Xgl

Matthias Hopf
SuSE Labs

Novell

What is Xgl?

- Xgl is:

- Xserver using OpenGL for its drawing operations
  - No more hardware drivers inside
  - 3D hardware is faster
  - Render can be accelerated independent of hardware
  - Future hardware won't have 2D core

- Xgl is ***not***:

- A display mechanism using different protocol or API
  - Still X11 on client side
- Accelerating OpenGL programs
  - Currently slower (indirection, composition manager)

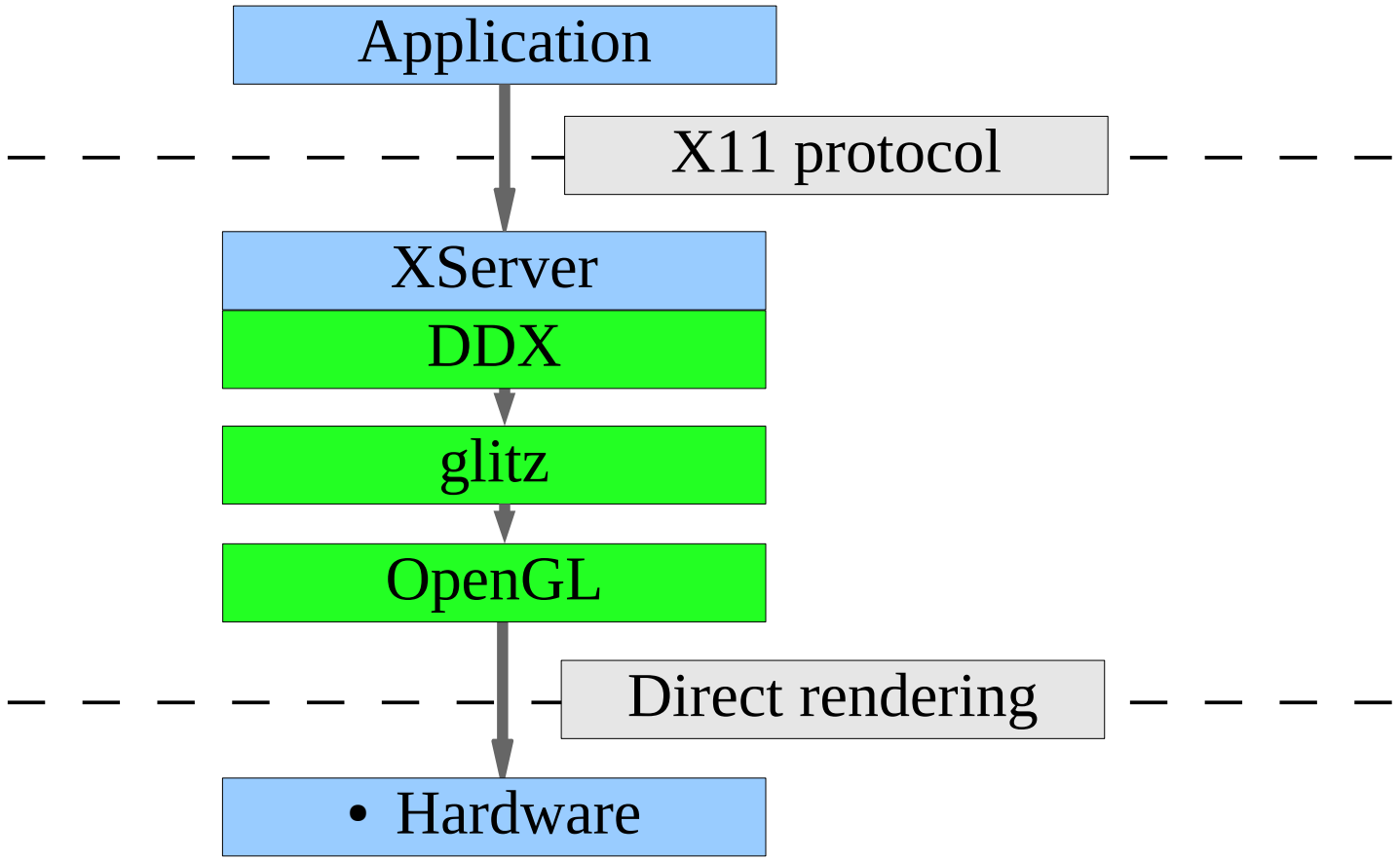
Flavors

- Xgl
  - Frontend that loads backend plugins
  - Dynamic loading to avoid name space collision
  - Currently based on KDrive Xserver, inclusion in Xorg after modularization
  - Using glitz as mid-level abstraction of OpenGL
- Xglx
  - Similar to Xnest, run in a window
  - Mostly worked on ATM
- Xegl
  - Standalone

Hardware Drivers

- libGL.so provides output drivers for Xgl
- Extensions needed for
  - Mode setting (EGL_MESA_screen_surface + others TBD)
  - Offscreen rendering (pbuffers + frame buffer objects)
  - Filtering + color conversion (pixel shader)
  - Hardware cursors (TBD)
  - Address space sharing (TBD):
    needed for running direct rendering applications
- Input
  - Under discussion
  - Most likely to use future Xorg mechanisms

Big Picture


Application
X11 protocol
XServer
DDX
glitz
OpenGL
Direct rendering
Hardware

glitz

- OpenGL acceleration for Cairo
  - Abstraction of pixel formats, buffers, geometry, clipping, filters, operators
- Accelerates almost everything of Render extension
- Using textures for Pixmaps
- No software fall-backs

DDX

- Software fall-backs for everything not accelerated
- Accelerated so far:
  - Core: solid and tiled fills, copy area, core fonts
  - Render: trapezoids, glyphs, convolution filters, ...
  - GLX: 1.2
- Unlikely / difficult to be accelerated:
