FOSDEM 2020

Back to the Linux Framebuffer!

LINUX-FBDEV

Linux Framebuffer support in free software

Nicolas Caramelli

Contents

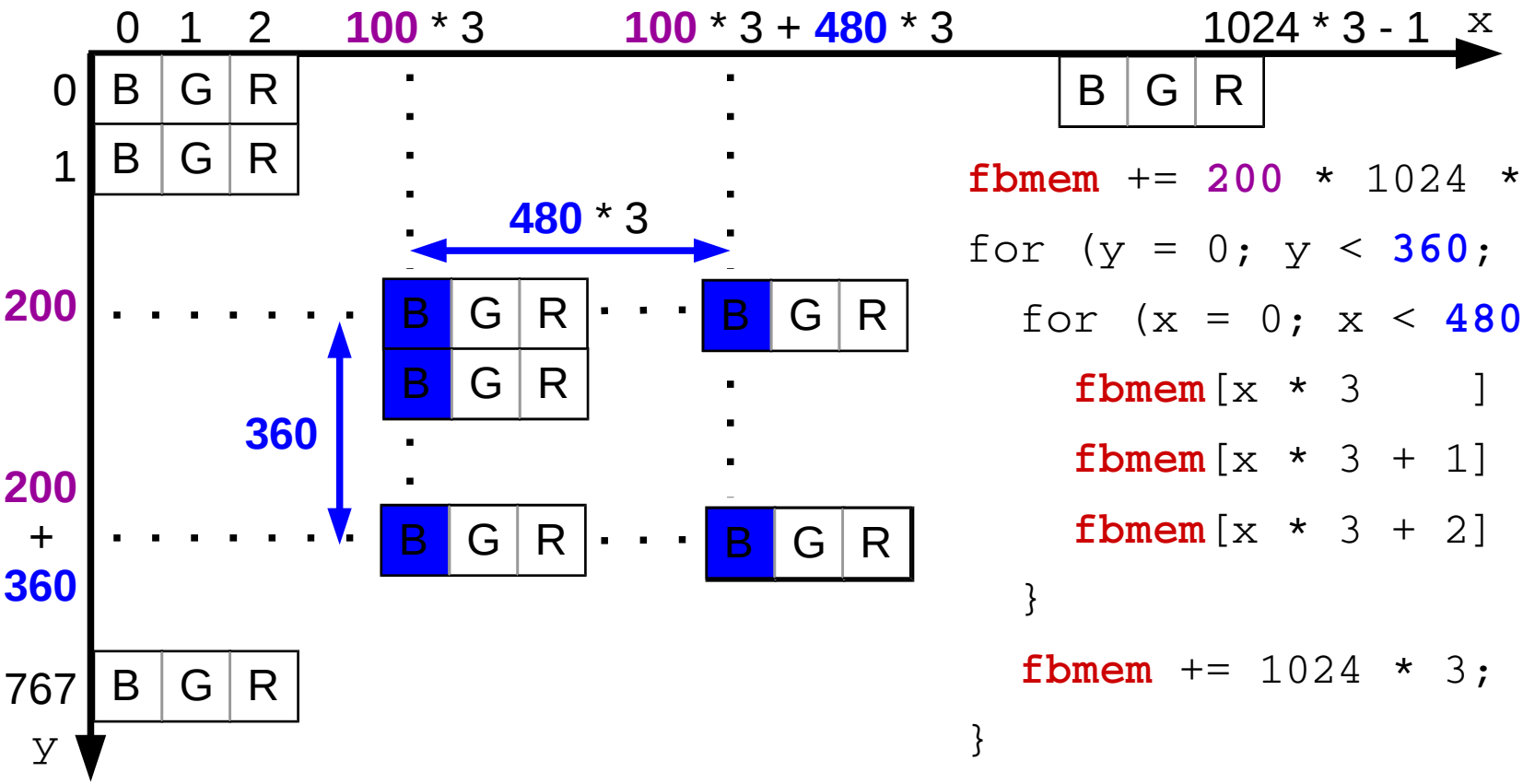
Contents

Contents

/dev/fb0 and mmap

- First check → cat /dev/urandom > /dev/fb0
- Example with 3 bytes (blue, green and red) per pixel on 1024 x 768 screen resolution
  → Framebuffer memory = 1024 * 768 pixels * 3 bytes

```
fd = open("/dev/fb0", O_RDWR);
fbmem = mmap(NULL, 1024 * 768 * 3, PROT_WRITE, MAP_SHARED, fd, 0);
```



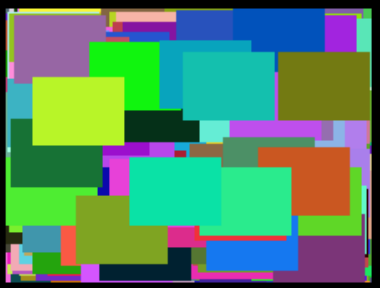

```
fbmem += 200 * 1024 * 3 + 100 * 3;
for (y = 0; y < 360; y++) {
  for (x = 0; x < 480; x++) {
    fbmem[x * 3    ] = 255;
    fbmem[x * 3 + 1] = 0;
    fbmem[x * 3 + 2] = 0;
  }
  fbmem += 1024 * 3;
}
```

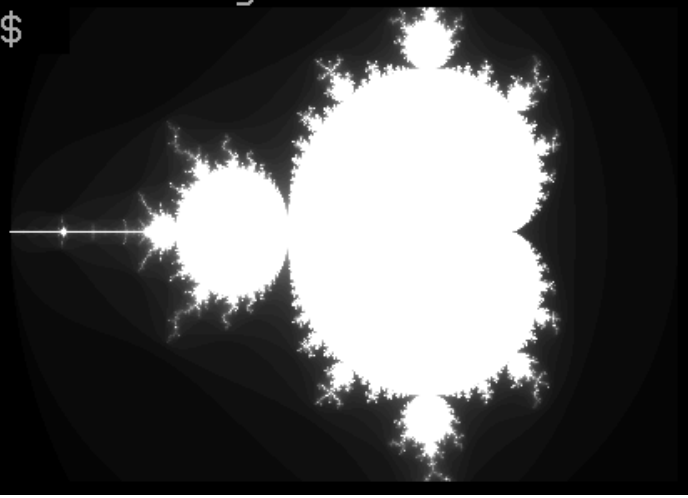
/dev/fb0 and mmap demo

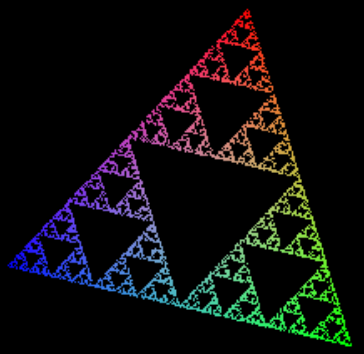
fb-test-app, fbmark

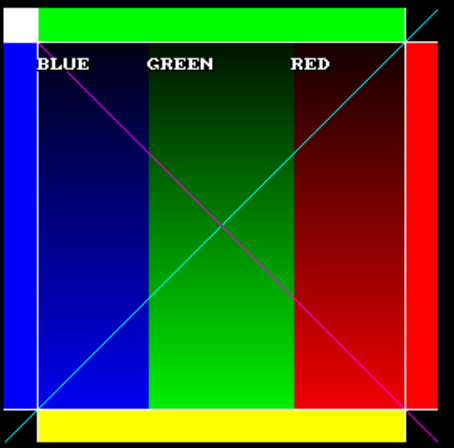
```
~$ fb-test -o 260x260+740+480
fb res 1024x768 virtual 1024x768, line_len 3072, bpp 24
~$ fb-string 820 40 "Linux Framebuffer" 0xff 0
fb res 1024x768 virtual 1024x768, line_len 3072, bpp 24
~$ WIDTH=260 HEIGHT=260 POSX=440 POSY=480 fb_sierpinski
Sierpinski frame buffer test bench

Benchmarking     1024 iterations:  7237.46 Frames/second
Benchmarking     2048 iterations:  4239.95 Frames/second
Benchmarking     4096 iterations:  2325.17 Frames/second

~$ WIDTH=400 HEIGHT=300 POSX=20 POSY=460 fb_mandelbrot
Mandelbrot frame buffer test bench
Benchmarking 48 iterations: 0.43 seconds
~$ WIDTH=220 HEIGHT=165 POSX=800 POSY=220 fb_rectangle
Rectangle frame buffer test bench
Benchmarking 55x41 size: 126.30 MPixels/second
~$
```

Linux Framebuffer









fb-test-app, fbmark

- fb-test-app test suite

  https://gitlab.com/meetroger/fb-test-app

  fb-string.c, fb-test.c

  → `fb_open()` in *common.c*

  → **fbmem = mmap()** on **/dev/fb0**

- fbmark benchmarks

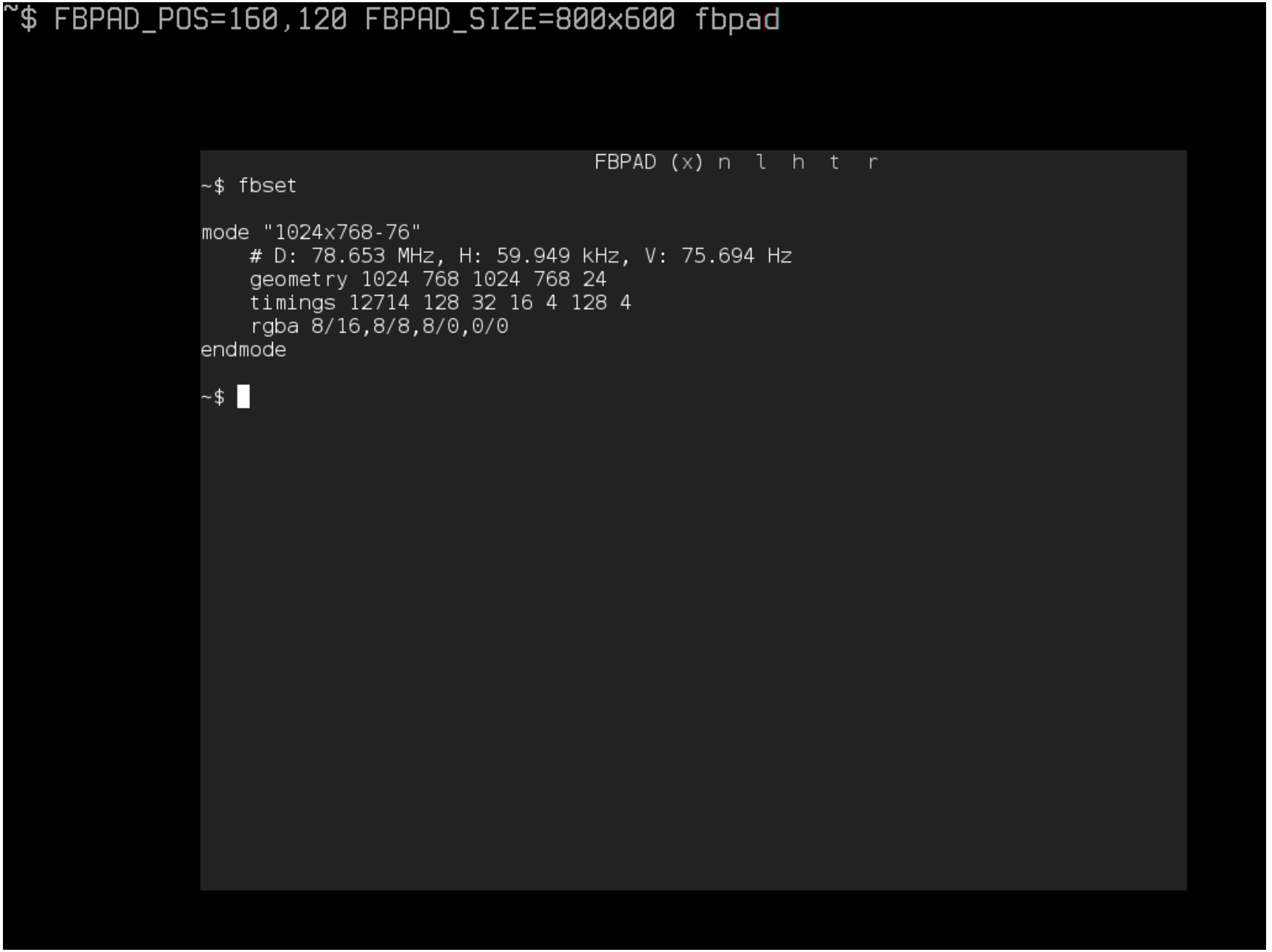
  https://github.com/caramelli/fbmark

  fb_sierpinski.c, fb_mandelbrot.c, fb_rectangle.c

  → **fbmem = mmap()** on **/dev/fb0**

Contents

Fbpad terminal emulator



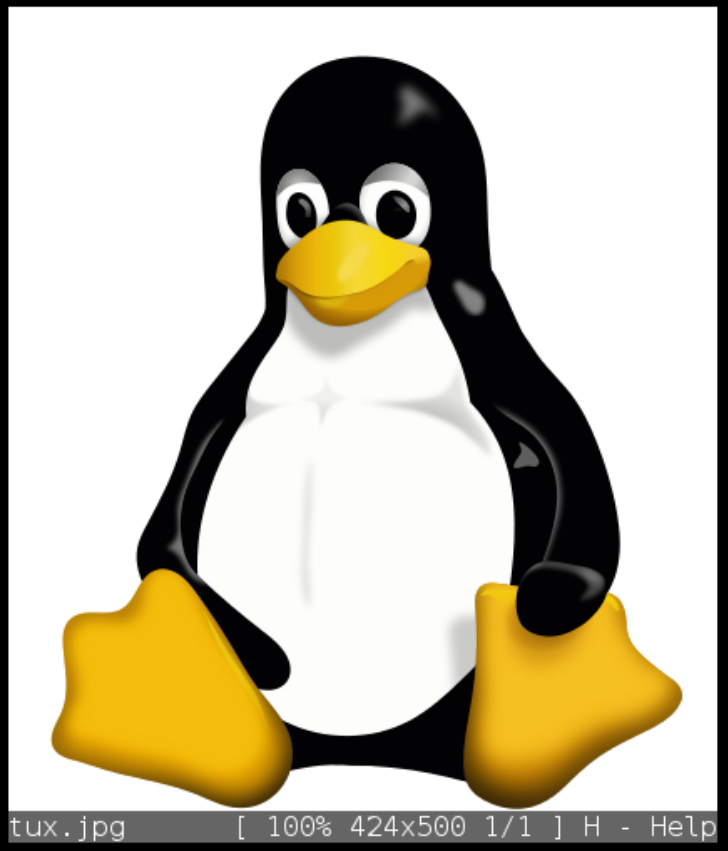
Fbpad terminal emulator

Fbpad https://github.com/aligrudi/fbpad

- → `fb_init()` in *fbpad/draw.c*
  - → **fbmem = mmap()** on **/dev/fb0**
- → `execterm("sh")` in *fbpad/fbpad.c*
  - → `term_exec("sh")` in *fbpad/term.c*
    - → `openpty()` → open pseudoterminal master **/dev/ptmx** and slave **/dev/pts/0** descriptors
    - → `fork()` → child process duplicates slave descriptor to 0, 1, 2 and calls execve("sh")
- → `pollterms()` in *fbpad/fbpad.c*
  - → `readchar()` in *fbpad/fbpad.c* → read on stdin
  - → `term_send()` in *fbpad/term.c*
    - → `writepty()` → master writes to the slave
  - → `term_read()` in *fbpad/term.c*
    - → `readpty()` → master reads data written by the slave
    - → `pad_put()` in *fbpad/pad.c*
      - → `fb_set()` → copy of character to **fbmem**

Fbi, FIM image viewers

```
~$ fbi -geometry 424x500+20+220 tux.jpg &
~$ fim -o fb=500x500+480+180 gnu.png
_
```



Fbi, FIM image viewers

- Fbi https://git.kraxel.org/cgit/fbida
  - → `fb_init()` in *fbi/fbtools.c*
    - → **fbmem = mmap()** on **/dev/fb0**
  - → `read_image()` in *fbi/fbi.c* → use of **libjpeg**, **libpng**, ...
  - → `shadow_render()` in *fbi/fb-gui.c*

