Embedded Wizard
GUI Solutions by TARA Systems

GUI Development on Resource-Constrained Embedded Systems

The Balancing Act Between Memory, Performance, and Design

Manuel Melic and Vanessa Reingruber, TARA Systems GmbH

This paper examines the challenges and solutions involved in GUI development on 32-bit MCUs with limited memory. It explains the tension between performance, memory, and design, and discusses how developers can balance these factors.

A graphical user interface (GUI) is an interface that allows users to interact with an electronic device in an intuitive way by using visual elements such as icons, buttons, and text on a display. There are various types of displays where GUIs can be used. In this article, we focus on simple displays connected to a host system. The host system is an embedded system.

Embedded systems are specialized computers designed to perform specific tasks within larger systems. They can be found in a wide range of applications, from household appliances to medical devices to automobiles. Embedded systems can be based on microcontroller units (MCUs) or microprocessor units (MPUs). MCUs integrate all the necessary components on a single chip, making them cost-effective, compact, and energy-efficient; ideal for simple, specialized tasks. However, they have limited processing power and memory. MPUs, on the other hand, offer higher processing power and expandable resources, making them suitable for complex, high-performance applications, though at the cost of higher energy consumption and more complex integration[1].

Challenges of GUI Development on MCUs

Developing GUIs on MCUs presents developers with a number of complex tasks. What requirements does the target hardware impose on GUI development? How can limited memory be used efficiently without compromising visual quality? How can the design be optimized to both save memory and ensure high performance? What specific challenges arise when implementing animations, and how can these be managed with the available resources?

We summarize these challenges in the so-called tension triangle, which describes the balance between three central factors: performance, memory, and design. When developing GUIs on MCUs, developers must reconcile these three aspects.

[1] J.T. Wunderlich.

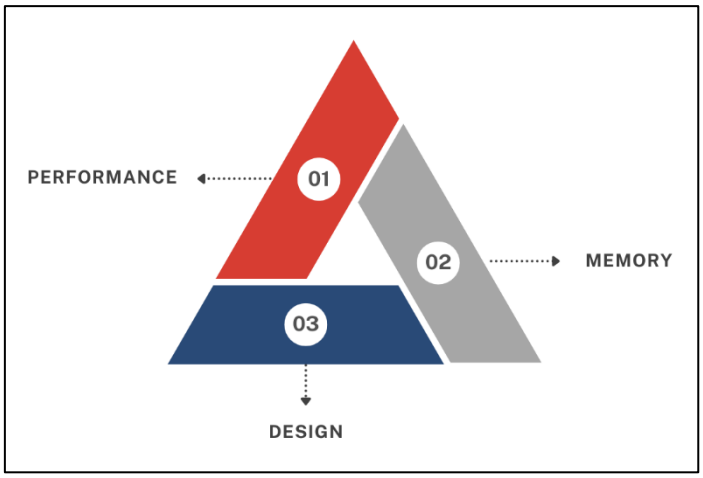

Figure 1: The Triangle of Tension in GUI Development. Own depiction.

The Balancing Act Between Memory and Performance

One of the greatest challenges in GUI development lies in the complexity of the data that needs to be processed. The smallest unit of a screen, the pixel, is crucial for displaying content. In a full-color design, the color depth of a pixel is typically calculated as 1 byte per color channel (red, green, blue) per pixel. However, if a multi-layer operation is required—such as a GUI overlaid on a video (e.g., in a rear-view camera)—an alpha channel is added, which stores transparency information. In this case, each pixel requires 4 bytes, as the alpha channel must also be stored alongside the RGB values.

While this might initially seem like a manageable amount of data, at a resolution of 800 x 480 pixels, the memory requirement adds up quickly. Furthermore, during the construction of a single frame, it is important to ensure that no unwanted drawing operations are visible in the background, for example when text is placed over images. This is where the concept of double buffering comes into play: the next frame to be displayed is built in the background while the current frame is shown. Once the construction is complete, the frames are swapped to ensure smooth transitions.

The challenge here is that these large amounts of data must be processed quickly and frequently. MCUs, compared to MPUs, have lower processing power, which makes processing complex graphics and animations more difficult. Additionally, MCUs typically have limited amounts of flash memory and RAM available, which limits the amount of data that can be stored and processed for the GUI. Studies have shown that the human eye perceives stuttering at frame rates below 15-20 Hz[2], which is why a frame rate of at least 25 frames per second is typically targeted. The rule is: the higher the frame

[2] vgl. Chen and Thropp, 2007.

rate, the smoother the GUI. Here we see the first major balancing act—the need to ensure high performance while maintaining limited memory space and processing capacity.

