Enlightenment Foundation Libraries

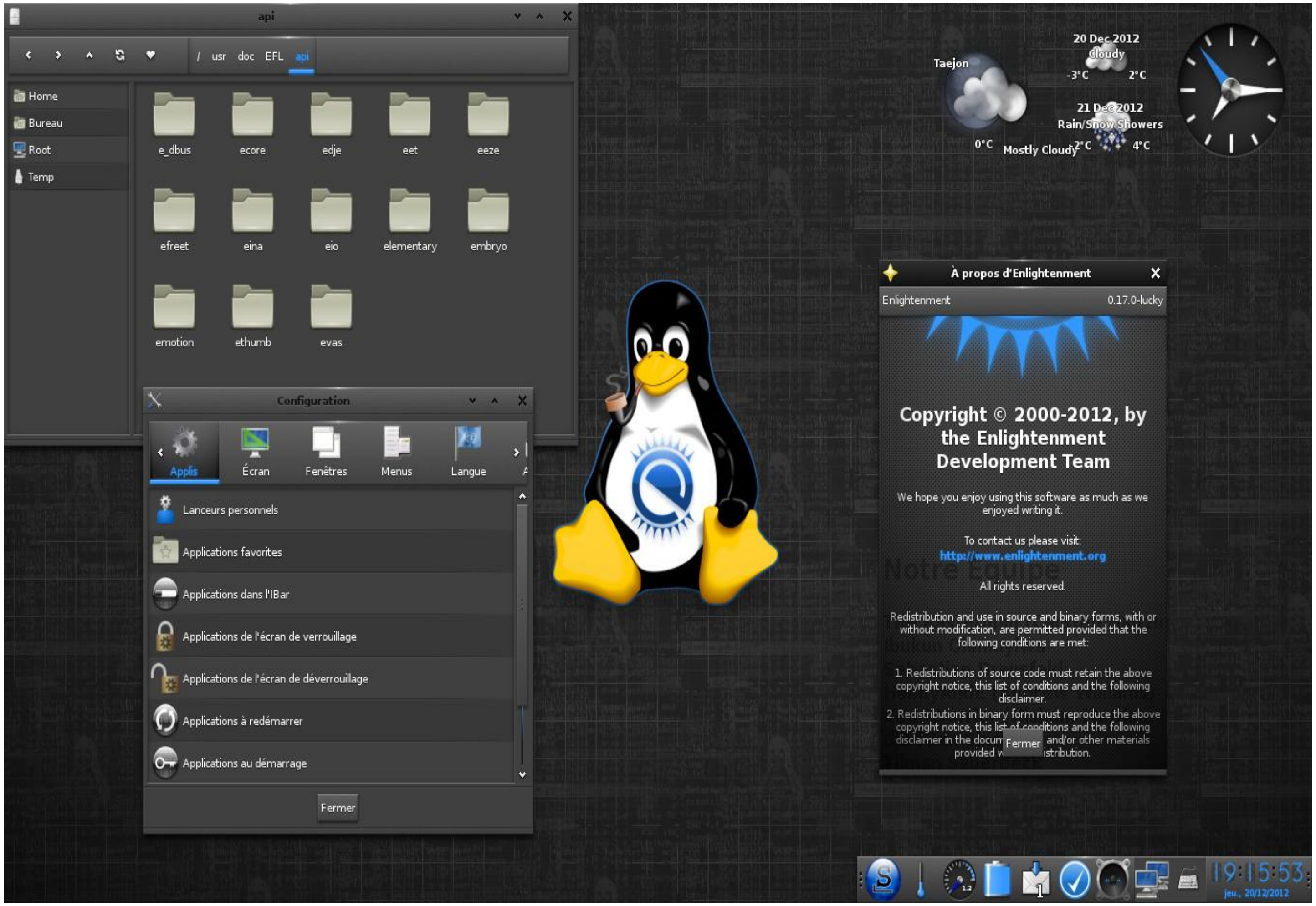
New Vector Graphics API For Designing User Interfaces

Cedric Bail
Senior Graphics Developer
Samsung Open Source Group
cedric@osg.samsung.com

- EFL, in short !
- Vector graphics for user interface
- Designing a modern rendering pipeline for vector graphics
- Questions?

EFL, in short !

EFL: A Toolkit Created for Enlightenment 17



À propos d'Enlightenment

Enlightenment 0.17.0-lucky

Copyright © 2000-2012, by the Enlightenment Development Team

We hope you enjoy using this software as much as we enjoyed writing it.

To contact us please visit: http://www.enlightenment.org

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the docum Fermer and/or other materials provided w istribution.

