Video on Dope

How to Render Video with OpenGL

Matthias Hopf

SUSE R&D / Novell

Properties of Video

- Motivation
  - Fast display of live video data
  - Data typically not RGB, with subsampled chrominance: Most codecs work in $Y'C_bC_r$ ($Y'P_bP_r$ / YUV) color space
  - (Direct copy for framegrabber cards)
  - Processor load
  - Bandwidth
    - ARGB: 100%   YUY2/UYVY: 50%   YV12: 37.5%
    - Limiting factor for HDTV for many chipsets
- Sensitive to
  - Scaling
  - Color conversion, gamma
  - Tearing, jitter

Novell

Properties of $Y'C_bC_r$

- Resolution of the eye is higher for luminance
  - Encode luminance independent from chrominance
  - Subsample chrominance
- Color space conversion resides in a FUD cloud
  - YUV is PAL color space, subtly different to $Y'C_bC_r$
  - Still $Y'C_bC_r$ is called YUV...
  - FourCCs YUY2, UYVY, YV12 all are $Y'C_bC_r$
  - Analog values 0-1, two quantization schemes common, for MPEG1/2, H.263, H2.64, VC-1 only one used
  - Conversion to R'G'B' only possible if RGB color space is well defined
  - Ignored almost everywhere (NTSC / PAL conversions)

SUSE A NOVELL COMPANY
Novell

Properties of $Y'C_bC_r$ (2)

- Quantized conversion scheme (sRGB?)

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.164 & & 1.596 \\ 1.164 & -0.391 & -0.813 \\ 1.164 & 2.018 & \end{pmatrix} \cdot \begin{pmatrix} Y-16 \\ Cb-128 \\ Cr-128 \end{pmatrix}$$

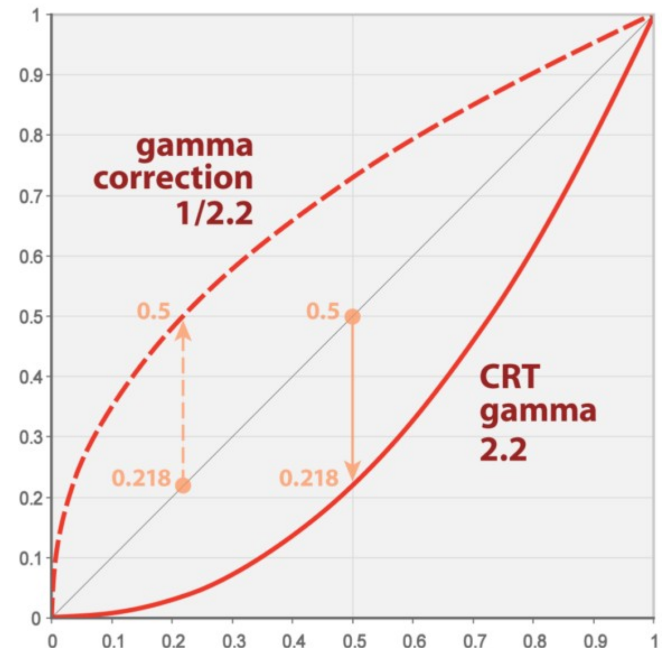
- Y є [16,235], Cb,Cr є [16,240]
- $Y'C_bC_r$ is nonlinear (gamma corrected)
- RGB is in fact nonlinear R'G'B' (gamma corrected)
- Conversion to/from R'G'B' is linear
  → Interpolation and color conversion can be swapped
- $Y'P_bP_r$ is analog $Y'C_bC_r$ (also misused for interlaced)

Novell

What the Gamma?