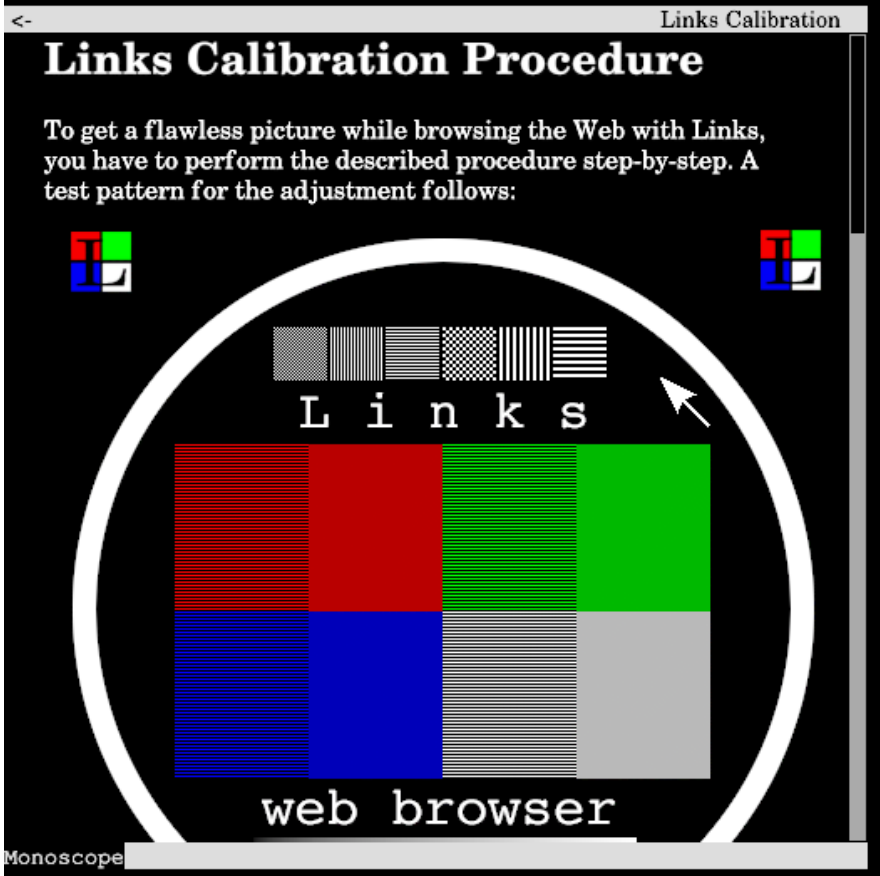
    → copy of decoded image to **fbmem**

- FIM http://svn.savannah.nongnu.org/svn/fbi-improved/trunk
  - → `fb_init()` in *fim/src/FramebufferDevice.cpp*
    - → **fbmem = mmap()** on **/dev/fb0**
  - → `read_image()` in *fim/src/FbiStuff.cpp* → use of **libjpeg**, **libpng**, ...
  - → `convert_line()` in *fim/src/FramebufferDevice.cpp*

    → copy of decoded image to **fbmem**

NetSurf, Links web browsers

```
~$ netsurf-linux --window_width 480 --window_height 480 --window_x 528 --
window_y 172 &
~$ links -mode 0,172,508,80 links/calibration.html
```



NetSurf, Links web browsers

- NetSurf https://git.netsurf-browser.org/netsurf.git

  → Libnsfb https://git.netsurf-browser.org/libnsfb.git

  → `linux_register_surface()` `__attribute__((constructor))` in *libnsfb/src/surface/linux.c*

  → `nsfb_register_surface("linux", NSFB_SURFACE_LINUX, linux_rtns)` in *libnsfb/src/surface/surface.c*

  → `gui_init()` in *netsurf/framebuffer/gui.c*

  → `framebuffer_initialise()` in *netsurf/framebuffer/framebuffer.c*

  → `nsfb_t * nsfb_new(NSFB_SURFACE_LINUX)` in *libnsfb/src/libnsfb.c*

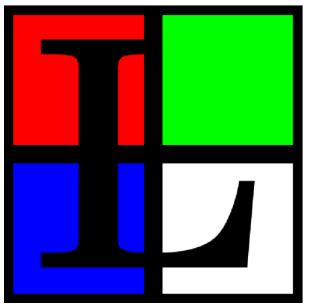
  → `nsfb_init(nsfb_t *)` in *libnsfb/src/libnsfb.c*

  → `linux_initialise()` in *libnsfb/src/surface/linux.c*

  → **fbmem = mmap()** on **/dev/fb0**

  → `nsfb_plot_bitmap()`, … in *libnsfb/src/plot/api.c* → draw in **fbmem**

- Links http://links.twibright.com/download



  → `init()` in *links/main.c*

  → `init_graphics()` in *links/drivers.c*

  → `init_graphics_driver(fb_driver)` in *links/drivers.c*

  → `fb_init_driver()` in *links/framebuffer.c*

  → **fbmem = mmap()** on **/dev/fb0**

  → `fb_draw_bitmap()`, … in *links/framebuffer.c* → draw in **fbmem**

Fbff, MPlayer media players

```
~$ fbff -z 0.2 -x 640 -y 452 02_gran_dillama_1080p.mp4 &
~$ mplayer -quiet -vf scale=640:360 -geometry 0:380 01_llama_drama_1080p.
mp4
==========================================================================
Opening video decoder: [ffmpeg] FFmpeg's libavcodec codec family
Selected video codec: [ffh264] vfm: ffmpeg (FFmpeg H.264)
==========================================================================
==========================================================================
Opening audio decoder: [faad] AAC (MPEG2/4 Advanced Audio Coding)
Selected audio codec: [faad] afm: faad (FAAD AAC (MPEG-2/MPEG-4 Audio))
==========================================================================
AO: [alsa] 48000Hz 2ch s16le (2 bytes per sample)
VO: [fbdev] 640x360 => 640x360 BGR 24-bit
```

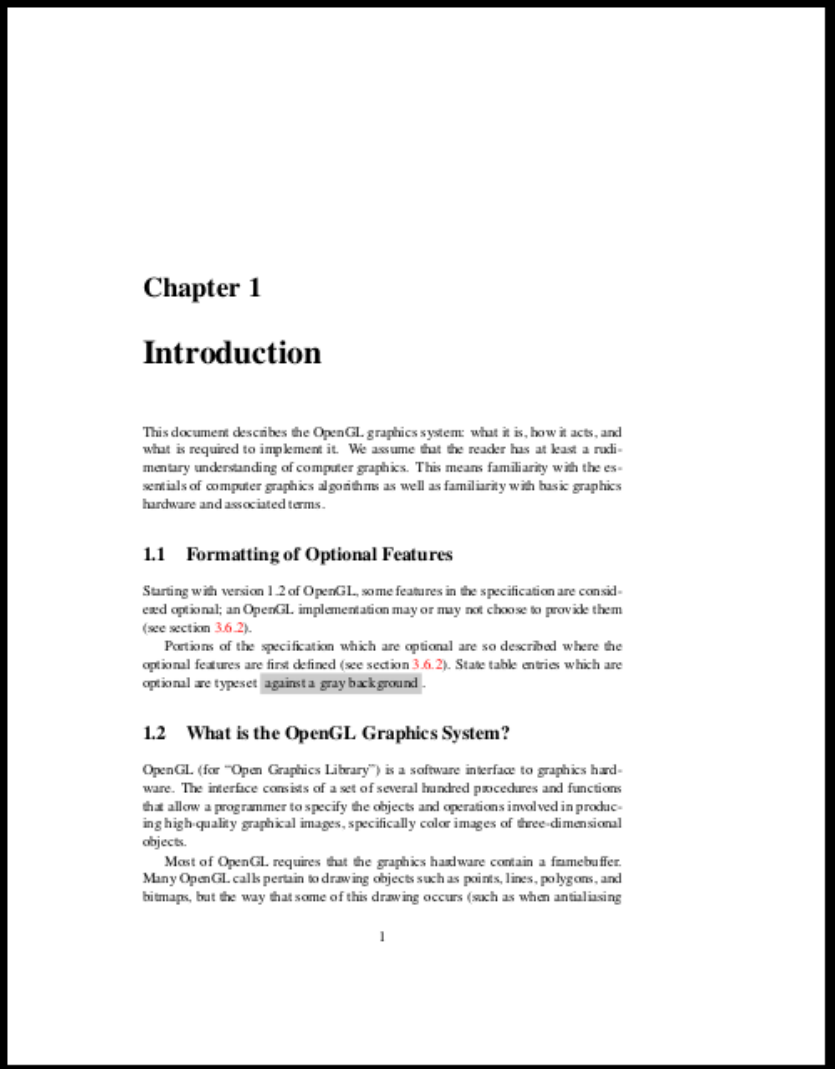
Fbff, MPlayer media players

- Fbff https://github.com/aligrudi/fbff
  - → `fb_init()` in *fbff/draw.c*
    - → **fbmem = mmap()** on **/dev/fb0**
  - → `ffs_vdec()` in *fbff/ffs.c*
  - → `draw_frame()` in *fbff/fbff.c*
    - → `fb_set()` in *fbff/draw.c* → copy of decoded video to **fbmem**

- MPlayer http://svn.mplayerhq.hu/MPlayer/releases
  - → `vo_functions_t * init_best_video_out()` in *MPlayer/libvo/video_out.c*
    - → `return LIBVO_EXTERN(fbdev)`
  - → `decode_video()` in *Mplayer/libmpcodecs/dec_video.c*
    - → `mpcodecs_config_vo()` in *MPlayer/libmpcodecs/vd.c*
      - → `config()` in *MPlayer/libmpcodecs/vf_vo.c*
        - → `config_video_out()` in *MPlayer/libvo/video_out.c*
          - → `config()` in *MPlayer/libvo/vo_fbdev.c*
            - → **fbmem = mmap()** on **/dev/fb0**
    - → `mpcodecs_get_image()` in *MPlayer/libmpcodecs/vd.c*
  - → `filter_video()` in *MPlayer/libmpcodecs/dec_video.c*
    - → `put_image()` in *MPlayer/libmpcodecs/vf_vo.c*
      - → `draw_slice()` in *MPlayer*/libvo/vo_fbdev.c → copy of decoded video to **fbmem**

Fbpdf document viewer

```
~$ fbpoppler -z 8 -x 20 -y 120 glspec21.pdf &
~$ fbmupdf -z 8 -x 525 -y 120 opengles20-reference-card.pdf &
~$ _
```



Chapter 1

Introduction

This document describes the OpenGL graphics system: what it is, how it acts, and what is required to implement it. We assume that the reader has at least a rudimentary understanding of computer graphics. This means familiarity with the essentials of computer graphics algorithms as well as familiarity with basic graphics hardware and associated terms.

1.1 Formatting of Optional Features

Starting with version 1.2 of OpenGL, some features in the specification are considered optional; an OpenGL implementation may or may not choose to provide them (see section 3.6.2).

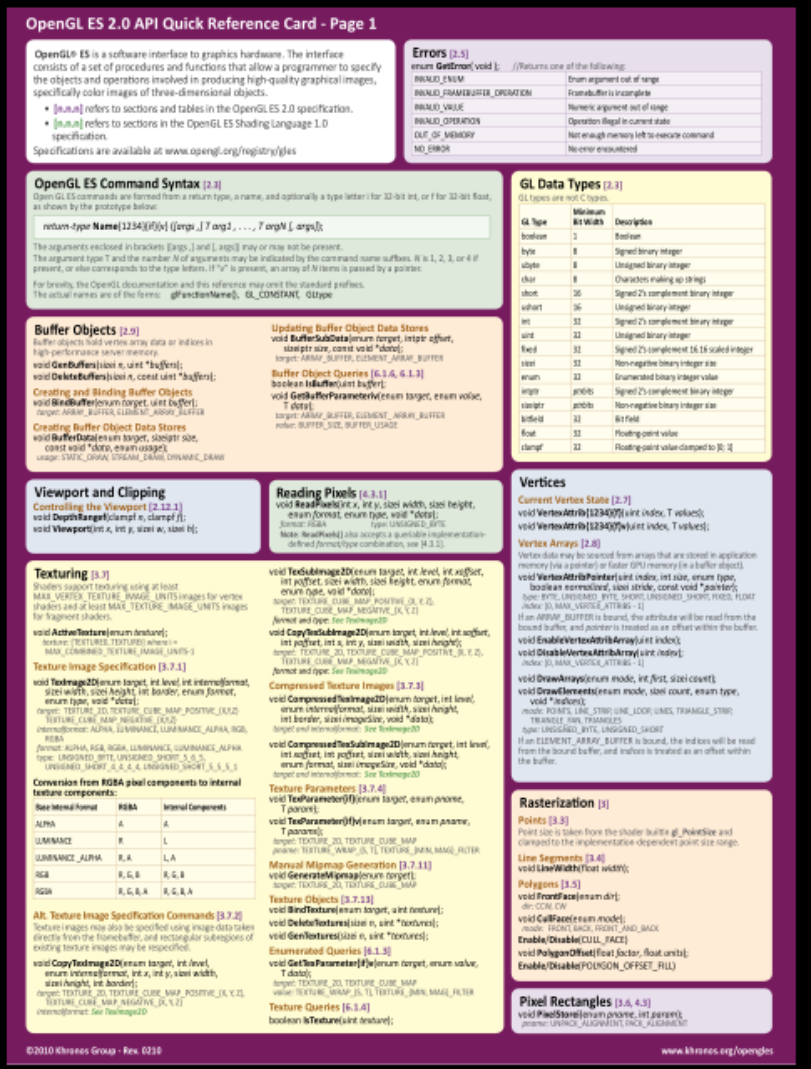
Portions of the specification which are optional are so described where the optional features are first defined (see section 3.6.2). State table entries which are optional are typeset against a gray background.

1.2 What is the OpenGL Graphics System?

OpenGL (for "Open Graphics Library") is a software interface to graphics hardware. The interface consists of a set of several hundred procedures and functions that allow a programmer to specify the objects and operations involved in producing high-quality graphical images, specifically color images of three-dimensional objects.

Most of OpenGL requires that the graphics hardware contain a framebuffer. Many OpenGL calls pertain to drawing objects such as points, lines, polygons, and bitmaps, but the way that some of this drawing occurs (such as when antialiasing

1



OpenGL ES 2.0 API Quick Reference Card - Page 1

OpenGL® ES is a software interface to graphics hardware. The interface consists of a set of procedures and functions that allow a programmer to specify the objects and operations involved in producing high-quality graphical images, specifically color images of three-dimensional objects.

- [n.n.n] refers to sections and tables in the OpenGL ES 2.0 specification.
- [n.n.n] refers to sections in the OpenGL ES Shading Language 1.0 specification.

Specifications are available at www.opengl.org/registry/gles

Errors [2.5]

enum GetError(void); //Returns one of the following:

INVALID_ENUM	Enum argument out of range
INVALID_FRAMEBUFFER_OPERATION	Framebuffer is incomplete
INVALID_VALUE	Numeric argument out of range
INVALID_OPERATION	Operation illegal in current state
OUT_OF_MEMORY	Not enough memory left to execute command
NO_ERROR	No error encountered

OpenGL ES Command Syntax [2.3]

OpenGL ES commands are formed from a return type, a name, and optionally a type letter i for 32-bit int, or f for 32-bit float, as shown by the prototype below:

return-type Name{1234}{if}{v} ([args ,] T arg1 , ..., T argN [, args]);

The arguments enclosed in brackets ([args ,] and [, args]) may or may not be present.
The argument type T and the number N of arguments may be indicated by the command name suffixes. N is 1, 2, 3, or 4 if present, or else corresponds to the type letters. If "v" is present, an array of N items is passed by a pointer.

For brevity, the OpenGL documentation and this reference may omit the standard prefixes.
The actual names are of the forms: glFunctionName(), GL_CONSTANT, GLtype

GL Data Types [2.3]

GL types are not C types.

GL Type	Minimum Bit Width	Description
boolean	1	Boolean
byte	8	Signed binary integer
ubyte	8	Unsigned binary integer
char	8	Characters making up strings
short	16	Signed 2's complement binary integer
ushort	16	Unsigned binary integer
int	32	Signed 2's complement binary integer
uint	32	Unsigned binary integer
fixed	32	Signed 2's complement 16.16 scaled integer
sizei	32	Non-negative binary integer size
enum	32	Enumerated binary integer value
intptr	ptrbits	Signed 2's complement binary integer
sizeiptr	ptrbits	Non-negative binary integer size
bitfield	32	Bit field
float	32	Floating-point value
clampf	32	Floating-point value clamped to [0, 1]

Buffer Objects [2.9]

Buffer objects hold vertex array data or indices in high-performance server memory.

void GenBuffers(sizei n, uint *buffers);
void DeleteBuffers(sizei n, const uint *buffers);

Creating and Binding Buffer Objects

void BindBuffer(enum target, uint buffer);
target: ARRAY_BUFFER, ELEMENT_ARRAY_BUFFER

Creating Buffer Object Data Stores

void BufferData(enum target, sizeiptr size, const void *data, enum usage);
usage: STATIC_DRAW, STREAM_DRAW, DYNAMIC_DRAW

Updating Buffer Object Data Stores

void BufferSubData(enum target, intptr offset, sizeiptr size, const void *data);
target: ARRAY_BUFFER, ELEMENT_ARRAY_BUFFER

Buffer Object Queries [6.1.6, 6.1.3]

boolean IsBuffer(uint buffer);

void GetBufferParameteriv(enum target, enum value, T data);
target: ARRAY_BUFFER, ELEMENT_ARRAY_BUFFER
value: BUFFER_SIZE, BUFFER_USAGE

Viewport and Clipping

Controlling the Viewport [2.12.1]

void DepthRangef(clampf n, clampf f);
void Viewport(int x, int y, sizei w, sizei h);

Reading Pixels [4.3.1]

void ReadPixels(int x, int y, sizei width, sizei height, enum format, enum type, void *data);
format: RGBA
type: UNSIGNED_BYTE
Note: ReadPixels() also accepts a queriable implementation-defined format/type combination, see [4.3.1].

