern modalities like wheels or carousels can perfectly be implemented with a 2D rendering engine with scaling and blending capabilities.

Picture 1: Carousel widget purely based on 2D graphics processing

Of course, there are exceptions; one of the most popular ones might be a 3D cover flow flipping CD or DVD covers with a 3D effect. In that case, the perspective transformation capabilities of a hardware-based 3D rendering engine are
significantly improving the overall rendering performance. Avoiding such exceptional modalities that require 3D graphics acceleration is usually not a severe limitation of the HMI design. One might look at his own smartphone and think about how much 3D graphics rendering is applied in the user interface – nearly none! Consequently we can conclude that a system with 2D graphics processing capabilities is absolutely sufficient in most cases and thus one of the highest cost saving factors.

3. Graphics Rendering Requirements

As discussed a 2D graphics rendering engine is sufficient for rendering modern graphical user interfaces. Now we should look a bit deeper into the required graphics processing features. In order to do so we will analyze the graphics operations in some of the more advanced modern HMI modalities such as the carousel, kinetic scrolling, gauges and font rendering.

The BLIT operation is certainly the most important function. A BLIT copies (parts of) source images to a defined destination coordinate in the graphics output buffer. With just a simple BLIT engine all standard HMI elements – buttons, icons, bitmap text etc. – can be rendered. It is the basis of the more sophisticated functionality discussed in the following. Especially when movements and animations are needed the BLIT operation has usually to be extended with scaling, filtering and blending.

Let’s start with the carousel. The carousel displays its elements in a quasi-3D representation: Elements in the front are enlarged, elements in the back are downsized, i.e. all elements have to be scaled depending on their position. In order to avoid pixel-artifacts during the movement especially in highly detailed areas and on edges sub-pixel exact rendering in combination with bilinear filtering is needed. An output pixel is computed based on the interpolation of four input pixels whereas the interpolation factors correlate with the sub-pixel position.

As some of the carousel’s elements might be transparent or partly transparent, alpha blending during the rendering process is required. Usually several blending modes are offered in a 2D graphics engine to cope with different use cases and look & feel requirements.

Technology-wise a worst case scenario for an HMI is a slow full-screen animation (e.g. kinetic scrolling of the screen content). Since in this case the complete screen is manipulated, this is one of the most demanding use-cases for a fast BLIT operation. Again, pixel-artifacts should to be avoided by applying sub-pixel exact rendering in combination with bilinear filtering.

Also intelligent caching mechanisms between the graphics rendering engine and the HMI framework as well as a display controller supporting blending of several layers can support implementation of full-screen animations.

The needle in a gauge can be realized for example by rotating and blitting an image of the needle or by using a primitive 2D graphics operation such as drawing one or several lines or a polygon. Also here it is essential to avoid aliasing artifacts – visible as a stair-case effect – especially at the edges of the needle. Again, sub-pixel accurate rendering in combination with anti-aliasing techniques is required.

We can conclude that for modern, “smartphone like” HMIs a simple BLIT engine is not sufficient, but certain feature enhancements and processing attributes are needed: scaling and blending can be regarded as must-haves,
rotations and primitive graphics operations (such as lines, polygons, arcs) help for some modalities like gauges. All these operations shall be executed with sub-pixel exactness and appropriate anti-aliasing techniques.

4. Vector vs. Bitmap Fonts
Displaying text is certainly one of the most important user interface elements. Regarding the fonts two fundamental concepts can be distinguished: vector fonts (e.g. FreeType) and bitmap fonts. In a vector font each character is represented in a vector description, i.e. it can be drawn by a sequence of simple graphics operations. This has the advantage that a character can be drawn in any size selectable at runtime without artifacts, provided that the graphics rendering engine offers proper anti-aliasing and blending functionality!

In bitmap fonts each character is represented by an image which is stored in memory. In order to avoid artifacts at the edges of the character these images should be stored with an alpha channel. A bitmap font is pre-computed offline, e.g. based on vector fonts available on PC. In principle also each font size can be provided but the sizes finally available are selected at design time.

Supposing that decent font rendering techniques are used – either online for vector fonts or offline for the generation of bitmap fonts - and that the bitmap fonts are stored with sufficient colour depth, the quality of vector fonts and bitmap fonts can be equal.

It shall be noted that very often vector fonts are implemented in a way that all characters are computed once at application start-up, i.e. a bitmap font is generated, and the application then works based on the online generated bitmap font.

We can conclude that choosing the right font type strongly depends on the target application requirements. Depending on the required number of fonts and sizes, the colour depth, the available processing power (vector fonts need CPU time!), one or the other might be the right choice.

5. HMI Framework requirements
Discussing all requirements for modern HMI frameworks in detail would by far exceed the scope of this paper. However, we will discuss on some key requirements with respect to the realization of “smartphone like” graphical user interfaces on cost efficient embedded systems.

Animations
One of the eye-catchers in today’s smartphone graphical user interfaces and therefore a key element for their lifestyle image are animations. Either single icons are animated or complete dialog transitions are implemented with all kind of graphical effects. The latter requires high graphics performance and is often reserved for high-end platforms. But for cost-efficient systems a limited set of eye-catching animations is also possible, and allows increasing the product’s quality rating.