- Out = In^{γ}
- Sometimes γ, sometimes $1/\gamma$ ...
- CRTs had (inherent) gamma of 2.2
- Applications, Web, etc. adapted to that (try xgamma -gamma 0.45), also close to eye sensitivity
  → even LCDs have this gamma!
- sRGB is **almost** gamma 2.2
- Output drivers of video cards are linear (for PCs, that is)
- Often inaccurate, especially for dark regions

→ Need gamma correction for monitors
No correction needed for (good) video projectors



Novell

“Legacy” Video Rendering Engines

- X11 / MITShm
  - Only RGB
  - No hardware scaling

- XVideo / Color keying
  - Doesn't work with Composite
  - Often doesn't work with secondary output
  - Often few or only single frame
  - Often limited frame size
  - Inflexible color conversion, no gamma correction, ...

Novell.

Current Video Rendering Engines

- XVideo / Blitter
  - Still often limited
  - Still inflexible, no gamma correction, ...
- Native OpenGL
  - First fragment program based approach in xine
  - Many different algorithms in mplayer
  - Can't use direct rendering with composition managers (yet) – except with newest NVIDIA drivers

New Video Rendering Engines

- XVideo / Xgl
  - Uses OpenGL fragment programs
  - Needs pBuffers or FBOs
  - No support for that in open source drivers (yet)
  - One data copy due to inflexible abstraction in glitz
- compiz
  - Send images to Xserver, but let compiz do the rendering with OpenGL
  - Most flexible approach
  - Allows for distorted video, video shadows, etc., ...
  - Still to be implemented

SUSE A NOVELL COMPANY
Novell.

Rendering with OpenGL

- Issue 1: Color conversion
  - Needs memory bandwidth
  - AGP / PCIe bandwidth an even scarcer resource
  - Only linear operations except for gamma correction
- Issue 2: Texture upload
  - If data doesn't reside in AGP memory, it has to be copied
  - EXT_pixel_buffer_object often only accelerates RGBA
- Issue 3: Scaling
  - Possibly the simplest thing in OpenGL
  - Better filters need more sophisticated renderers

Novell

SUSE A NOVELL COMPANY

Color (1): Software or YUV Textures

- Software: Same as for X11 / MITShm
  - Without embedded scaling
  - Simple code, pretty fast with MMX + SSE2
  - No reading from destination pointer, can be AGP memory
  - Fast upload of RGBA textures from AGP to Gfx memory

- 1st OpenGL try: Native YUV texture formats
  - MESA_ycbcr_texture, APPLE_ycbcr_422, EXT_422_pixels, SGIX_ycrcb
  - Subtle differences
  - Not widely supported (read: not at all)
  - No planar YUV format
- Best forget about it

Novell.

Color (2): Approach for Legacy Cards

- Register combiners for old NVIDIA cards
  - Reconfigurable hardware (nv15 and up)
  - Rather complex to program
  - Only linear conversions, inexact
  - Due to negative values two combiner units needed
- “Fragment shader” for old ATI cards
  - Despite the name actually reconfigurable hardware
  - Program language, but extremely limited,
    esp. on dependent texture lookups
  - Only linear conversions, inexact

Novell.

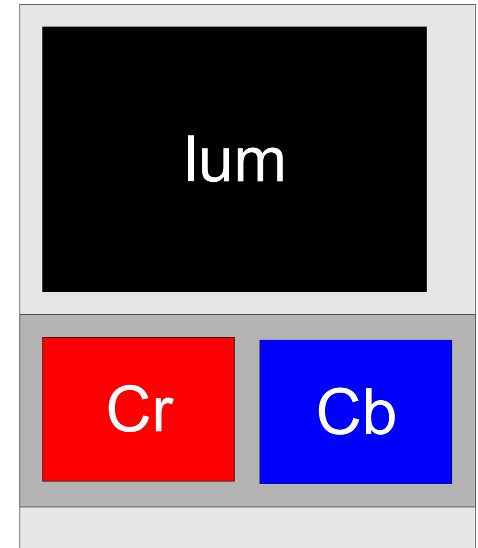
Color (3): Fragment Programs

- Use programmable pixel engine
- Probably the most flexible approach
- Exact conversion
- Probably the only way for YUY2
- Needs modern graphics hardware
  - ATI r300 or better
  - Intel 915 or better
  - NVIDIA nv30 or better
- Contrast, saturation, brightness control color matrix
  - No extra instructions needed
- Gamma correction (exp) slow on first generation cards
  - Use lookup table in this case

SUSE A NOVELL COMPANY
Novell.