Texturing [3.7]

Shaders support texturing using at least MAX_VERTEX_TEXTURE_IMAGE_UNITS images for vertex shaders and at least MAX_TEXTURE_IMAGE_UNITS images for fragment shaders.

void ActiveTexture(enum texture);
texture: TEXTURE0, TEXTURE1, where i = MAX_COMBINED_TEXTURE_IMAGE_UNITS-1

Texture Image Specification [3.7.1]

void TexImage2D(enum target, int level, int internalformat, sizei width, sizei height, int border, enum format, enum type, void *data);
target: TEXTURE_2D, TEXTURE_CUBE_MAP_POSITIVE_{X,Y,Z}, TEXTURE_CUBE_MAP_NEGATIVE_{X,Y,Z}
internalformat: ALPHA, LUMINANCE, LUMINANCE_ALPHA, RGB, RGBA
format: ALPHA, RGB, RGBA, LUMINANCE, LUMINANCE_ALPHA
type: UNSIGNED_BYTE, UNSIGNED_SHORT_5_6_5, UNSIGNED_SHORT_4_4_4_4, UNSIGNED_SHORT_5_5_5_1

Conversion from RGBA pixel components to internal texture components:

Base Internal Format	RGBA	Internal Components
ALPHA	A	A
LUMINANCE	R	L
LUMINANCE_ALPHA	R, A	L, A
RGB	R, G, B	R, G, B
RGBA	R, G, B, A	R, G, B, A

Alt. Texture Image Specification Commands [3.7.2]

Texture images may also be specified using image data taken directly from the framebuffer, and rectangular subregions of existing texture images may be respecified.

void CopyTexImage2D(enum target, int level, enum internalformat, int x, int y, sizei width, sizei height, int border);
target: TEXTURE_2D, TEXTURE_CUBE_MAP_POSITIVE_{X,Y,Z}, TEXTURE_CUBE_MAP_NEGATIVE_{X,Y,Z}
internalformat: See TexImage2D

void TexSubImage2D(enum target, int level, int xoffset, int yoffset, sizei width, sizei height, enum format, enum type, void *data);
target: TEXTURE_CUBE_MAP_POSITIVE_{X,Y,Z}, TEXTURE_CUBE_MAP_NEGATIVE_{X,Y,Z}
format and type: See TexImage2D

void CopyTexSubImage2D(enum target, int level, int xoffset, int yoffset, int x, int y, sizei width, sizei height);
target: TEXTURE_2D, TEXTURE_CUBE_MAP_POSITIVE_{X,Y,Z}, TEXTURE_CUBE_MAP_NEGATIVE_{X,Y,Z}
format and type: See TexImage2D

Compressed Texture Images [3.7.3]

void CompressedTexImage2D(enum target, int level, enum internalformat, sizei width, sizei height, int border, sizei imageSize, void *data);

void CompressedTexSubImage2D(enum target, int level, int xoffset, int yoffset, sizei width, sizei height, enum format, sizei imageSize, void *data);
target and internalformat: See TexImage2D

Texture Parameters [3.7.4]

void TexParameter{if}(enum target, enum pname, T param);
void TexParameter{if}v(enum target, enum pname, T params);
target: TEXTURE_2D, TEXTURE_CUBE_MAP
pname: TEXTURE_WRAP_{S, T}, TEXTURE_{MIN, MAG}_FILTER

Manual Mipmap Generation [3.7.11]

void GenerateMipmap(enum target);
target: TEXTURE_2D, TEXTURE_CUBE_MAP

Texture Objects [3.7.13]

void BindTexture(enum target, uint texture);
void DeleteTextures(sizei n, uint *textures);
void GenTextures(sizei n, uint *textures);

Enumerated Queries [6.1.3]

void GetTexParameter{if}v(enum target, enum value, T data);
target: TEXTURE_2D, TEXTURE_CUBE_MAP
value: TEXTURE_WRAP_{S, T}, TEXTURE_{MIN, MAG}_FILTER

Texture Queries [6.1.4]

boolean IsTexture(uint texture);

Vertices

Current Vertex State [2.7]

void VertexAttrib{1234}{if}(uint index, T values);
void VertexAttrib{1234}{if}v(uint index, T values);

Vertex Arrays [2.8]

Vertex data may be sourced from arrays that are stored in application memory (via a pointer) or faster GPU memory (in a buffer object).

void VertexAttribPointer(uint index, int size, enum type, boolean normalized, sizei stride, const void *pointer);
type: BYTE, UNSIGNED_BYTE, SHORT, UNSIGNED_SHORT, FIXED, FLOAT
index: [0, MAX_VERTEX_ATTRIBS - 1]
If an ARRAY_BUFFER is bound, the attribute will be read from the bound buffer, and pointer is treated as an offset within the buffer.

void EnableVertexAttribArray(uint index);
void DisableVertexAttribArray(uint index);
index: [0, MAX_VERTEX_ATTRIBS - 1]

void DrawArrays(enum mode, int first, sizei count);
void DrawElements(enum mode, sizei count, enum type, void *indices);
mode: POINTS, LINE_STRIP, LINE_LOOP, LINES, TRIANGLE_STRIP, TRIANGLE_FAN, TRIANGLES
type: UNSIGNED_BYTE, UNSIGNED_SHORT
If an ELEMENT_ARRAY_BUFFER is bound, the indices will be read from the bound buffer, and indices is treated as an offset within the buffer.

Rasterization [3]

Points [3.3]

Point size is taken from the shader builtin gl_PointSize and clamped to the implementation-dependent point size range.

Line Segments [3.4]

void LineWidth(float width);

Polygons [3.5]

void FrontFace(enum dir);
dir: CCW, CW

void CullFace(enum mode);
mode: FRONT, BACK, FRONT_AND_BACK

Enable/Disable(CULL_FACE)

void PolygonOffset(float factor, float units);

Enable/Disable(POLYGON_OFFSET_FILL)

Pixel Rectangles [3.6, 4.3]

void PixelStorei(enum pname, int param);
pname: UNPACK_ALIGNMENT, PACK_ALIGNMENT

©2010 Khronos Group - Rev. 0210

www.khronos.org/opengles

Fbpdf document viewer

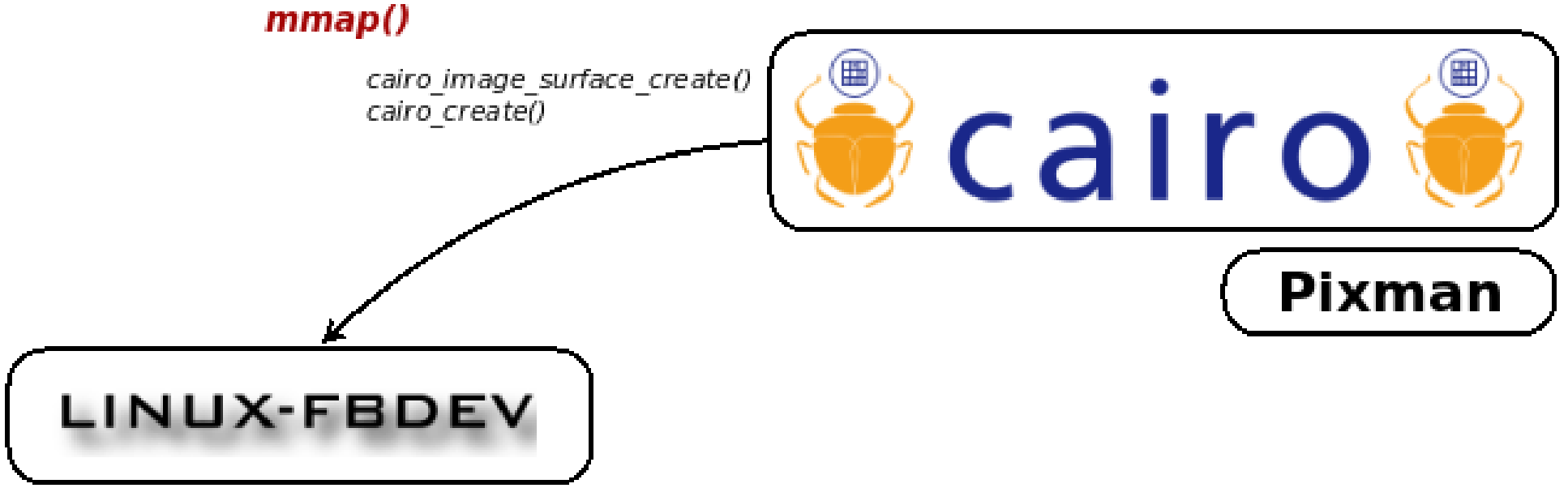
Fbpdf https://github.com/aligrudi/fbpdf

- fbmupdf → based on MuPDF framework
  - → `fb_init()` in *fbpdf/draw.c*
    - → **fbmem = mmap()** on **/dev/fb0**
  - → `showpage()` in *fbpdf/fbpdf.c*
    - → `doc_draw()` in *fbpdf/mupdf.c*
      - → `fz_run_page()`
    - → `fb_set()` in *fbpdf/draw.c* → copy of PDF page to render in **fbmem**

- fbpoppler → based on Poppler framework
  - → `fb_init()` in *fbpdf/draw.c*
    - → **fbmem = mmap()** on **/dev/fb0**
  - → `showpage()` in *fbpdf/fbpdf.c*
    - → `doc_draw()` in *fbpdf/poppler.c*
      - → `poppler_page_render()`
    - → `fb_set()` in *fbpdf/draw.c* → copy of PDF page to render in **fbmem**

Contents

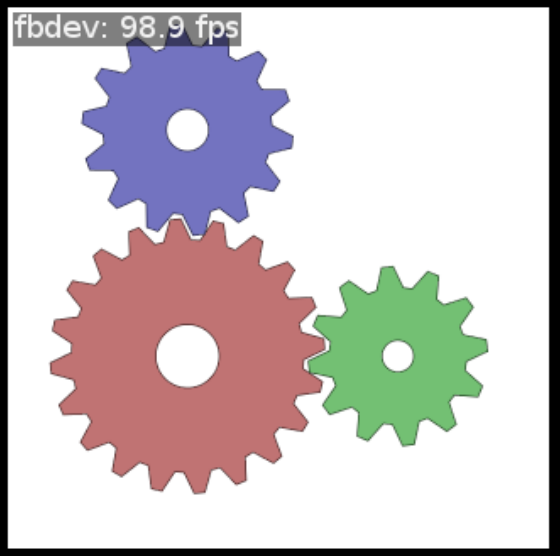
Cairo

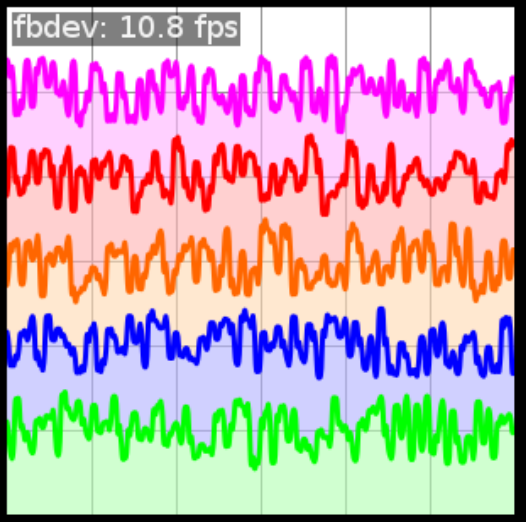


- Cairo https://cgit.freedesktop.org/cairo
- Cairo-demos
  - https://gitlab.com/cairo/cairo-demos
    - ➡ fbdev/*cairo-fb.c*
  - https://cgit.freedesktop.org/~ickle/cairo-demos
    - ➡ *fbdev.c*

Cairo demo

```
~$ cairo-fb
The framebuffer device was opened successfully
1024x768, 24bpp
The framebuffer device was successfully mapped
~$ tiger-demo --size 320x320 --position 680x430 &
~$ chart-demo --size 300x300 --position 360x450 &
~$ gears-demo --size 320x320 --position 20x430 &
~$ fish-demo --size 320x320 --position 700x60 &
~$ _
```





Cairo internal

- Create a target surface for Linux Framebuffer (2 methods)

  1) `cairo_surface_t * cairo_image_surface_create_for_data(void *`**fbmem**`)`

     where **fbmem = mmap()** on **/dev/fb0**

     → `pixman_image_create_bits()` in *cairo/src/cairo-image-surface.c*

  `cairo_surface_t * cairo_surface_create_similar(cairo_surface_t *)` for double buffering

  2) `cairo_surface_t * cairo_image_surface_create()`

     → `pixman_image_create_bits()` in *cairo/src/cairo-image-surface.c*

- Create Cairo context

  `cairo_t * cairo_create(cairo_surface_t *)`

- Draw using the Cairo API

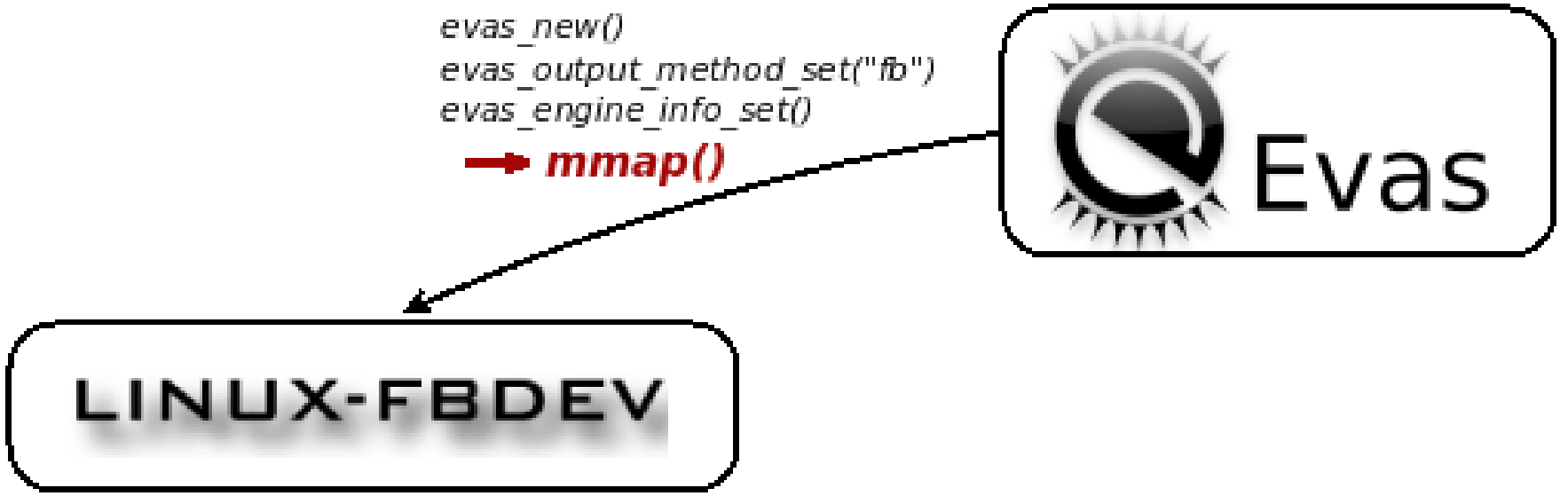
  `cairo_rectangle(cairo_t *), cairo_line_to(cairo_t *),`
  `cairo_arc(cairo_t *), cairo_show_text(cairo_t *), ...`
  `cairo_set_source_surface(cairo_t *, cairo_surface_t *), cairo_paint(cairo_t *)`