This dilemma requires creative solutions. One option to consider is foregoing a GUI framework. Frameworks provide predefined components that can speed up development and simplify handling complex GUIs, but they come with additional memory and resource overhead. With limited memory and processing power, it might make sense to forgo a framework and develop a custom solution. This could be more efficiently tailored to specific hardware resources, although it would require more development effort.

The RAM layer offers another way to reduce memory consumption. This involves minimizing double buffering to single buffering. Here, the image construction is synchronized with the update behavior of the display (using V-Sync), so only the non-visible part of the image is updated in the background. This reduces memory usage but can result in unwanted construction artifacts (e.g., staircase effects) if synchronization is lost.

Another approach to reducing memory consumption concerns flash memory, especially when processing background images that take up a lot of space. Storing them in raw format can overload the flash memory and slow down loading into the framebuffer. One solution is to compress the images, ideally in a lossless format, to save memory. For example, Runtime Length Encoding (RLE) could be used, which allows for fast decompression.

The Balancing Act Between Memory and Design

Designing user interfaces also poses a challenge due to limited memory space. GUIs, particularly those with high graphical requirements, require a significant amount of memory to store graphics, textures, and other visual elements. A colorful, detailed design demands a correspondingly large color space, which increases memory usage.

To address the design-memory challenge and save RAM, techniques like palette-based color formats or dithering can be employed, which reduce the color depth. From a greater distance, these savings are often not visible, but they may require compromises in design quality.

Another example is vector graphics, which are particularly useful for rendering icons. Icons can be described as paths and rendered at runtime, which requires more CPU power but reduces memory usage. This is especially useful when icons are needed in different sizes, as rendering in real-time uses less flash memory than storing multiple image versions.

The Balancing Act Between Design and Performance

Another problem is that users today expect aesthetic, smartphone-like graphics. This often requires elaborate animations and effects, such as 2.5D transformations (e.g., in the form of perspective distortions). Such elements often demand significant processing power. On MCUs, this leads to a dilemma: Should the visual demands be reduced in favor of performance? Or should performance be sacrificed to create an aesthetically appealing interface?

A common challenge is that videos cannot be displayed on MCUs due to limited processing power. In such cases, one might try to replace animations with programmed vector graphics that enable morphing effects. This method is more efficient than videos and can provide an attractive, schematic representation, but it is not suitable for complex content.

When implementing design elements such as perspective transformations, a compromise between visual appeal and system performance must often be found. A typical example is the Coverflow animation, which, due to its computational intensity, often places high demands on the CPU unless hardware acceleration is used. When the pixels do not align perfectly with the grid, bilinear filters can be applied to smooth them. While these filters improve visual quality, they require intensive CPU resources, as each pixel must consider all neighboring pixels in multiple iterations. Consequently, it may be necessary to disable such resource-intensive filters to boost performance, which, however, leads to a less appealing design result.

Another approach to solving design-performance issues is to integrate specialized hardware components such as graphics accelerators. These accelerators are designed to efficiently process graphics-intensive tasks without burdening the CPU. This allows for the implementation of demanding graphical effects, such as perspective transformations or animations, with minimal impact on system performance.

The Triangle of Tension Between Memory, Design, and Performance

In summary, the art of GUI development on MCUs lies in optimally balancing the three factors: design, performance, and memory. This tension triangle means that every decision in one area has an immediate impact on the others. There are many ways to optimize each available resource down to the last bit, but if overdone, at least one of the other aspects will suffer.

With advances in semiconductor technology, such as more powerful graphics accelerators and improved memory solutions, MCUs may soon be capable of supporting GUIs that are currently unimaginable. This development opens the door to increasingly sophisticated and visually appealing applications on resource-constrained systems, which will push the boundaries of the tension triangle in the embedded world.

References

[1] J. T. Wunderlich: Focusing on the Blurry Distinction between Microprocessors and Microcontrollers. American Society for Engineering Education Annual Conference & Exposition, Session 3547, 1999.

[2] J. Y. C. Chen and J. E. Thropp: Review of low frame rate effects on human performance. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 37(6), pages 1063-1076, 2007.

List of Figures

Authors

Vanessa Reingruber is a marketing manager at TARA Systems and responsible for Embedded Wizard. With a passion for content marketing, she creates content that clearly and understandably presents complex technical concepts. Her intention is to make expertise in GUI development with Embedded Wizard accessible and to strengthen the connection between technology and users.

Manuel Melic completed his computer science degree in 2007. After positions as a software engineer and project manager, he has been the product manager for Embedded Wizard, a market-leading and award-winning solution for embedded GUIs from TARA Systems, since 2015. In this role, he not only manages key customers but is also significantly involved in the product's development. His goal is to expand TARA Systems' portfolio through synergistic and innovative developments. As an experienced expert in embedded systems and GUI development, he is also eager to share his knowledge and experiences in these areas.

Contact

Website: www.embedded-wizard.de/contact

E-Mail: contact@embedded-wizard.de

Further information