YV12 – Planar Layout 4:2:0

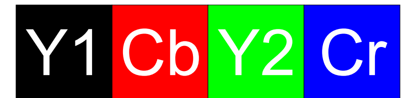
- Store in luminance texture
- Border at boundaries for interpolation
  - 0 is 16 in Lum, 128 in Cr/Cb ;-)
- Use bilinear interpolation in lookup
- Downside: luminance textures, no accelerated texture download




```
ADD u, tex, off.xwww;
TEX color, u, texture[0], 2D;
MUL v, tex, .5;
ADD u, v, off.xyww;
ADD v, v, off.zyww;
TEX tmp.x, u, texture[0], 2D;
MAD color, color, 1.164, -0.073;  /* -1.164*16/255 */
TEX tmp.y, v, texture[0], 2D;
SUB tmp, tmp, { .5, .5 };
MAD color.xyz, { 1.596, -.813, 0 }, tmp.xxxw, color;
MAD color.xyz, { 0, -.391, 2.018 }, tmp.yyyw, color;
```


Novell

YUY2 – Interleaved Layout 4:2:2



- Store 2 pixels in 1 ARGB texel
- Interpolate manually in fragment program
  - Even/odd pixel case dependency for inter-texel vs. intra-texel interpolation
- Higher program complexity
- Used less often (w32codecs)

- Not yet implemented

Novell.

Color (4): 3D Texture Lookup

- Interprete Y, C_b, C_r as texture coordinates
  - Look up RGB by a 3D dependent texture lookup
  - Due to trilinear interpolation no 256^3 texture necessary
  - A 2^3 texture would suffice, but negative values would be necessary, as the Y'$C_b C_r$ coordinate range is partially outside its color space
  - De-fact size needed is 32^3 which still fits in the cache
  - Needs fragment programs + hardware 3D textures

SUSE A NOVELL COMPANY
Novell.

Scaling (1): Bilinear

- Defined standard implementation for OpenGL
- Invoked by rendering a textured quad
  - Single fragment program instruction
- Not much else to do here :-)

- Exception: YUY2
  - Interpolation has to be hand-coded for luminance

Novell.

Scaling (2): Bicubic

- Trivial implementation would need 16(!) texture lookups per color component(!)
- More involved implementation
  - Uses dependent bilinear texture interpolation by intelligent texture coordinate calculation
  - 2 1D texture lookups for filter coefficients lookup
  - Only 4 2D texture lookups per color component
  - Typically it is enough to do this for luminance, and use a bilinear filter for chrominance values

SUSE A NOVELL COMPANY
Novell

Future Smoking...

- Fragment shaders allow for even more possibilities
- Remove tearing
  - Double buffering only possible for full screen rendering
  - Otherwise wait for vertical sync before blitting
  - Needs support from the drivers
  - Still being worked on
- Add noise
  - What?!?!?!
  - Actually improves visual quality
  - Reduce quantization from $Y'C_bC_r$ conversion (dithering)
  - Hides blocking artifacts, adds film grain effect
  - Easy to implement (one additional texture lookup)
  - Difficult to get right (which type of noise)

Future Smoking (2)...

- Adaptive deinterlacing
  - Fragment program needs current and last frame
  - Weighted average of both frames and adjacent pixels, according to interlace probability and motion estimation
- Remove jitter:
  motion compensated time-based interpolation
  - Show true moving 60fps from 24fps video
  - Extremely demanding (optical flow calculation, position dependent texture interpolation, hole filling)
  - Needs optical flow per pixel, not per block like video encoders do...
  - Certainly not possible for HDTV with current hardware
- Allow for custom program fragments (compiz)
  - E.g. MPEG deblocking

SUSE A NOVELL COMPANY
Novell

Pfff...

- Questions ?
- Comments ?
- (Short) demo

Novell.