  → copy to **fbmem**

  `cairo_image_surface_get_data(cairo_surface_t *)`

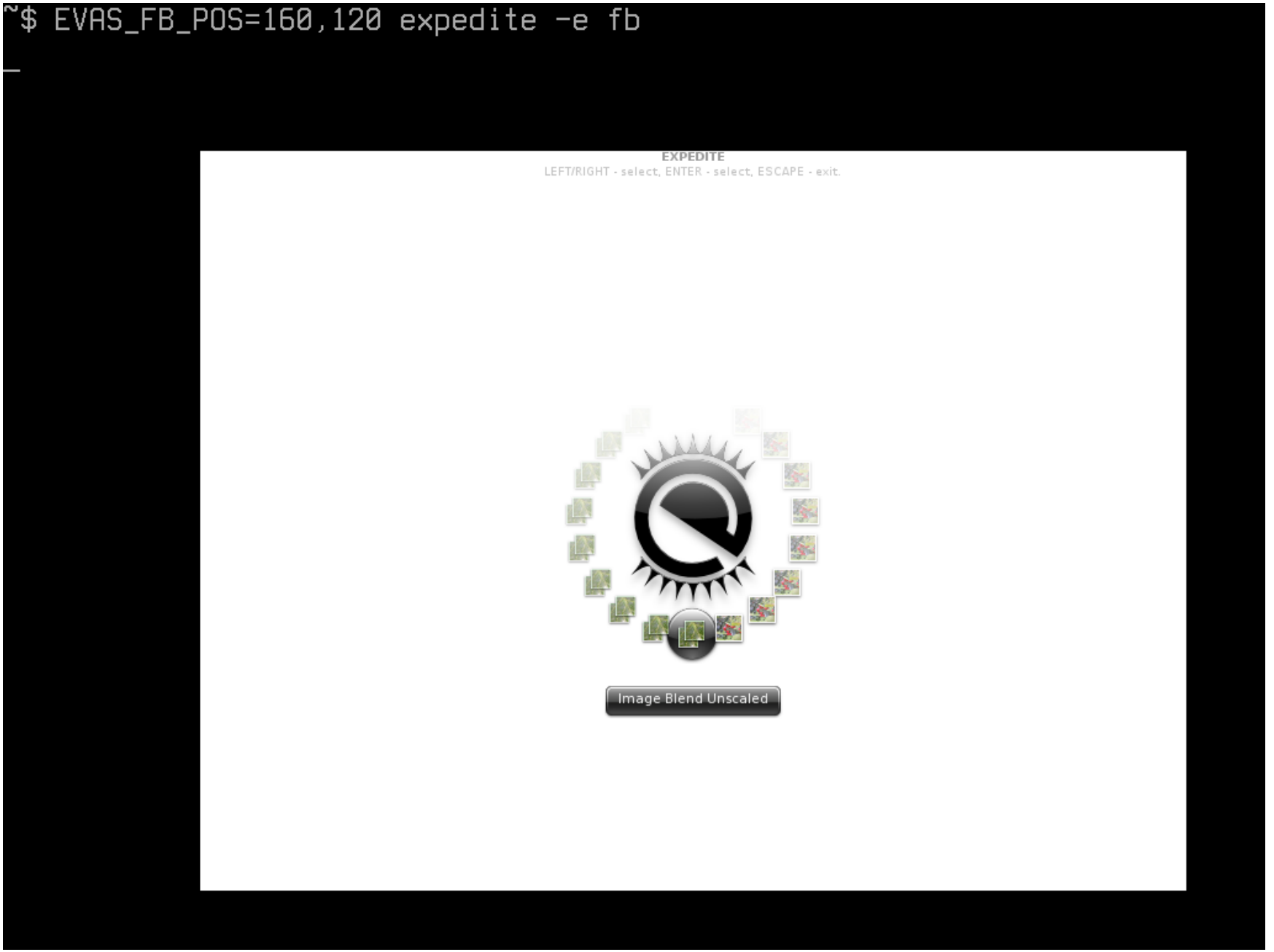
  → to be copied to **fbmem**

Evas



- Evas
  - https://git.enlightenment.org/core/efl.git/
  - https://git.enlightenment.org/legacy/evas.git/
    - → Linux Framebuffer support in *engines/fb* directory
- Expedite https://git.enlightenment.org/tools/expedite.git/
  - ➡ *src/bin/engine_fb.c*

Evas demo


~$ EVAS_FB_POS=160,120 expedite -e fb
EXPEDITE
LEFT/RIGHT - select, ENTER - select, ESCAPE - exit.
Image Blend Unscaled

Evas internal

- Setup a canvas for Linux Framebuffer
  - `Evas * evas_new()`
  - `evas_output_method_set(Evas *, evas_render_method_lookup("fb"))`
  - `evas_engine_info_set(Evas *, Evas_Engine_Info_FB *)`
    - → `setup()` in *evas/engines/fb/evas_engine.c*
      - → `evas_fb_outbuf_fb_setup_fb()` in *evas/engines/fb/evas_outbuf.c*
        - → `fb_postinit()` in *evas/engines/fb/evas_fb_main.c*
          - → **fbmem = mmap()** on **/dev/fb0**
- Draw using the Evas API

  `evas_object_rectangle_add(Evas *)`, `evas_object_line_add(Evas *)`, `evas_object_image_add(Evas *)`, `evas_object_text_add(Evas *)`, ...

  `evas_render(Evas *)` or `evas_render_updates(Evas *)`

  → `output_redraws_next_update_push()` in *evas/engines/fb/evas_engine.c*

  → `evas_fb_outbuf_fb_push_updated_region()` in *evas/engines/fb/evas_outbuf.c*

  → copy to **fbmem**

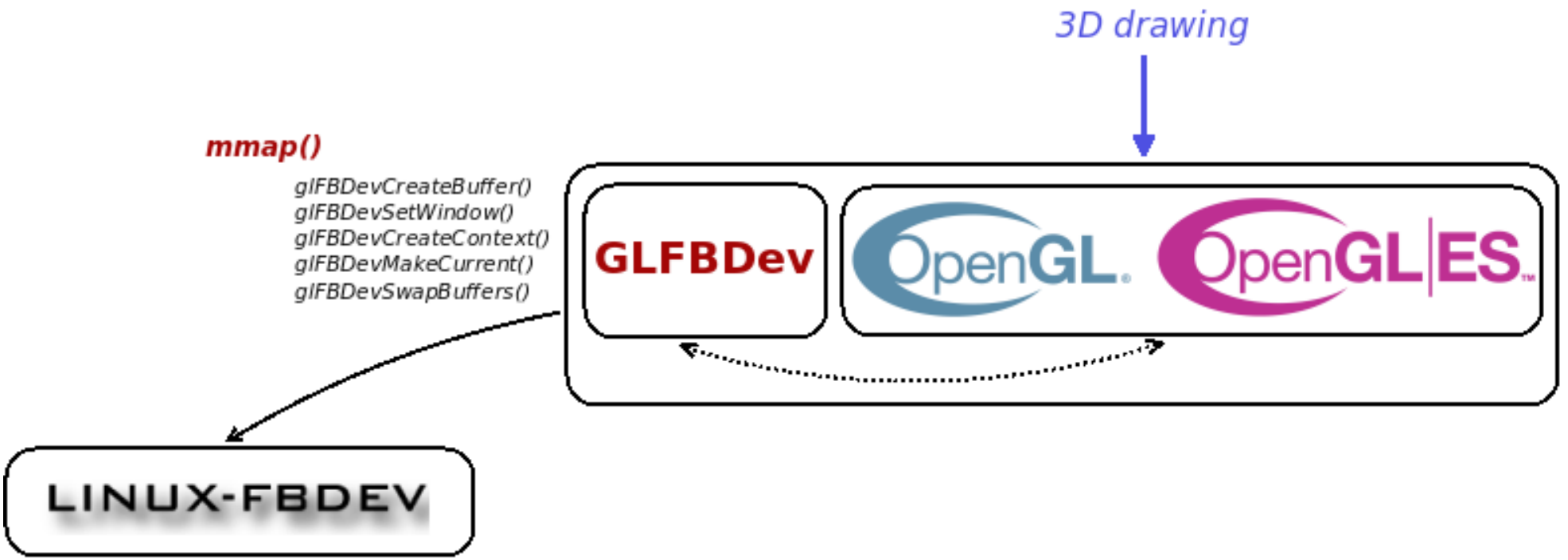
Contents

OpenGL rendering

- The Mesa 3D project makes OpenGL and OpenGL ES rendering possible using CPU operations only and the Linux Framebuffer (without requiring a GPU)

- Applications can choose between 2 APIs for rendering
  - **GLFBDev** (OpenGL Extension to the Linux Framebuffer)
    → based on *Mesa legacy* infrastructure
  - **EGL** for Linux Framebuffer platform
    → based on *Mesa Gallium3D* infrastructure

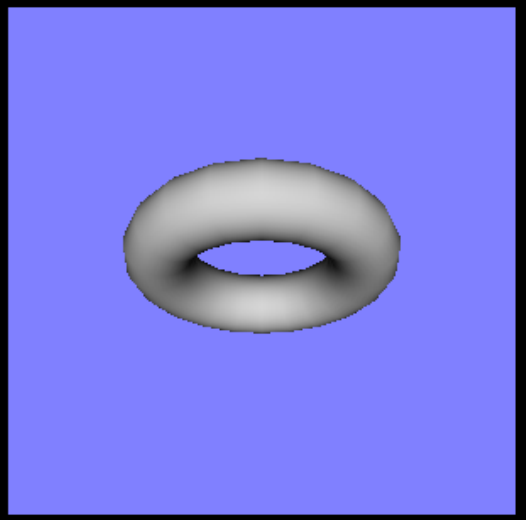
GLFBDev extension



- Implemented in Mesa https://gitlab.freedesktop.org/mesa/mesa
  → Interfaces and Linux Framebuffer support in *src/mesa/drivers/fbdev/glfbdev.c*
- Examples:
  - mesa-demos https://gitlab.freedesktop.org/mesa/demos
    ➡ *progs/fbdev/glfbdevtest.c*
  - yagears https://github.com/caramelli/yagears

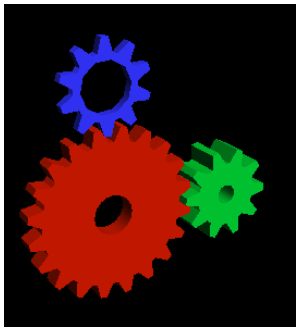
GLFBDev extension demo

```
~$ POSX=100 POSY=200 glfbdevtest -f 10000 &
~$ WIDTH=360 HEIGHT=360 POSX=560 POSY=200 yagears -b gl-fbdev -e gl &
~$
```



GLFBDev extension internal

yagears -b gl-fbdev -e gl



- `GLFBDevBufferPtr glFBDevCreateBuffer(void *`**fbmem**`)`

  where **fbmem = mmap()** on **/dev/fb0**

  → in *Mesa/src/mesa/drivers/fbdev/glfbdev.c*

  _mesa_initialize_window_framebuffer(struct gl_framebuffer *)

- `glFBDevSetWindow(GLFBDevBufferPtr, struct fb_window)`

  with `struct fb_window { int width; int height; int posx; int posy; };`

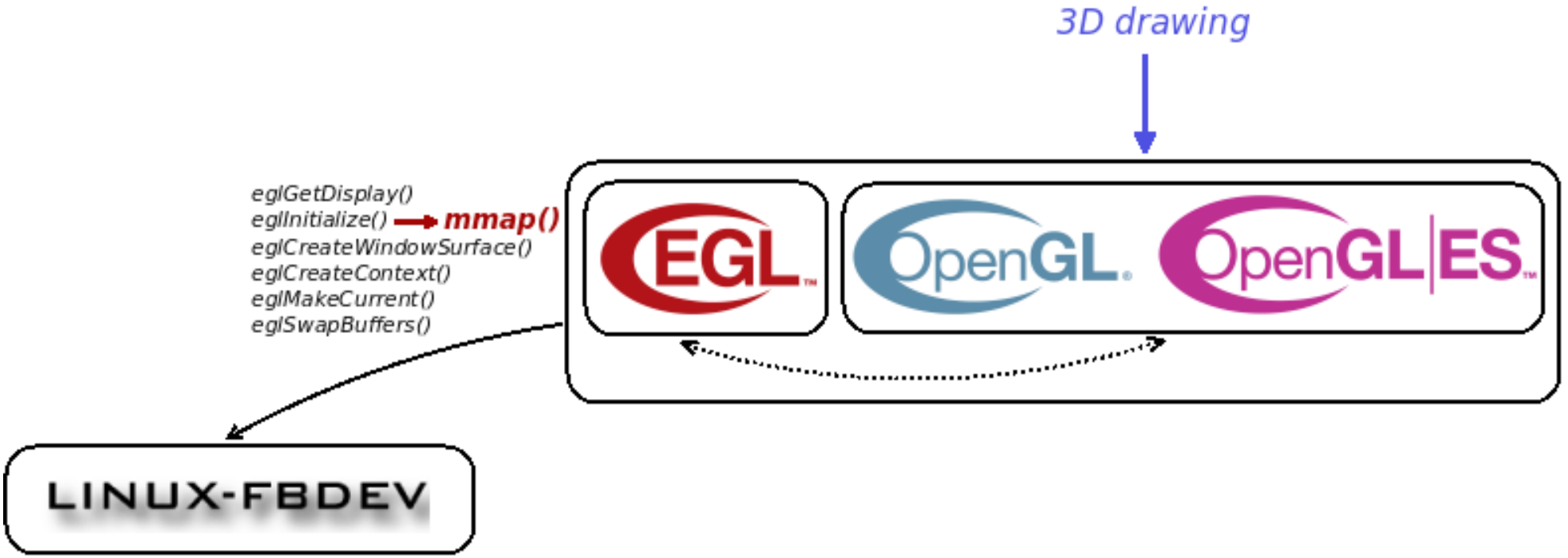
- `GLFBDevContextPtr glFBDevCreateContext()`

- `glFBDevMakeCurrent(GLFBDevContextPtr, GLFBDevBufferPtr)`

- `glFBDevSwapBuffers(GLFBDevBufferPtr)`

→ copy of 3D drawing based on OpenGL API to **fbmem**

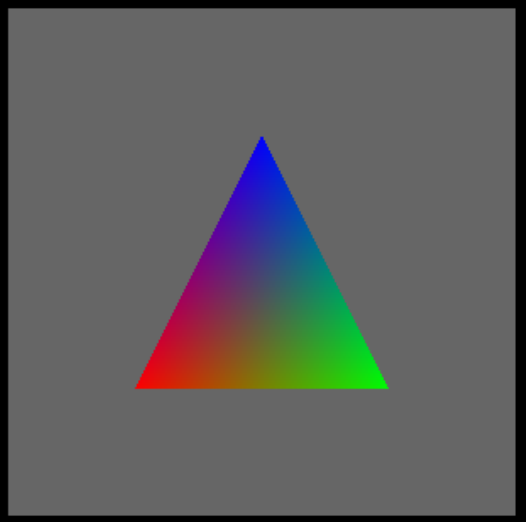
EGL for Linux Framebuffer



- Implemented in Mesa https://gitlab.freedesktop.org/mesa/mesa
  → Interfaces in *src/egl/main/eglapi.c*, Linux Framebuffer support in *src/gallium/state_trackers/egl/fbdev* and *src/gallium/winsys/sw/fbdev* directories
- Examples:
  - mesa-demos https://gitlab.freedesktop.org/mesa/demos
    → *src/egl/opengles1/eglfbdev.c*
  - yagears https://github.com/caramelli/yagears

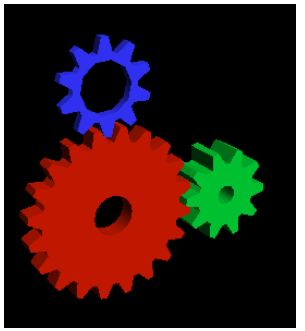
EGL for Linux Framebuffer demo

```
~$ POSX=100 POSY=200 egltri_fbdev &
~$ WIDTH=360 HEIGHT=360 POSX=560 POSY=200 yagears -b egl-fbdev -e gl &
~$ _
```



EGL for Linux Framebuffer internal

yagears -b egl-fbdev -e gl

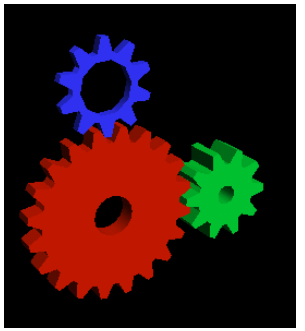


- `EGLDisplay eglGetDisplay()`
  - → `eglGetNativePlatform()` in *Mesa/src/egl/main/egldisplay.c*
    - → `getenv("EGL_PLATFORM")` ➡ `EGL_PLATFORM` needs to be set to **“fbdev”**
- `eglInitialize(egl_dpy, ...)`
  - → `egl_g3d_initialize()` in *Mesa/src/gallium/state_trackers/egl/common/egl_g3d.c*
    - → `native_create_display()` in *Mesa/src/gallium/state_trackers/egl/fbdev/native_fbdev.c*
      - → `fbdev_create_sw_winsys()` in *Mesa/src/gallium/winsys/sw/fbdev/fbdev_sw_winsys.c*
        - → **fbmem = mmap()** on **/dev/fb0**
- `EGLSurface eglCreateWindowSurface(EGLDisplay, struct fb_window *)`

  with `struct fb_window { int width; int height; int posx; int posy; };`
- `EGLContext eglCreateContext()`
- `eglMakeCurrent(EGLSurface, EGLContext)`
  - → `egl_g3d_make_current()` in *Mesa/src/gallium/state_trackers/egl/common/egl_g3d_api.c*
    - → `st_api_make_current()` in *Mesa/src/mesa/state_tracker/st_manager.c*
      - → **_mesa_initialize_window_framebuffer(struct gl_framebuffer *)**

