DISPLAY CLUSTER POTENTIAL SOLUTIONS.
Rendering on one ECU and transfer via video to another ECU (e.g. APIX2 or H.264)

(E.g. Display of a navigation map from HeadUnit inside instrument cluster)

- Needs very high permanent bandwith for good quality on the network, and maybe also a dedicated link between the ECUs (Hardware costs)

- Can have compression artefacts

- Computation power is calculated in the sending ECU, even if the receiving ECU is not build in the car

- No graphical interaction between local and remote content possible (e.g. Reflections, Shadows, etc.)
Connecting all displays to one ECU
(E.g. Connecting main display and instrument cluster display to one powerful ECU)

- Would solve distribution challenge, but we still have to exchange content between local processes

- May be suitable for HeadUnit and instrument cluster, but does not scale very well for higher amount of displays

- Computation power have to be calculated in this one ECU for all possible combinations
Usage of a dedicated protocol per UseCase
(E.g. Display of entertainment lists inside the instrument cluster)

- Development effort for each use case
- Hard to change or extend after the protocol is defined, especially at late phase of the development phase
Stream of OpenGL commands over the wire
(glGenBuffers, glBindBuffer, glBufferData, glDrawElements, etc)

- Description for each single frame has to be transferred. Therefore a lot of duplicate information is communicated.
- High amount of bandwidth needed for 60fps with about 1000 draw calls.
- No graphical interaction possible, because the command stream is rendered into a surface and just integrated as is, into the remote scene.
- No compression artefacts.
THE RAMSES WAY.

- Distribution of graphical content on a logical SceneGraph level
- Content provider (Application) and Content Consumer (Renderer) are logically split, and can run on different ECUs (decoupling)
- Updates or late changes in the development cycle on application side possible without changes on the rendering side
- Graphical interaction possible between scenes from different ECUs
- Animated transitions for content from one screen to another
- Several times less bandwidth needed during runtime, compared to video distribution
ARCHITECTURE.
DISPLAY CLUSTER.
ARCHITECTURE GOALS.

– Distributed rendering of arbitrary content by providing a consistent look and feel of same content classes
– Flexible, extensible use case independent architecture / protocol
– Handling different variants (Low/High, with HUD/without HUD)
– Independent of deployment and ECU partitioning
– Split application logic from display location
– One render engine for all displays
HMI
application
Widgets

Scene graph
Zero copy local communication

SomeIP

Renderer

OpenGL ES

OS/Driver

GPU

Navigation

DB Lookup

RAMSES
COMPONENTS.
HIGH-LEVEL FEATURES.

Client
- C++ Client API
- Text Rendering
- File asset

Renderer
- C++ Renderer API
- OpenGL ES 3.1 Rendering
- Embedded Wayland Compositor
- Window abstraction

Framework
- Scene mapping/linking
- Embedded Wayland Compositor
- Post processing

Scene graph provider / consumer
- Resource provider /
- Platform abstraction
- Logging

SOME/IP services / communication
- Process-internal communication
- Plain TCP based communication

Stack adapter
- Platform dependent SOME/IP stacks
FEATURES.
C++ CLIENT API.

- Standard data structures:
  - Scene
  - Node, TransformationNode, GroupNode, MeshNode,TextNode
  - Appearance, Shaders (glslang input parsing), TextureSampler, Texture2D/3D/Cube, Index/Vertex array inputs
  - Camera
  - RenderGroup
  - RenderPass
  - RenderTarget
- Defining static animations
TEXT/FONT RENDERING.

- Support for multiple font rendering engines
  - Freetype
  - RT-Font
- Harfbuzz based font shaping support e.g. for Thai rendering
- Font cascades
- Configuration of text style attributes (height, ascender, descender, etc.)
ASSETS.
LOADING SCENE AND RESOURCES FROM FILE.

- Store and load scene and resources in a RAMSES specific format
- Fast loading
- Startup requirements, especially needed e.g. for the Instrument Cluster
SOME/IP BASED COMMUNICATION.

- Standard communication protocol for all Ethernet based ECUs in our cars.
- Currently different SomeIP Stacks with different toolchains and APIs are used on different ECUs.
RENDERER C++ API.
WINDOW AND DISPLAY CONFIGURATION.

Window support for WGL, EGL, X11, Integrity

Display configuration such as
- Display Size
- Viewport settings
- Camera parameters
- Stereo rendering support
- Warping support
- Multi-sampling
RENDERER C++ API.
SCENE MAPPING.

Map scene to a buffer

- Offscreen render buffer
- Frame buffer
RENDERER C++ API.
DATA LINKING.

Interaction between scenes

Data links:
- Transformation linking
- Data object linking
- Texture linking
EMBEDDED WAYLAND COMPOSITOR.

Renderer implements wayland-based compositor

Client \(\xrightarrow{\text{textureSampler with streamTexture (ref ivi surface id)}}\) Renderer \(\xleftarrow{\text{Wayland wayland-ivi protocol (surface id)}}\) Wayland application
TECH FACTS.

- RAMSES provides a SDK
- Graphical tooling available (RAMSES studio) to create scenes
- C++ API, SomeIP-based serialization for IPC
- Supported platforms:
  - Windows
  - Linux
  - Integrity
  - Android (planned)
- OpenGL ES 3.1 support