  - Core: Logical ops, plane masks
  - Render: PictOpSaturate
- Only indirect rendering for OpenGL applications
  - Off-screen framebuffer has to be in same address space as textures for composition manager
  - Seamless integration with Composite

Compositing

- Rendering to off-screen framebuffer per window
- Composition manager renders these on screen
  - -> Effects like transparency etc.
- Windows in 3D space
  - -> Composition manager using OpenGL: glxcompmgr
- Bind Pixmaps to textures:
  GLX_MESA_render_texture + indirect rendering
- Currently only supported in Xgl
- Pixmaps stay in graphics memory all the time
- Small issues with GLX 1.3
- (demo)

Issues

- More acceleration
- Texture storage format is static, pixmaps are not
  - Only one picture format per depth
- Source picture clipping
- Clamp to transparent for pictures without alpha
- Direct rendering support
- Memory management
- Most important: drivers quality

Near to Mid-term Future

- Move Xgl into X.org
- Xegl improvements
- Stabilization + more acceleration
- Integration into desktops (delayed)
- Getting vendors to support EGL_MESA_screen_surface
- Work on other extension specifications
- First attempts to think about scene graph alike API

XVideo

- Motivation
  - Fast display of live video data
- Building blocks
  - MITShm PutImage with YCbCr color space
  - Hardware assisted Image copy (from Video cards)
  - Properties (brightness, contrast, gamma, ...)
- Two different implementations
  - PutImage w/ color space conversion
  - Color keying + additional scan converter

Why Care About XVideo

- Processor load
  - Direct copy for TV
  - Most codecs work in YCbCr color space
  - Good upscaling algorithms
- Bandwidth
  - ARGB: 100% YUY2/UYVY: 50% YV12: 37.5%
  - Still limiting factor for HDTV for many chipsets, even nowadays