EGL for Linux Framebuffer internal

yagears -b egl-fbdev -e gl



- `eglSwapBuffers(EGLSurface)`

  → `egl_g3d_swap_buffers()` in *Mesa/src/gallium/state_trackers/egl/common/egl_g3d_api.c*

  → `fbdev_surface_present()` in *Mesa/src/gallium/state_trackers/egl/fbdev/native_fbdev.c*

  → `resource_surface_present()` in *Mesa/src/gallium/state_trackers/egl/common/native_helper.c*

  → `softpipe_flush_frontbuffer()` in *Mesa/src/gallium/drivers/softpipe/sp_screen.c*

  → `fbdev_displaytarget_display()` in *Mesa/src/gallium/winsys/sw/fbdev/fbdev_sw_winsys.c*

  → copy of 3D drawing based on OpenGL API to **fbmem**

- `EGL_LOG_LEVEL = debug` to print some informations

Contents

Multimedia frameworks

- Multimedia frameworks (FFmpeg, GStreamer, Xine, VLC) provide an output based on the Linux Framebuffer

- Display of the decoded video is CPU based

  → no video overlay

FFmpeg

```
~$ ffmpeg -nostdin -hide_banner -loglevel quiet -nostats -s 854x480 -pix_
fmt bgr24 -f fbdev -xoffset 120 -yoffset 150 /dev/fb0 -re -i big_buck_bun
ny_480p_stereo.avi &
~$ ffmpeg -nostdin -hide_banner -loglevel quiet -v -1 -s 404x108 -pix_fmt
 bgr24 -f fbdev -xoffset 120 -yoffset 650 /dev/fb0 -i ffmpeg-logo.png
~$ _
```

FFmpeg internal

FFmpeg https://git.ffmpeg.org/ffmpeg.git

→ ffmpeg in fftools or root directory

- → `avcodec_register_all() / av_register_all()`
- → `avdevice_register_all()` in *libavdevice/alldevices.c*
  - → `av_register_output_format(fbdev_muxer)` in *libavformat/utils.c*
- → `parse_options()` in *cmdutils.c*
  - → `AVFormatContext *ic = avformat_alloc_context(), av_open_input_file(ic, …)`
  - → `AVFormatContext *oc = avformat_alloc_context(), av_guess_format("fbdev"), av_new_stream(oc), av_guess_codec(fbdev_muxer)` → **CODEC_ID_RAWVIDEO**
- → `transcode()` in *ffmpeg.c*
  - → `av_write_header(oc)` in *libavformat/utils.c*
    - → `fbdev_write_header(oc)` in *libavdevice/fbdev_enc.c*
      - → **fbmem = mmap()** on **/dev/fb0**
  - → `av_read_frame(ic, AVPacket *)`
  - → `avcodec_decode_video(ic->streams[0]->codec, AVFrame *, AVPacket *)`
  - → `avcodec_encode_video(oc->streams[0]->codec, AVPacket *, AVFrame *)`
  - → `av_interleaved_write_frame(oc, AVPacket *)` in *libavformat/utils.c*
    - → `fbdev_write_packet(oc, AVPacket *)` in *libavdevice/fbdev_enc.c*

→ copy to **fbmem**

GStreamer

```
~$ gst-launch-0.10 -q filesrc location=big_buck_bunny_480p_stereo.avi ! d
ecodebin ! ffmpegcolorspace ! fbdevsink offset-x=120 offset-y=150 &
~$ gst-launch-1.0 -q filesrc location=gstreamer-logo.jpg ! decodebin ! vi
deoconvert ! imagefreeze ! fbdevsink offset-x=120 offset-y=650 &
~$
```

GStreamer internal

- GStreamer https://gitlab.freedesktop.org/gstreamer/gstreamer

➡ gst-launch in *tools* directory

→ `gst_init()`

→ `GstElement * gst_parse_launchv("... ! fbdevsink ...")`

→ `gst_parse_launch_full()` in *gstreamer/gst/gstparse.c*

→ `gst_element_factory_make()` in *gstreamer/gst/gstelementfactory.c*

→ `g_object_new(GST_TYPE_FBDEVSINK)` from GLib

→ `gst_fbdevsink_class_init()` in *gst-plugins-bad/sys/fbdev/gstfbdevsink.c*

→ `gst_element_set_state(GstElement *, GST_STATE_PLAYING);`

→ Decoding (for example by **Libavcodec** from FFmpeg through GStreamer Libav plug-in)

- Rendering to Linux Framebuffer in GStreamer Bad Plug-ins

https://gitlab.freedesktop.org/gstreamer/gst-plugins-bad

→ `gst_fbdevsink_start()` in *gst-plugins-bad/sys/fbdev/gstfbdevsink.c*

→ **fbmem = mmap()** on **/dev/fb0**

in *gst-plugins-bad/sys/fbdev/gstfbdevsink.c*

→ `gst_fbdevsink_render()` on GStreamer 0.10

→ `gst_fbdevsink_show_frame()` on GStreamer 1

➡ copy to **fbmem**

Xine


~$ fbxine -x 120 -y 150 big_buck_bunny_480p_stereo.avi &
~$ fbxine -x 120 -y 650 xine-ui_logo.png &
~$ _
Big Buck
BUNNY
XINE

Xine internal

- xine-lib
  - https://sourceforge.net/p/xine/xine-lib
  - https://sourceforge.net/p/xine/xine-lib-1.2

- xine-ui https://sourceforge.net/p/xine/xine-ui

➡ fbxine in *src/fb* directory

→ `xine_t * xine_new(), xine_init(xine_t *)`

→ `xine_video_port_t * xine_open_video_driver(xine_t *, "fb")`

→ `load_video_driver()` in *xine-lib/src/xine-engine/load_plugins.c*

→ `load_plugin_class()` in *xine-lib/src/xine-engine/load_plugins.c*

→ `fb_init_class()` in *xine-lib/src/video_out/video_out_fb.c*

→ `fb_open_plugin()` in *xine-lib/src/video_out/video_out_fb.c*

→ **fbmem = mmap()** on **/dev/fb0**

→ `xine_stream_t * xine_stream_new(xine_t *, xine_video_port_t *)`

→ `xine_open(xine_stream_t *, ...), xine_play (xine_stream_t *)`

→ Decoding (for example by **Libavcodec** from FFmpeg)

→ `fb_display_frame()` in *xine-lib/src/video_out/video_out_fb.c* ➡ copy to **fbmem**

VLC

```
~$ vlc --quiet --vout fb --video-x=120 --video-y=150 big_buck_bunny_480p_
stereo.avi &
~$ vlc --quiet --no-video-title-show --vout fb --video-x=120 --video-y=65
0 vlc-logo.jpg &
~$
```

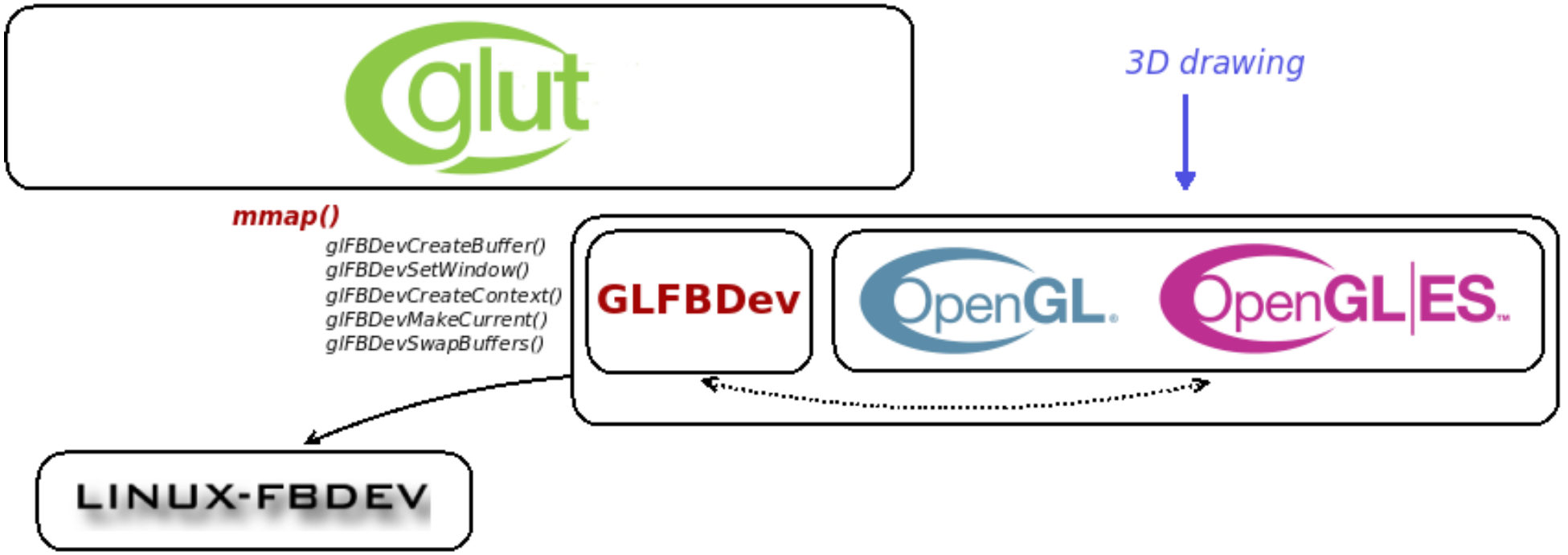
VLC internal

VLC https://git.videolan.org/?p=vlc.git

- vlc in *bin* directory
  - → `libvlc_new("... --vout fb ...")` in *vlc/src/control/core.c*
    - → `libvlc_InternalCreate()`, `libvlc_InternalInit()` in *vlc/src/libvlc.c*
      - → `playlist_ThreadCreate()` in *vlc/src/playlist/thread.c*
        - → `playlist_PlayItem()` in *vlc/src/playlist/control.c*
          - → `input_CreateThreadExtended()` in *vlc/src/input/input.c*
            - → `input_DecoderNew()` in *vlc/src/input/decoder.c*
              - → `vout_Create()` in *vlc/src/video_output/video_output.c*
                - → `module_Need()` in *vlc/src/modules/modules.c*
                  - → `module_Call()` in *vlc/src/modules/os.c*
                    - → `vlc_entry()` in *vlc/modules/video_output/fb.c*
                    - → `Create()` in *vlc/modules/video_output/fb.c*
                      - → **fbmem = mmap()** on **/dev/fb0**
              - → Decoding (for example by **Libavcodec** from FFmpeg)
              - → `Display()` in *vlc/modules/video_output/fb.c* → copy to **fbmem**

Contents

GLUT



- GLUT https://gitlab.freedesktop.org/mesa/glut

  Linux Framebuffer support in *src/glut/fbdev* directory

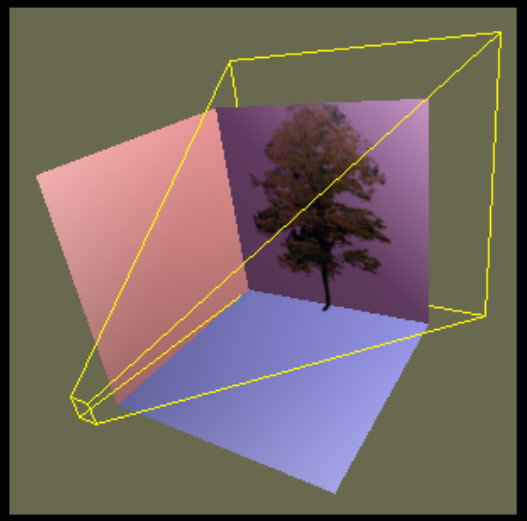
  Note: EGL for Linux Framebuffer could be used instead of GLFBDev

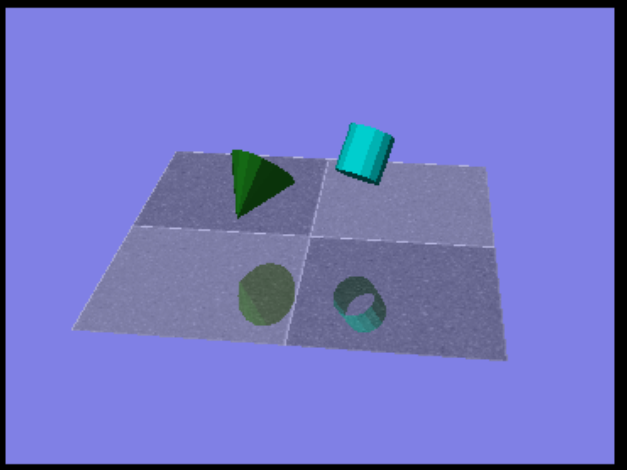
- Examples
  - mesa-demos https://gitlab.freedesktop.org/mesa/demos

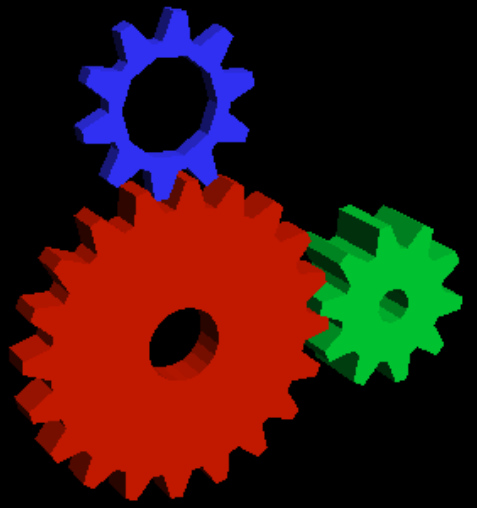
    ➡ projtex, reflect, ... in *src/demos* directory
  - yagears https://github.com/caramelli/yagears

GLUT demo

```
~$ GLUT_WINDOW_POS=80,240 projtex &
~$ GLUT_WINDOW_POS=520,120 reflect &
~$ WIDTH=360 HEIGHT=360 POSX=600 POSY=400 yagears-gui -t glut -e gl &
~$
```







GLUT internal

- `glutInit()`
- `glutCreateWindow()`

  → in *MesaGLUT/src/glut/fbdev/fbdev.c*

  fbmem = mmap() on **/dev/fb0**
  glFBDevCreateBuffer(), glFBDevSetWindow(struct fb_window *)
  glFBDevCreateContext(), glFBDevMakeCurrent()

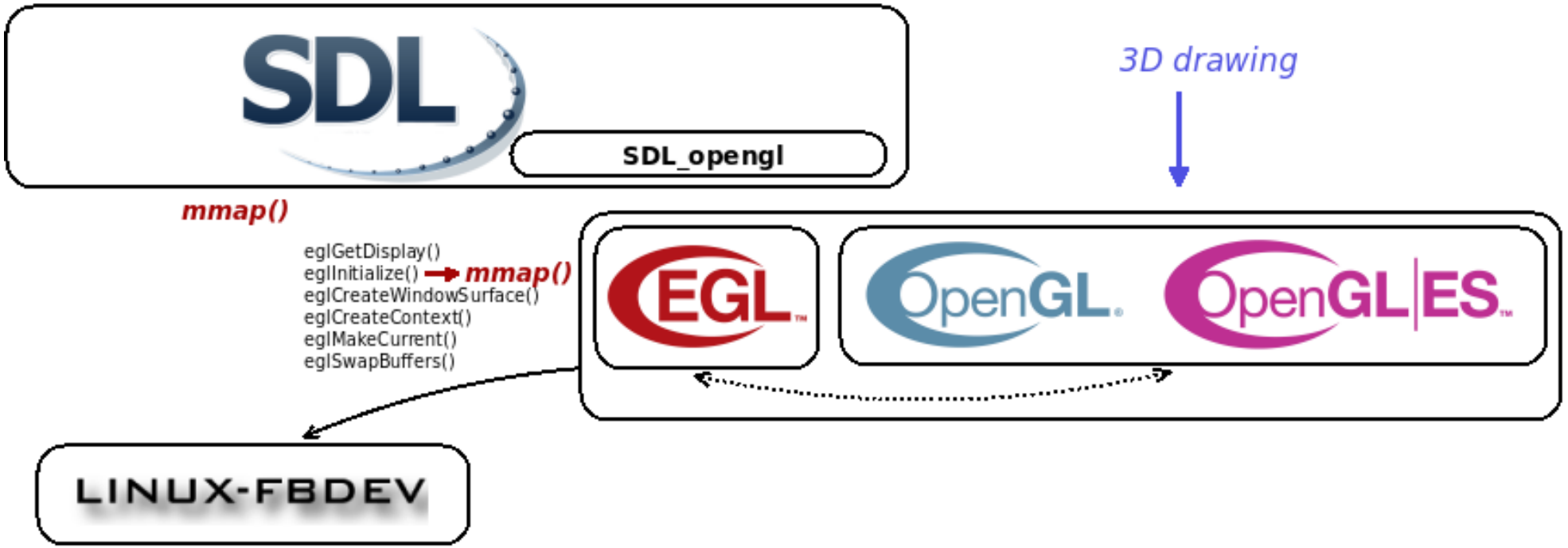
- `glutSwapBuffers()`

  → in *MesaGLUT/src/glut/fbdev/fbdev.c*

  glFBDevSwapBuffers()

