Graphics Sharing & Distributed HMI

2019 Project Update

Gunnar Andersson, GENIVI Development Lead, 14 May 2019

GENIVI®

Fundamentals – project name

“Graphics Sharing”

Graphics in the form of
- bitmap,
- scene graph / model representation,
- or drawing commands,

...generated on one ECU* and transferred for display by another ECU* (or between virtual machine instances)

* incl. related connected systems such as Mobile Device

Fundamentals – project name

“Distributed HMI Compositing”

Methods and technologies to turn a multi-ECU system into what appears and acts as a single user-experience

Graphics Sharing & Distributed HMI

1 Knowledge sharing

2 Create classification of methods

3 Examples of methods and existing De-facto standards?

4 Case Studies

5 Methods of analysis, choosing the right method

6 Reference designs, blueprints and recommendations

7 Related (general) graphics technologies

8 Published documents and Communication

1. Knowledge Sharing

Knowledge Sharing

- Reaching “same level” understanding
  - Required for collaborative innovation
- Market survey – needs & availability
- Runs as a theme throughout projects
- Nomenclature

2. Methods Classification

Methods Classification

The 5 Categories of Graphics Sharing technologies

A) GPU Sharing

B) Display Sharing

C) Surface Sharing

- Sub-category: Virtual Display

D) API Remoting

E) Shared state - independent rendering

A) GPU Sharing

The GPU can be used from multiple operating systems, so it is shared. Concurrent access to the physical GPU has to be controlled by a hypervisor, hardware or other means which are implementation specific.

A) GPU Sharing

Multiple Virtual Machines access a single GPU*

Considerations:

API standard?

*Unique hardware support?

→ Pass-through features instead of virtualization?

Portability across hardware standards

Are standards feasible?

→ Hypervisor Project group

B) Display Sharing

One physical display can be shared across multiple operating systems.

A HW compositor unit composites final display buffer
from HW Layers of each OS.

This requires virtualization of the display controller hardware.

Hardware Layers

(contrast or complement: GENIVI Layer Management)

C) Surface Sharing

Operating systems exchange graphical (bitmap) content

GPU rendering first, then transfer

Receiving system has flexibility to use this content (compositing)

In some cases, the compositor API is made available remotely, e.g. Wayland → Waltham

Typical: Difference encoding (i.e. “video stream”)

C-2) Virtual Display (surface sharing)

The project considers this a sub-category to Surface Sharing

Full display transfer, (by encoding as a video stream)

- Displayed as full screen (layer)
  or treated as a surface in composite HMI.

Often characterized by a “transparent” API such that applications can use it as if it were a real display, but still identified as a separate object type – Example: See Android

D) API Remoting

Transfer API calls, corresponding to "drawing commands", or other abstract graphics representation from one ECU to another.

Commands or scene representation to be executed on the GPU of the receiving ECU.

→

- “Remoting” existing APIs – (note the GPRO project for protocol evaluation)
- Create custom API for the task

D) API Remoting

Bandwidth efficient?

“It depends...”

E) Shared State – Independent Rendering

Each system has independent graphics systems and bitmap information.
The systems only synchronize their internal state and exchange abstract data.

Based on this shared data, each system independently render graphics to make it appear like they are showing the same or related graphics.

Example: An appearance of synchronized map rendering could be achieved by drawing maps independently, and exchanging only the GPS position, scale of map, etc.

E) Shared State – Independent Rendering

Consequences...

Each participating system needs its own basic graphics (bitmaps, textures)

Data and rules for HMI look & feel must be aligned beforehand,

Software updates – required on both sides if look&feel changes

Navigation example: Both must have map data.

Advantages:

- The state/data transfer could be very small and bandwidth efficient

3. Finding Examples

Finding Examples

- Building the required knowledge basis
- The “state of the industry” in this area
- Evaluate and encourage FOSS licensed code for shared basic technology
- Identify gaps – start collaborative development
- Identifying non-open-source implementations
  - – facilitate market use (make them interoperable)
  - – encourage shared implementation efforts

Surface Sharing example: Wayland, Waltham

- Wayland is a display server protocol for Linux
- Wayland defines how applications communicate their graphical content (surfaces) to a compositor that assembles (multiple) applications' combined appearance into the complete graphical screen output.
- On most computers this is a local communication between apps and system.
- Waltham enables Wayland to work over a network. (very simplified*) Since Wayland juggles "surfaces" - Waltham becomes an example of surface sharing

- **GSHA project wiki contains an analysis with much more depth*

Surface Sharing examples: Phone “projection”

- Apple Carplay
- Android Auto
- (SDL)

Surface Sharing

- Other examples exist – probably numerous
- Proprietary HMI systems in particular
- Wayland is very Linuxy… can Waltham / Wayland protocol over network become cross-platform standard?
  - GSHA topic: Study and compare to Android APIs
- Surface sharing with QNX? – See later case study

API Remoting Examples

- RAMSES
- Smart Device Link

4. Case Studies

Case Studies

- **Qt Remote Objects** (Shared State)
- **Qt WebGL** (API Remoting)
- **Qt WebAssembly** (API Remoting)
- **Implementing Waltham in practice : ADIT/Bosch**
  (Surface Sharing)
- **Android/Linux Navigation interaction HMI**
  (Shared State, Independent Rendering)
- **Android/QNX surface exchange : Harman**
  (Surface Sharing)
- **The Canvas-demo : Renesas**
  (Display Sharing, GPU sharing)
- **AllGo Multiple-display demo : AllGo**
  (Virtual Display)
- **RAMSES : BMW**
  (API Remoting)

5. Methods of Analysis

Methods of Analysis

Don’t miss the presentation tomorrow on How to choose!

- System architecture / network connections / bandwidth
- GPU availability and load + CPU load
- Internal memory bandwidth
- Implementation complexity
- Safety level
- Supplier preference

...

6. Reference Designs & Blueprints

Reference Designs & Blueprints

- Look at common Use-cases
- Can we recommend not only solutions but software/system architectures?

7. General Graphics Technologies

General Graphics Technologies

- Building the required knowledge basis
- Graphics (also non-distributed) APIs matter!
  - Easily remotable?
  - Implementations available on selected OS combinations?
- Therefore: **Collaboration with Khronos Group**
- Presentation on **Vulkan** tomorrow (HV track)!

8. Documents and Communication

Documents and Communication

- After investigation, analysis, surveys…

  Consolidate!

  Clarify!

  Communicate!

- Inform about automotive industry consensus
- Guidance to buyers and sellers of solutions

Documents and Communication

- Multiple Tech Briefs ***published*** on each of the categories + case study
- Whitepaper consolidating all info plus deeper analysis
  - Draft status

Outlook and roadmap

Outlook and Roadmap

- Finalize Graphics Sharing whitepaper
  (draft is available – please review)
- Graphics exchange with Android
  – from project/ad-hoc to agreed upon ways
- Further investigate, promote, encourage and drive forward open implementations
- Standard approaches – recommended APIs – Interoperability

THANKS FOR LISTENING!
JOIN OUR EFFORTS!

Visit GENIVI:

http://www.genivi.org

http://projects.genivi.org

Contact us:

help@genivi.org