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

Advantages and Drawbacks in XAA/KAA

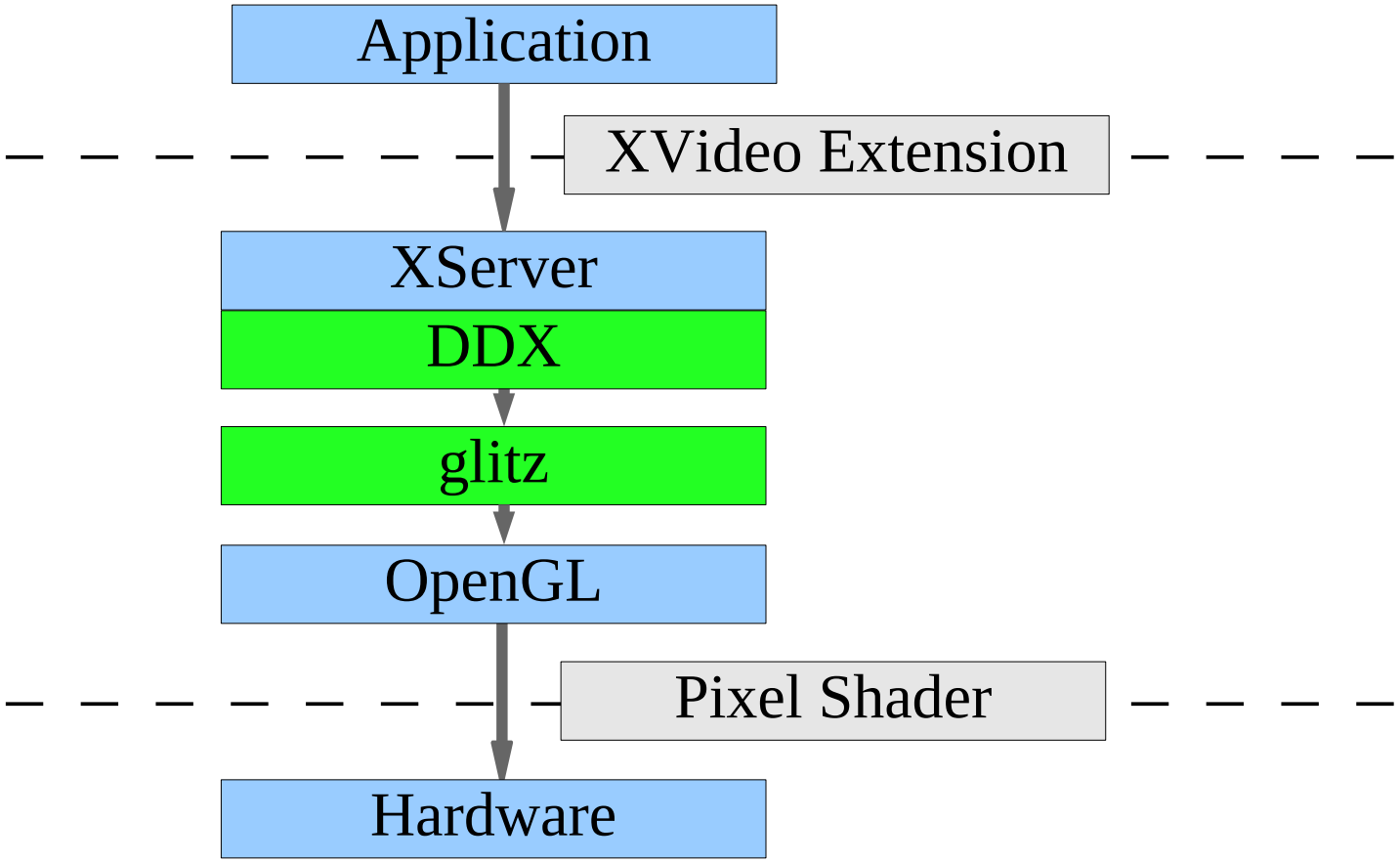
+ Rather easy to add in hardware
+ Fast and resource friendly

- Sometimes drivers don't get it right
- Limited in size (often < HDTV i.e. 1920x1080)
- Color keying considered broken
- Small number of simultaneous streams (color keying: 1?)
- Often does not work on secondary output
- Color keying doesn't work with composition managers
- Attributes dependent on hardware + driver