  → copy of 3D drawing based on OpenGL API to **fbmem**

SDL



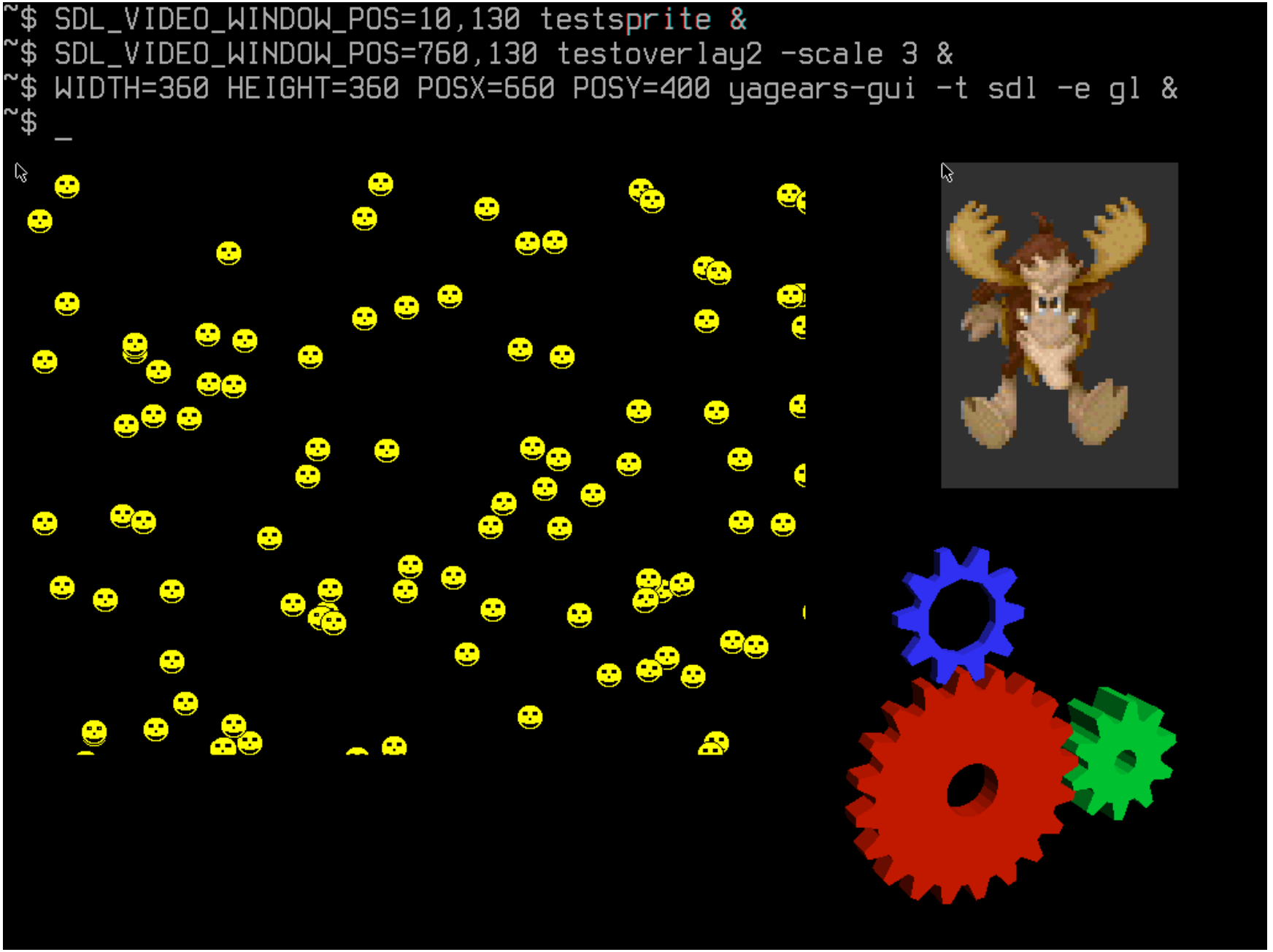
- SDL https://hg.libsdl.org/SDL

  Linux Framebuffer support in *src/video/fbcon* directory

  Note: GLFBDev extension could be used instead of EGL

- Examples
  - testsprite, testoverlay2, ... in *test* in directory
  - yagears https://github.com/caramelli/yagears

SDL demo


~$ SDL_VIDEO_WINDOW_POS=10,130 testsprite &
~$ SDL_VIDEO_WINDOW_POS=760,130 testoverlay2 -scale 3 &
~$ WIDTH=360 HEIGHT=360 POSX=660 POSY=400 yagears-gui -t sdl -e gl &
~$ _

SDL internal

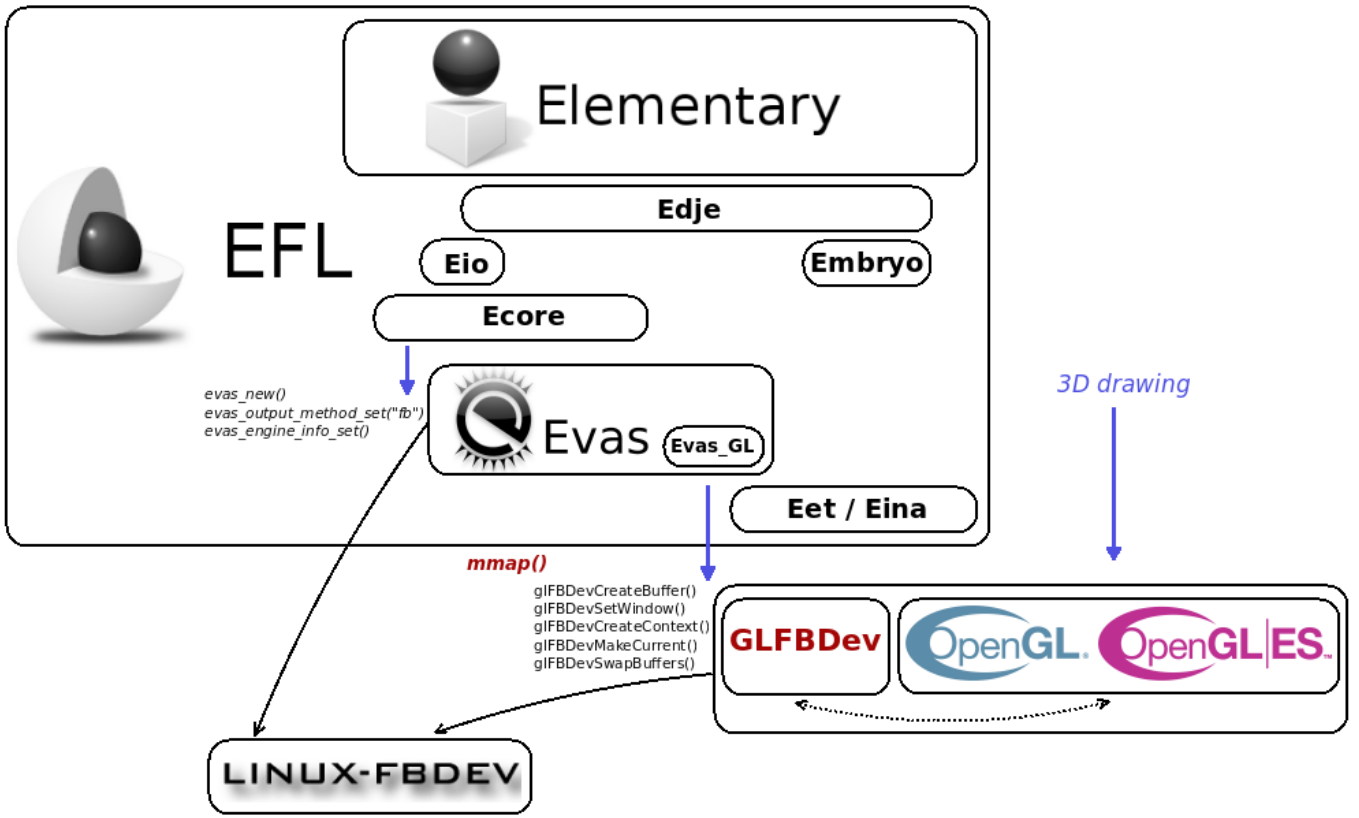
- `SDL_Init()`
  - → `SDL_VideoInit()` in SDL/src/video/SDL_video.c
    - → `FB_VideoInit()` in *SDL/src/video/fbcon/SDL_fbvideo.c*
      - → **fbmem = mmap()** on **/dev/fb0**
- `SDL_SetVideoMode()` → Set up window for drawing
  - → `FB_SetVideoMode()` in *SDL/src/video/fbcon/SDL_fbvideo.c*

    For OpenGL
    - → **eglGetDisplay(), eglInitialize(), eglCreateWindowSurface(), eglCreateContext()**

    For OpenGL
  - → `FB_GL_MakeCurrent()` in *SDL/src/video/fbcon/SDL_fbvideo.c*
    - → **eglMakeCurrent()**
- `SDL_Flip(), SDL_UpdateRects()`
  - → `SDL_LowerBlit()` in *SDL/src/video/SDL_surface.c*
    - → `SDL_SoftBlit()` in *SDL/src/video/SDL_blit.c* → copy to **fbmem**
- **For OpenGL**, `SDL_GL_SwapBuffers()`
  - → `FB_GL_SwapBuffers()` in *SDL/src/video/fbcon/SDL_fbvideo.c*
    - → **eglSwapBuffers()** → copy of 3D drawing based on OpenGL API to **fbmem**

Contents

EFL



- EFL

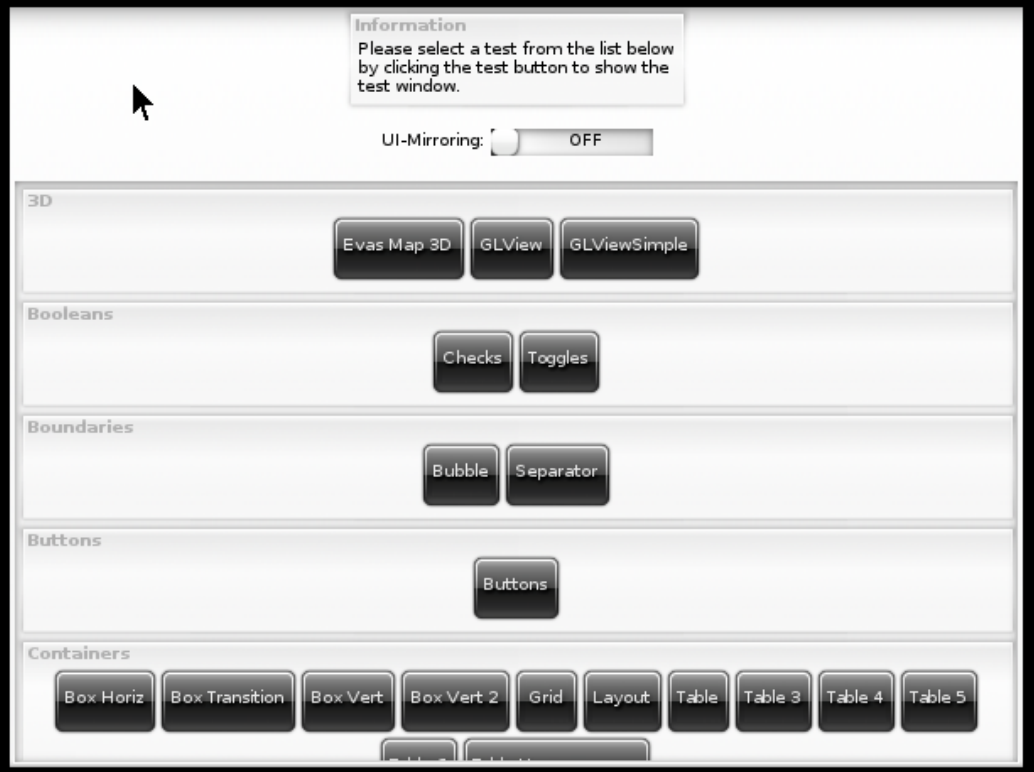
  https://git.enlightenment.org/core/elementary.git/

  https://git.enlightenment.org/core/efl.git/ or https://git.enlightenment.org/legacy/{ecore.git,evas.git,...}/
  - Linux Framebuffer support in ecore/*src/lib/ecore_evas/ecore_evas_fb.c* and ecore/*src/lib/ecore_fb* directory
  - Evas_GL interfaces in evas/src/lib/canvas/evas_gl.c

    Note: EGL for Linux Framebuffer could be used instead of GLFBDev
- Examples
  - elementary_test in *elementary/src/bin* in directory
  - yagears https://github.com/caramelli/yagears

EFL demo

```
~$ EVAS_FB_POS=20,160 elementary_test &
~$ WIDTH=360 HEIGHT=360 POSX=640 POSY=200 yagears-gui -t efl -e gl &
~$ _
```



EFL internal

EFL

- `elm_init()`
- `elm_win_add()` → toplevel window for widgets drawing
  - → `ecore_evas_fb_new()` in *ecore/src/lib/ecore_evas/ecore_evas_fb.c*
    - → **evas_new(), evas_output_method_set("fb"), evas_engine_info_set()**
      - → **fbmem = mmap()** on **/dev/fb0**
- `elm_run()`
  - → `ecore_main_loop_begin()` in *ecore/src/lib/ecore/ecore_main.c*
    - → `ecore_evas_idle_enter()` in *ecore/src/lib/ecore_evas/ecore_evas.c*
      - → `ecore_evas_fb_render()` in *ecore/src/lib/ecore_evas/ecore_evas_fb.c*
        - → **evas_render_updates()**
          - → `output_redraws_next_update_push()` in *evas/engines/fb/evas_engine.c*
            - → copy to **fbmem**

EFL internal

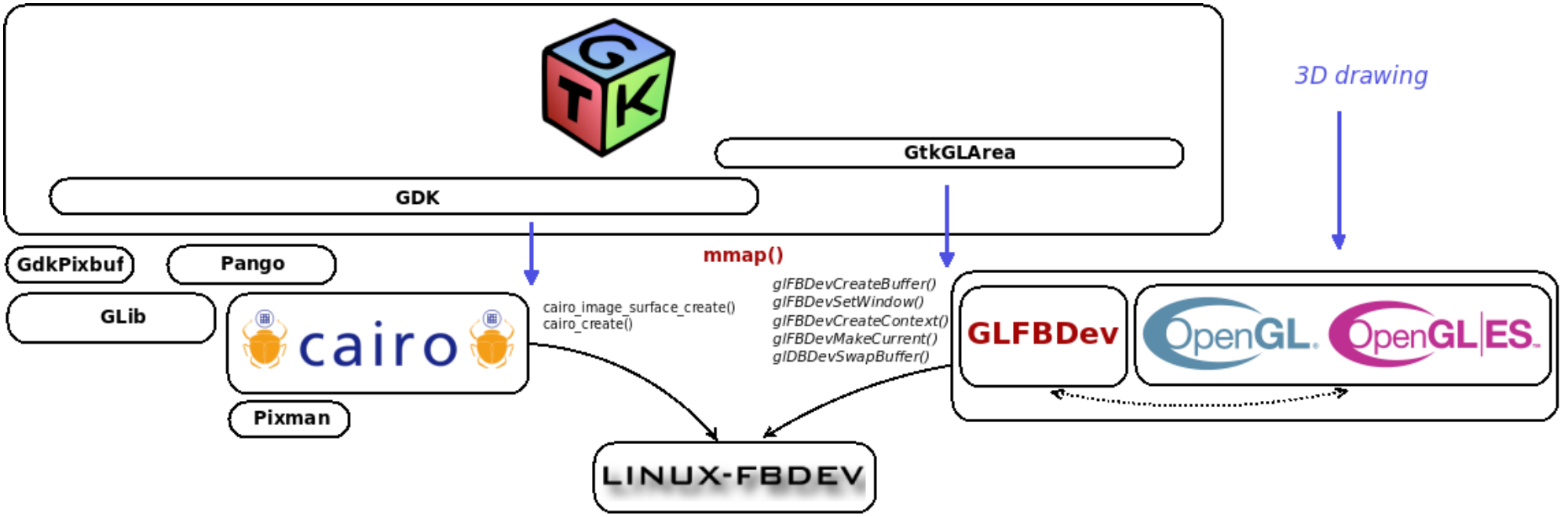
EFL

For OpenGL

- `elm_glview_add()`
  - → `evas_gl_surface_create()`
    - → `gl_surface_create()` in *evas/engines/fb/evas_engine.c*
      - → **glFBDevCreateBuffer()**
  - → `evas_gl_context_create()`
    - → `gl_context_create()` in *evas/engines/fb/evas_engine.c*
      - → **glFBDevCreateBuffer()**
- `elm_glview_changed_set()`
  - → `evas_gl_make_current()`
    - → `gl_make_current()` in *evas/engines/fb/evas_engine.c*
      - → **glFBDevMakeCurrent()**
  - → `evas_object_image_pixels_dirty_set()` in *evas/src/lib/canvas/evas_object_image.c*
    - → `output_redraws_next_update_push()` in *evas/engines/fb/evas_engine.c*
      - → **glFBDevSetWindow(struct fb_window *), glFBDevSwapBuffers()**