Reaction time
The user’s experience is significantly influenced by the reaction time to the users input. In principle an “immediate” response is expected. Generally spoken, a latency below 10 ms results in a smooth usage perception. Also the start-up performance is crucial. The typical user expects an “instant on” behavior, thus the HMI framework has to provide features like fast start-up time, intelligent resource loading or even the loading of resources in a dedicated background task.

Multilingual
Since most products are sold internationally being able to switch between different languages dynamically at runtime is essential. Consequently UTF-8/16 support together with flexible text layout functionality is required.

Resource management
Especially for very cost sensitive embedded systems the economical usage of the system resources is essential. The amount of volatile and non-volatile memories and the memory bandwidth are often the most limiting factors; the limited processing power has also to be considered. On portable systems the battery lifetime also adds to this list. The HMI framework has to take this into account by applying intelligent algorithms and concepts such as smart re-draw, texture and image compression techniques and intelligent caching mechanisms. The program memory size of the HMI framework itself is very often another relevant factor.
6. Model View Controller (MVC) Architecture

Besides the technical requirements discussed above an HMI framework shall also ensure high reusability, portability and maintainability. This is supported by applying the Model-View-Controller architecture concept providing the following advantages:

- An HMI representation can be easily re-used on different products.
- Product extensions and adaptations (e.g. a touchscreen instead of buttons) can be easily supported.
- Porting an HMI to different target platforms is simplified.
- The strict separation of the MVC components eases debugging and maintenance.

The following diagram illustrates the interaction of the three MVC components: Model, View and Controller:

![Model-View-Controller (MVC) concept](image)

**Model:**
The model contains the application data to be presented via the HMI – and often also applications logic itself.

**View:**
The view presents the application data to the user. In modern architectures the view component is also responsible for capturing the user input – although not for its interpretation or further processing.

**Controller:**
The controller processes the user interaction and updates the view or model components accordingly. Typically it is implemented as a state machine handling the HMI state flow.

7. HMI development workflow

The growing expectation for the quality of graphical user interfaces requires a closer interaction between the designers and software engineers. Consequently a new workflow is required supported by an appropriate HMI tool chain.

Graphical user interface designs are usually developed by using tools like Photoshop and starting with a sketch or prototype design in the early phase of a project. In this phase it is recommended to already establish a dialog between design artists and engineers to create awareness for the graphical capabilities of the target platform. While the designer is responsible for the appearance the software engineer is responsible for the behavior of the user interface on the target. Using an HMI framework allows quickly implementing the behavior and porting the user interface to the target system.

The HMI framework should be provided together with an HMI tool chain that supports direct import of the media outputs of the design tools, easy development and simulation of the HMI behavior in PC environment as well as an easy process to install an HMI on the target platform. The core element of these tools is usually a “WYSIWYG” (What You See Is What You Get) Editor. Such a tool chain supports the above mentioned dialog and allows an interactive and iterative design flow.

![HMI development workflow](image)
8. An HMI system solution at the sweet spot of performance, features and cost

We have discussed the basic requirements for modern graphical user interfaces on highly cost driven products. We have seen that sub-pixel exact 2D graphics processing including blending, scaling, bilinear filtering and anti-aliasing is required, while a 3D rendering engine can be regarded as significant overhead. The HMI Framework shall exploit these graphics capabilities and shall be designed for portability and resource efficiency. A tool chain supporting a rapid development flow and an interactive and iterative workflow between designers and engineers is required. Renesas’ RZ/A MPU in combination with the TES Guiliani HMI framework and the TES eGML graphics rendering library is an HMI system solution at the sweet spot of performance, features and cost. The Renesas RZ/A1 MPU has been designed to give an optimum cost / performance balance for Human Machine Interface (HMI) applications. In contrast to a standard microprocessor architecture the family incorporates up to 10 MB of embedded SRAM, meaning that there is no need for external RAM. It can also give a significant performance boost versus devices required to run code over an external bus interface. This means highest performance can be achieved and without requiring external RAM; the system cost and complexity can be reduced dramatically. TES’ Guiliani is a platform independent HMI framework designed to meet today’s user expectations for smartphone-like HMIs on highly cost-driven embedded systems. Being OS- and CPU-agnostic and without requiring an expensive hardware GPU it covers a wide range of cost efficient MCUs and MPUs and matches ideally with Renesas’ RZ/A family. Guiliani is delivered with an extensive but customizable and extendible set of modern widgets and features like carousels, wheels, gauges, animations, transition effects, multi-language support and skinning. By default applying modern graphics processing features like sub-pixel accurate rendering, anti-aliasing, scaling, filtering and blending in combination with smart redraw and caching mechanisms leads to an eye-catching and high-performance visual experience. Guiliani’s PC drag & drop “WYSIWYG” editor and simulator support rapid HMI design and prototyping resulting in fast development cycles.

More information about Guiliani can be found on

Sources

Media Contact
Thomas Hase
Senior Business Development Manager – Embedded Graphics
TES Electronic Solutions GmbH
Frankenstr. 3
D-20097 Hamburg
Germany
Tel. +49 176 1011 1874
Email: graphics@tes-dst.com
A "Smart HMI" system solution for cost driven embedded systems