

What's New in GStreamer D3D12

Seungha Yang

What's New in GStreamer D3D12

- Updates to Direct3D12 backend
- New Rust binding support
- New D3D12 elements
- Improved upload/download performance

Rust Binding Support

- New Rust binding for the GStreamer D3D12 backend
- Built on Microsoft's official **windows** crate
- D3D12's ref-counted object model aligns safely with Rust's strict lifetime rules
- Provides a safe and idiomatic Rust interface to D3D12

New and Enhanced D3D12 Elements

Introduces new and enhanced D3D12 elements

d3d12overlaycompositor

- Blends GstVideoOverlayCompositionMeta on GPU
- Avoids copying GPU frames to system memory for blending

d3d12interlace / d3d12deinterlace

- Converts between progressive and interlaced formats
- GPU-based counterparts to software interlace / deinterlace
- **d3d12deinterlace**: Compute-shader implementation
 - Independent of vendor-implemented ID3D12VideoProcessor
 - NOTE: d3d11deinterlace element is based on ID3D11VideoProcessor
 - Consistent quality and capabilities across hardware

d3d12remap / d3d12fisheyedewarp

- **d3d12remap**: Arbitrary UV mapping with user-supplied LUT
 - Similar to OpenCV's remap()
- **d3d12fisheyedewarp**: Fisheye correction projection mapping
 - Specialized version of d3d12remap
 - Generates LUT internally from user-defined parameters
- LUT-based design ensures consistent and real-time mapping

Existing Element Updates

- d3d12swapchainsink: Added UV-remap action signal
 - Enables interactive LUT updates and dynamic view transformation even while paused
 - e.g., pass a fisheye dewarp LUT via signal for real-time projection changes
- Added GstColorBalance interface support
 - Supported by d3d12videosink / d3d12swapchainsink / d3d12convert
 - Real-time control of hue, brightness, contrast and saturation on GPU

Existing Element Updates

- Added HDR screen capture support in `d3d12screencapturesrc`
- Windows HDR mode uses FP16 scRGB color space internally
 - DXGI Desktop Duplication API gives incorrect colors in 8-bits RGBA format mode
- Output FP16 scRGB as-is lossless but tricky
 - 16bits floating point format is not defined in GStreamer
 - Neither scRGB – scRGB white point is display-referred
- New tone-mapping path converts scRGB → sRGB for proper SDR output

Performance Improvements

Upload and download redesign

Why Upload / Download Matters

- Simple GPU pipeline can stay fully on GPU
 - Upload / download not always critical
- Real applications often mix software elements
 - Frequent data transfer between system and GPU memory
- Large-scale buffering cannot always fit in GPU memory
 - Upload / download becomes unavoidable
- These are often the slowest path in GPU pipelines
 - Optimizing them is essential for stable performance

Previous Limitations

- Old design inherited from GstD3D11
- Three key problems in upload / download path
 - Extra copy: every frame went through system $\leftrightarrow$ staging $\leftrightarrow$ GPU
 - Adding unnecessary memory traffic and latency
 - No batched copy: each texture handled its transfer separately
 - e.g., 3 staging $\rightarrow$ GPU copy commands per frame in case of I420 format
 - Creating many small copy commands instead of one efficient batch
 - No async copy
 - Upload ran on the 3D queue $\rightarrow$ serialized with 3D rendering
 - Download blocked the CPU thread waiting for GPU fence

The New Design

- Introduced GstD3D12StagingMemory
 - Removed system ↔ staging copy
 - Supports batched and async transfers
- Transfers handled inside d3d12upload / d3d12download
- **queue-type** property lets users pick copy or compute queue
 - Allows task overlap instead of blocking 3D queue
- Although some elements (e.g., d3d12videosink) can upload internally, using an explicit d3d12upload is recommended

Summary

- Rust binding built on Microsoft's official Windows crate
- New D3D12 elements: overlay, (de)interlace and fisheye remapping
- Extended existing elements with new capabilities
- Redesigned upload / download
 - Removed extra copy
 - Adds batched and async transfer
 - Enables queue control for better parallelism

Questions?