→ copy of 3D drawing based on OpenGL API to **fbmem**

GTK+

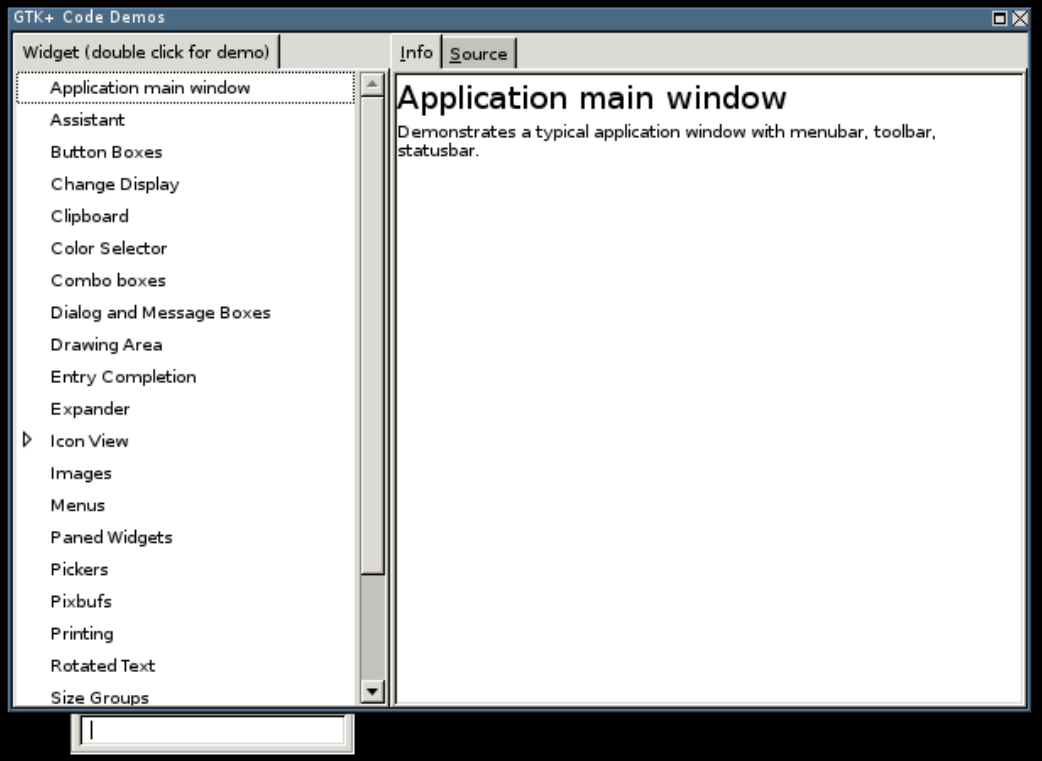


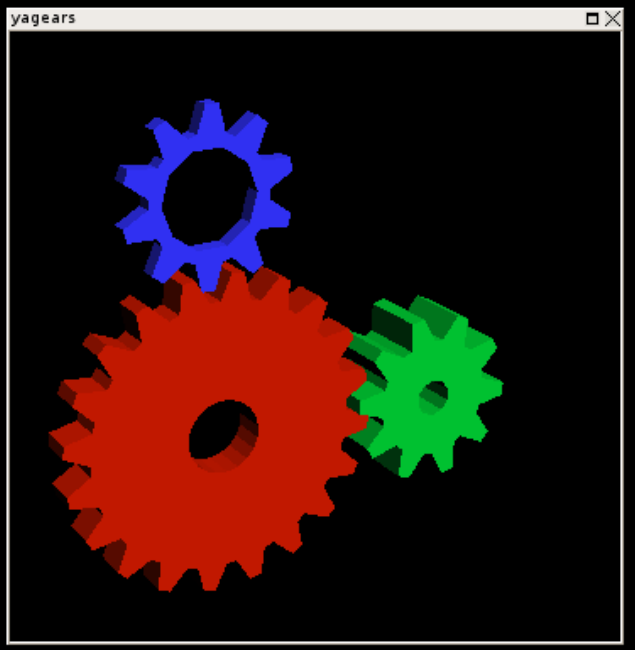
- GTK+ https://gitlab.gnome.org/GNOME/gtk
  - Linux Framebuffer support in *gdk/linux-fb* directory
  - GtkGLArea interfaces in *gtkgl/gtkglarea.c*

    Note: EGL for Linux Framebuffer could be used instead of GLFBDev
- Examples
  - gtk-demo in *demos/gtk-demo* directory
  - yagears https://github.com/caramelli/yagears

GTK+ demo

```
~$ GDK_WINDOW_POS=20,160 gtk-demo &
~$ WIDTH=360 HEIGHT=360 POSX=640 POSY=200 yagears-gui -t gtk -e glesv2 &
~$
```





GTK+ internal

- `gtk_init()`
  - → `gdk_pre_parse_libgtk_only()` in *gtk/gdk/gdk.c*
    - → `gdk_fb_display_new()` in *gtk/gdk/linux-fb/gdkmain-fb.c*
      - → **fbmem = mmap()** on **/dev/fb0**
- `gtk_window_new()` → toplevel window for widgets drawing
- `gtk_main()`
  - → `gdk_drawable_ref_cairo_surface()` in *gtk/gdk/gdkdraw.c*
    - → `gdk_fb_ref_cairo_surface()` in *gtk/gdk/linux-fb/gdkdrawable-fb2.c*
      - → **cairo_image_surface_create()**
  - → `gdk_draw_drawable()` in *gtk/gdk/gdkdraw.c*
    - → `gdk_fb_draw_drawable()` in *gtk/gdk/linux-fb/gdkdrawable-fb2.c*
      - → `gdk_fb_draw_drawable_memmove()` in *gtk/gdk/linux-fb/gdkrender-fb.c*
        - → **cairo_image_surface_get_data()**

→ copy to **fbmem**

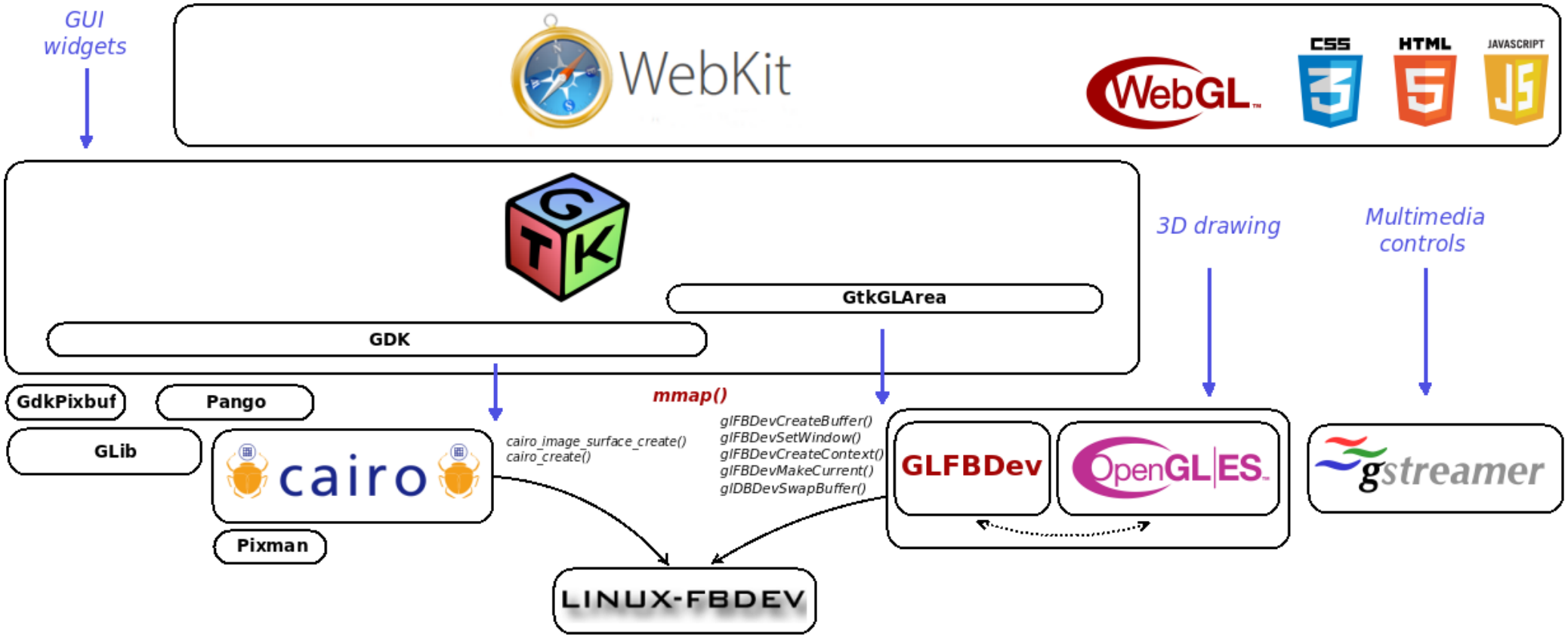
GTK+ internal

For OpenGL

- `gtk_gl_area_new()`
  - → `gdk_gl_context_share_new()` in *gtk/gtkgl/gdkgl.c*
    - → **glFBDevCreateBuffer(), glFBDevCreateContext()**
- `gtk_gl_area_make_current()`
  - → `gdk_gl_make_current()` in *gtk/gtkgl/gdkgl.c*
    - → **glFBDevMakeCurrent()**
- `gtk_gl_area_swap_buffers()`
  - → `gdk_gl_swap_buffers()` in *gtk/gtkgl/gdkgl.c*
    - → **glFBDevSetWindow(struct fb_window *), glFBDevSwapBuffers()**

→ copy of 3D drawing based on OpenGL API to **fbmem**

WebKitGTK+



WebKitGTK+ https://svn.webkit.org/repository/webkit/releases/WebKitGTK

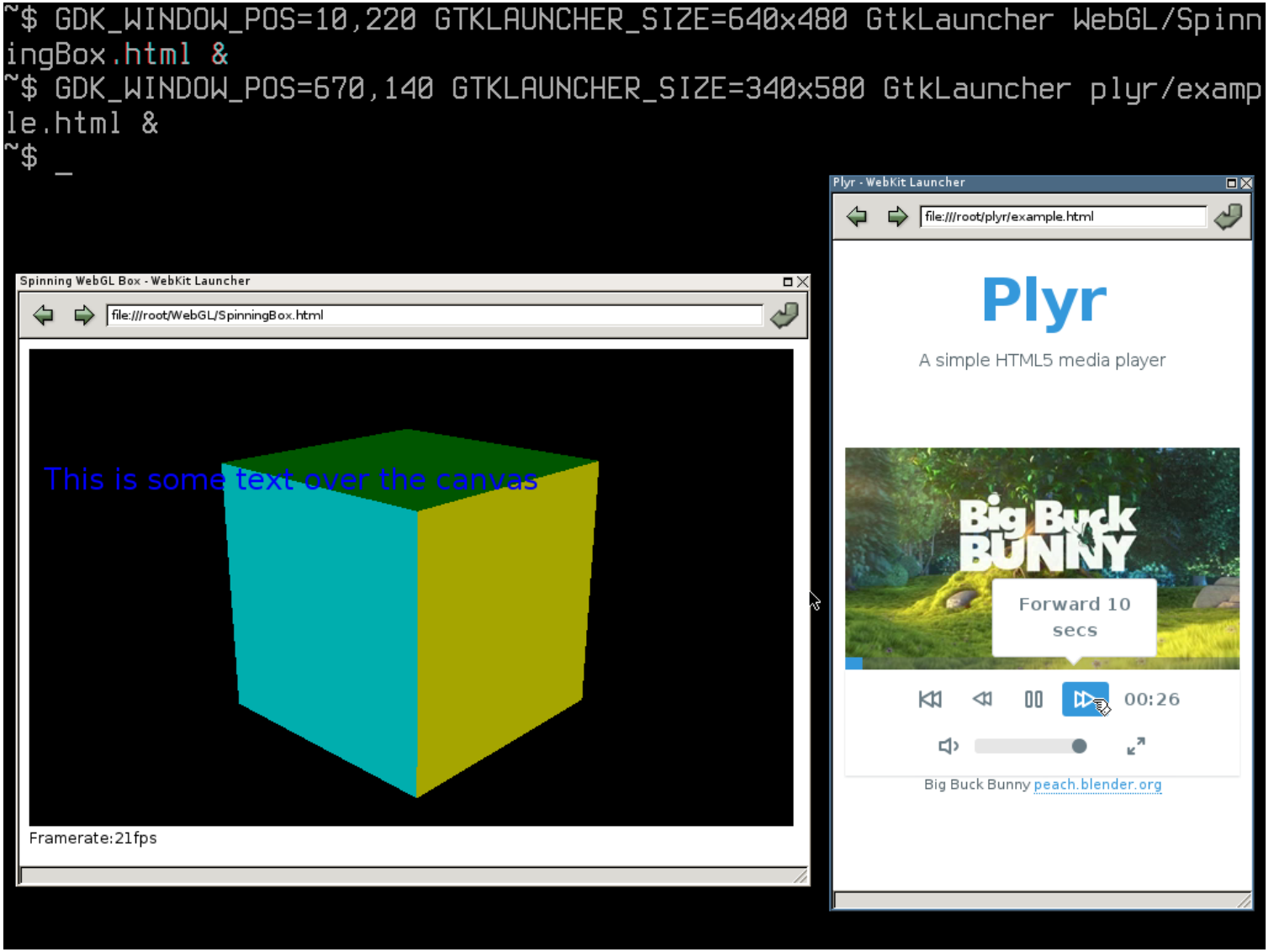
→ *Tools/GtkLauncher*

→ *Source/WebKit/gtk* ➡ WebView widget

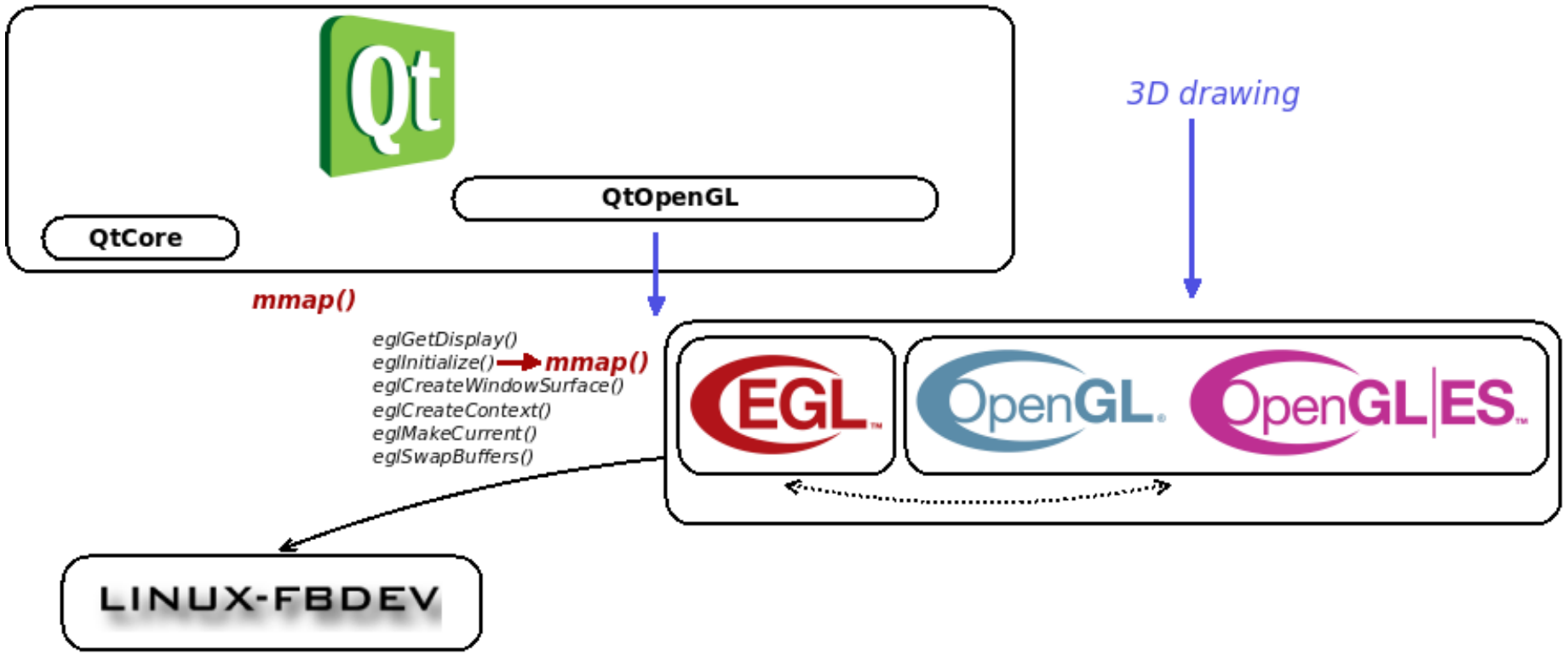
→ *Source/WebCore/platform/{gtk, graphics/gtk, graphics/gstreamer}*