Approach in Xgl

- 1st try: Native YUV texture formats
  - MESA_ycbcr_texture, APPLE_ycbcr_422, EXT_422_pixels, SGIX_ycrcb
  - Subtle differences
  - Not widely supported
  - No planar YUV format
- 2nd try: Use fragment program for conversion to RGB

Big Picture for XVideo in Xgl



The DDX

- PutImage:
  - Calc clip region, set geometry
  - Re-allocate YV12 surface if needed, if that fails, RGB
  - Upload YV12 pixel data to surface
  - Render scaled surface to screen
-

```
glitz_buffer_set_data (portPriv->buffer, 0, portPriv->size, data);
glitz_set_pixels (src, 0, 0, width, height, &pixformat, portPriv->buffer);
transform.matrix[0][0] = FIXED_ONE * src_w / (float) drw_w + .5;
transform.matrix[1][1] = FIXED_ONE * src_h / (float) drw_h + .5;
glitz_surface_set_transform (src, &transform);
glitz_composite (GLITZ_OPERATOR_SRC, src, NULL, dst, src_x, src_y, 0, 0,
                 drw_x + xOff + pDraw->x, drw_y + yOff + pDraw->y, drw_w,
drw_h);
```

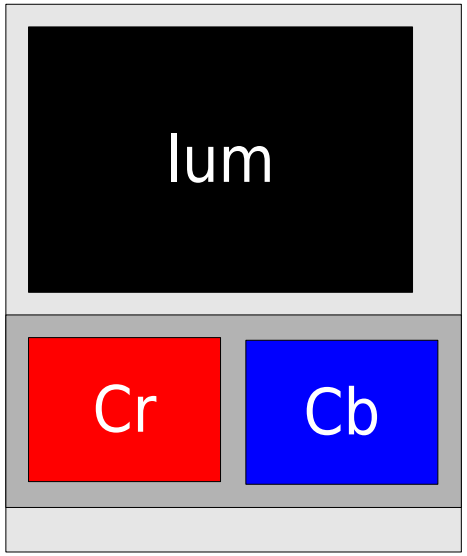
Extensions to Glitz

- Add color space description: fourcc
  - So far: only RGB, now additionally YV12 + YUY2
  - RGB color space continues to use bit field descriptions
  - glitz_color_format_t, glitz_pixel_format_t, glitz_texture_t
- YV12 + YUY2 pixel formats
- YV12 + YUY2 surface formats
- Texture formats for these
- Support for texture only top-down scanline order
- Software based color conversion
- Fragment programs for color conversion on-the-fly

YV12 – Planar Layout 4:2:0

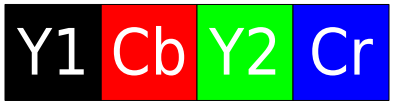
- Color subsampled 2:1 in both x and y
- Store in luminance texture
- Border at boundaries for interpolation

  0 is 16 in Lum, 128 in Cr/Cb ;-)
- Use bilinear interpolation in lookup




```
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;
```

YUY2 – Interleaved Layout 4:2:2



- Color subsampled 2:1 in x only
- Store 2 pixels in 1 ARGB texel
- Interpolate manually in fragment program
- Higher program complexity
- Used less often

- Not yet implemented

Not Yet Implemented / ToDo

- Border initialization code
- HW capabilities auto detection & software fallback
- YUY2 / UYVY image formats
- Properties (luminance, contrast, saturation, gamma)
- Higher order filters
- Fragment program optimization
- Register combiner path for older cards
- Maximum texture size / tiling

Open Complex Issues

- Color keying? Video ports?
  - Color keying possible (complex) – but useful?
- Fragment program linking (YV12 w/ Gaussian?)
  - OpenGL 2.0?
- Vertex attribute allocation
  - Right now: 1 texture = 1 set of texture coordinates
- Right now: YCbCr surfaces only intermediate
  - One copy w/ composition manager could be saved
  - Doesn't work with RGB overdraw & partial windows
- More filters: Deinterlacing
  - Belongs to application domain?