WebKitGTK+ demo

- WebGL demos https://github.com/KhronosGroup/WebGL
- Plyr HTML5 media player https://github.com/sampotts/plyr



Qt

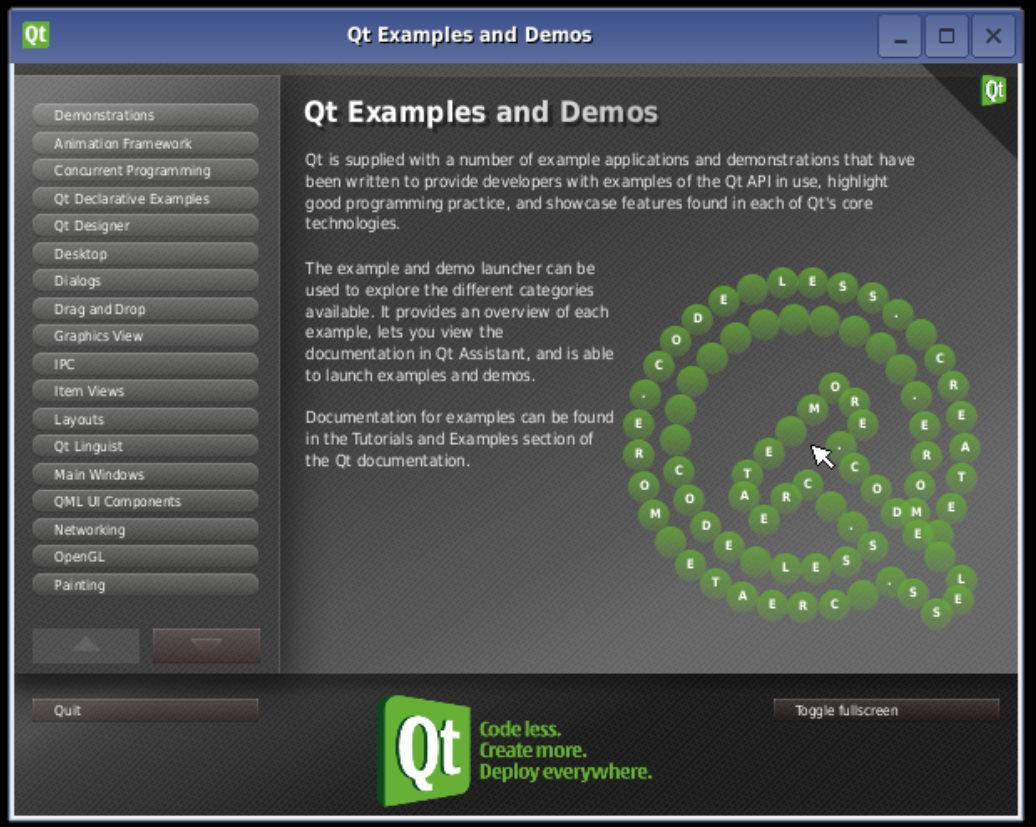


- Qt 4 https://code.qt.io/cgit/qt/qt.git and QT 5 https://code.qt.io/cgit/qt/qtbase.git
  - Linux Framebuffer support for Qt 4 in *src/plugins/gfxdrivers/linuxfb* directory → **QWS** (Qt Windowing System)
  - Linux Framebuffer support for Qt 5 in *src/plugins/platforms/linuxfb* directory → **QPA** (Qt Platform Abstraction)
  - QtOpenGL interfaces in *src/opengl/qgl.cpp*

    Note: GLFBDev extension could be used instead of EGL
- Examples
  - qtdemo in *demos/qtdemo* directory
  - yagears https://github.com/caramelli/yagears

Qt demo

```
~$ QWS_POS=20,160 qtdemo &
~$ WIDTH=360 HEIGHT=360 POSX=640 POSY=200 yagears-gui -t qt -e glesv2 &
~$ _
```



Qt internal

- `QApplication::QApplication()`
  - → `qt_init()` in *qt/src/gui/kernel/qapplication_qws.cpp*
    - → `QWSServer::startup()` in qt/*src/gui/embedded/qwindowsystem_qws.cpp*
      - → `qt_init_display()` in *qt/src/gui/kernel/qapplication_qws.cpp*
        - → `qt_get_screen()` in *qt/src/gui/embedded/qscreen_qws.cpp*
          - → `QscreenDriverFactory::create()` in *qt/src/gui/embedded/qscreendriverfactory_qws.cpp*
            - → `QScreenLinuxFbPlugin::create()` in *qt/src/plugins/gfxdrivers/linuxfb/main.cpp*
              - → `QlinuxFbScreen::QlinuxFbScreen()` in *qt/src/gui/embedded/qscreenlinuxfb_qws.cpp*
          - → `QLinuxFbScreen::connect()` in *qt/src/gui/embedded/qscreenlinuxfb_qws.cpp*
            - → **fbmem = mmap()** on **/dev/fb0**

- `QWidget` → class for widgets drawing

- `Qapplication::exec()`
  - → *qt/src/gui/painting/qdrawhelper.cpp* → copy to **fbmem**

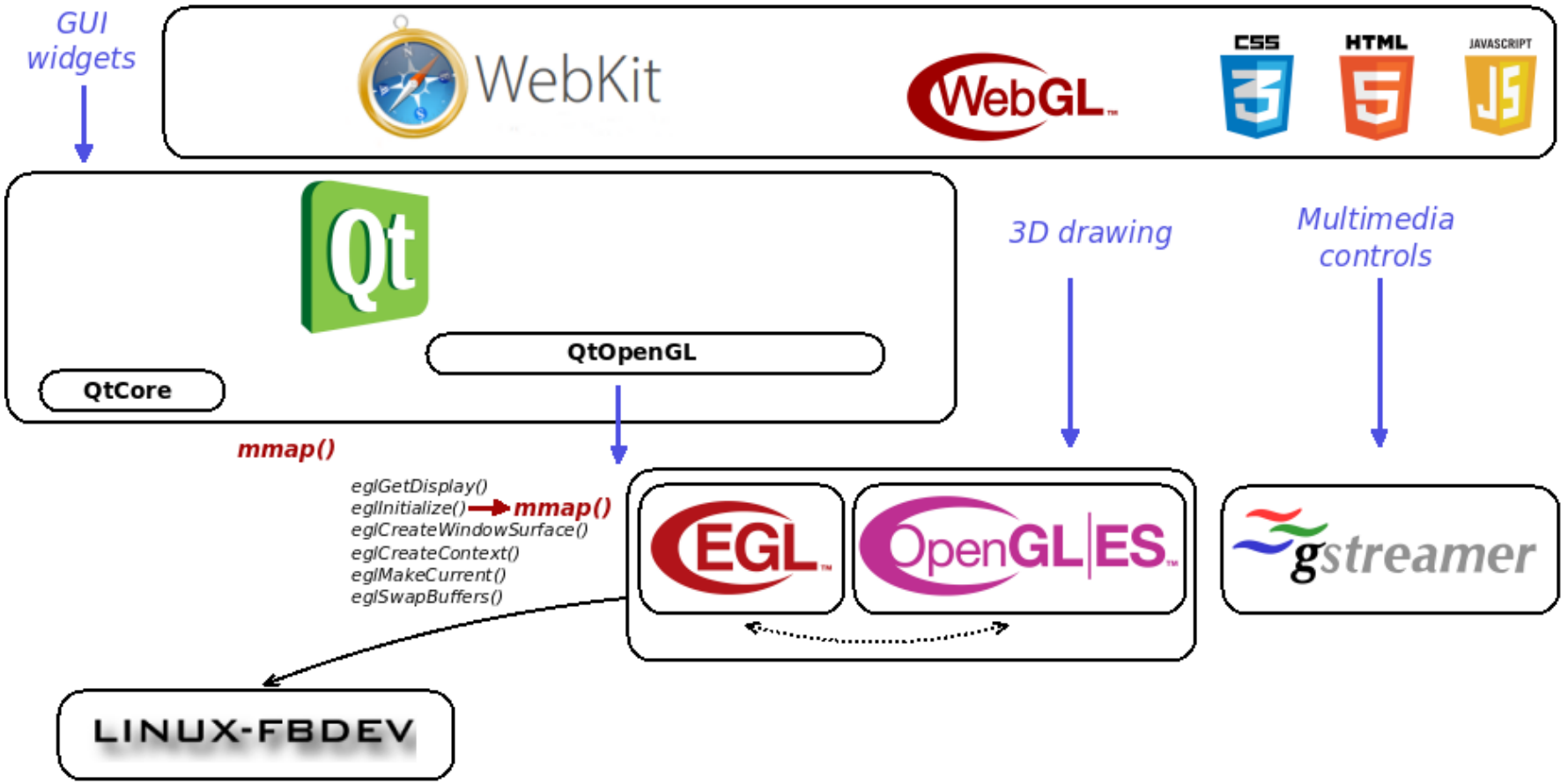
Qt internal

For OpenGL

- `QGLWidget` → class for OpenGL rendering
  - → `QEglContext::chooseConfig()` in *qt/src/gui/egl/qegl.cpp*
    - → **eglGetDisplay(), eglInitialize()**
  - → `QEglContext::createContext()` in *qt/src/gui/egl/qegl.cpp*
    - → **eglCreateContext()**
  - → `QGLWidget::glInit()` in *qt/src/opengl/qgl.cpp*
    - → `qt_egl_create_surface()` in *qt/src/opengl/qgl_qws.cpp*
      - → **struct fb_window *** `QlinuxFbScreenSurfaceFunctions::createNativeWindow()` in *qt/src/gui/embedded/qscreenlinuxfb_qws.cpp*
      - → **eglCreateWindowSurface(struct fb_window *)**

- `QGLWidget::updateGL()`
  - → `QeglContext::makeCurrent()` in *qt/src/gui/egl/qegl.cpp*
    - → **eglMakeCurrent()**
  - → `QEglContext::swapBuffers()` in *qt/src/gui/egl/qegl.cpp*
    - → **eglSwapBuffers()** → copy of 3D drawing based on OpenGL API to **fbmem**

QtWebKit



QtWebKit for QT 4 https://gitorious.org/webkit/qtwebkit or https://gitorious.org/webkit/qtwebkit-23

QtWebKit for QT 5 https://code.qt.io/cgit/qt/qtwebkit.git

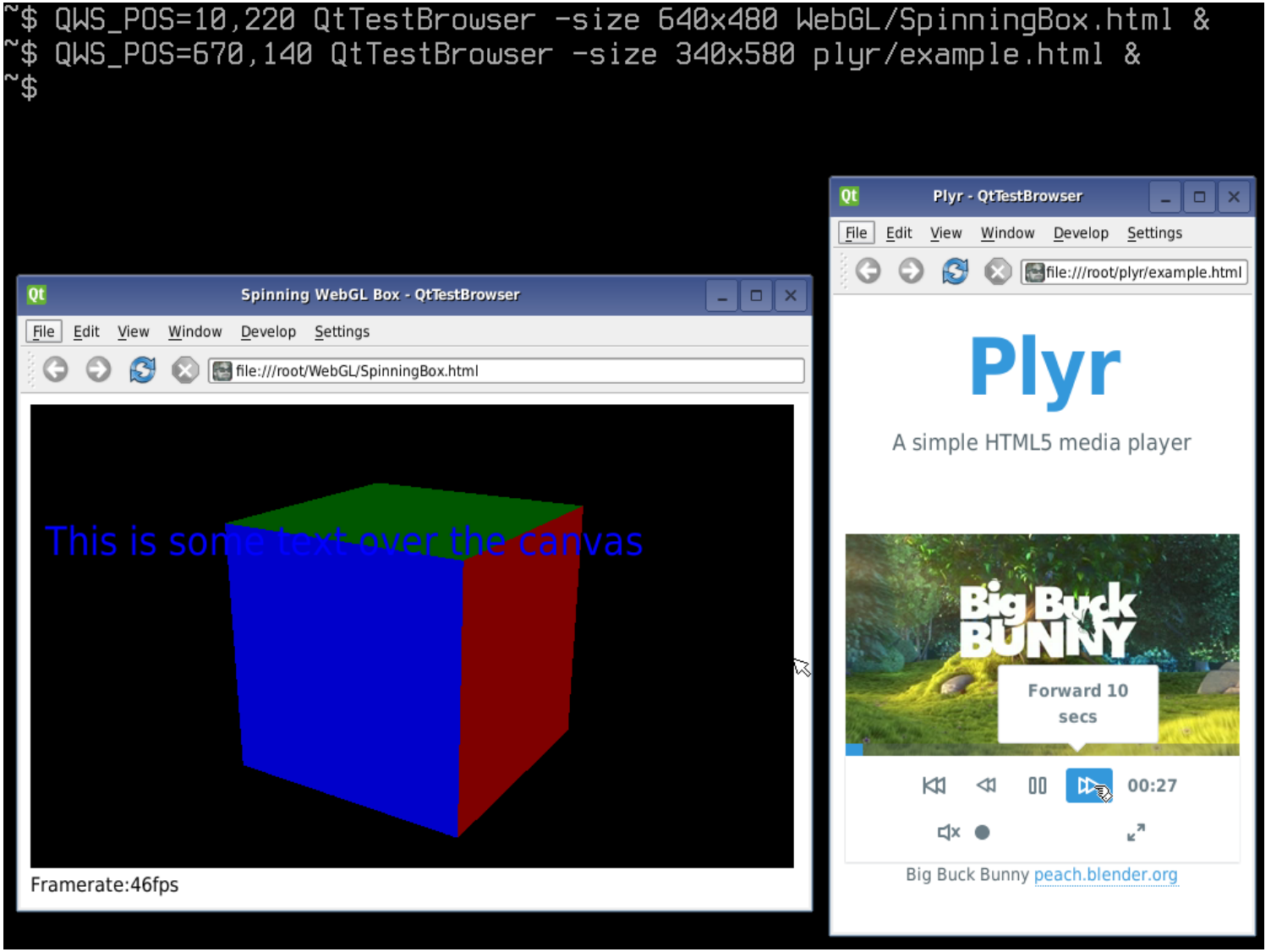
→ *Tools/QtTestBrowser*

→ *Source/WebKit/qt* ➡ WebView widget

→ *Source/WebCore/platform/{qt, graphics/qt, graphics/gstreamer}*

QtWebKit demo

- WebGL demos https://github.com/KhronosGroup/WebGL
- Plyr HTML5 media player https://github.com/sampotts/plyr



Contents

Running DirectFB, X11 or Wayland on top of the Linux Framebuffer

DirectFB https://github.com/deniskropp/DirectFB

→ Linux Framebuffer support in *systems/fbdev* directory

X11 https://gitlab.freedesktop.org/xorg/driver/xf86-video-fbdev

→ Xorg Device Dependent X (DDX) for Linux Framebuffer

Wayland https://gitlab.freedesktop.org/wayland

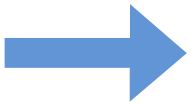
→ Linux Framebuffer support in *weston/src/compositor-fbdev.c*

But it's another story ...

Origins of this presentation ?

- Started a Linux from scratch distribution in 2005 for understanding how graphics work, and have continued to play with for the past 15 years

- Realized that some graphics backends will not be maintained anymore, and now belong to the past



HiGFXback (History of graphics backends) project in order to preserve them

- More infos on https://github.com/caramelli/higfxback/wiki