Enlightenment Foundation Libraries (*EFL*)

- Spent a decade writing a modern graphic toolkit
- Licensed under a mix of LGPL and BSD license
- Focus on embedded devices, but scale from the low end to the high end.
- First release on January 2011
- Stable, long term API/ABI
- In the process of releasing version 1.16
- 3 month release cycle

Enlightenment Community

- The Enlightenment community
  - 60 uniq contributors per release (10 cores)
  - 1000 users that build from source
  - 2 distributions based on Enlightenment (Bodhi and Elive)
- The Enlightenment community expected Linux to takeoff in the embedded world, not on the desktop
- The values shared by this community:
  - Fast
  - Light
  - Feature Rich
  - Customizable
  - Scalable

State of EFL

- Designed for creating a Windows Manager (WM), now used for any type of application
- Has its own scene graph and rendering library
- Optimized to reduce CPU, GPU, memory and battery usage
- Supports international language requirements (LTR/RTL, UTF8)
- Supports all variations of screens and input devices (scale factor)
- Fully Themable (layout of the application included)
- Supports profiles
- Can take up as little as 8MB of space with a minimal set of dependencies
- Has a modular design

Why We Care About Optimization

- Moore's law doesn't apply to battery and memory bandwidth
- Most rendering operations are limited directly by memory bandwidth
- Many embedded devices have less available memory than a low end phone
  - Refrigerator, oven, dish washer, washing machine, home automation…
- Even a low end phone doesn't have much memory to spare once you run a browser!
- GL context at best consumes 10MB, usually more around 40MB; this is bad

for multitasking!

Current State of Optimization

- Application runtime memory use is mostly driven by screen size
- EFL can fit in 8MB on disk (static compilation with minimal dependencies, OS included)
- No hard requirement on the GPU
- Enlightenment + Arch Linux combined :
  - 48 MB RAM
  - 300 Mhz (1024 x 768)
  - Yes, for a desktop profile!

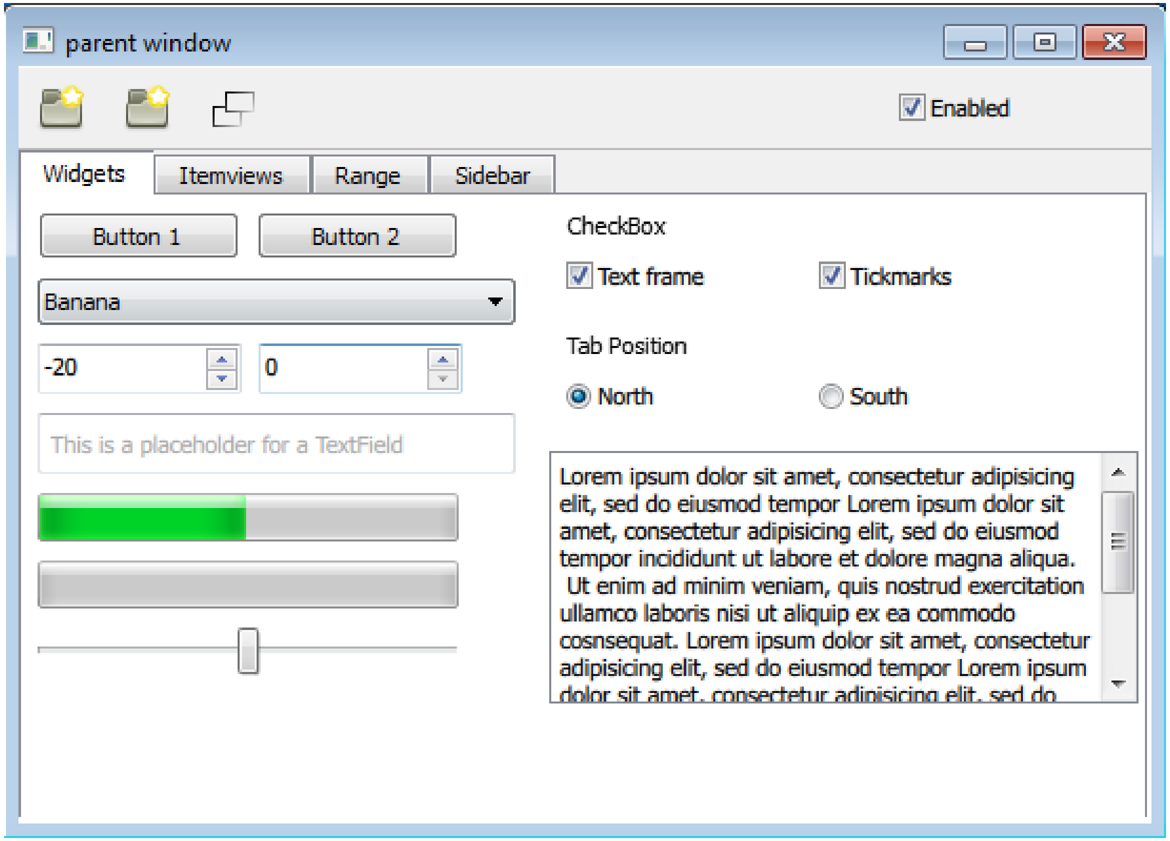
Vector graphics for user interface

Vector graphics quick definition

Wikipedia :

> “*Vector graphics are based on vectors (also called* ***paths****), which lead through locations called* ***control points*** *or* ***nodes****. Each of these points has a definite position on the x and y axes of the work plane and determines the direction of the path; further, each path may be assigned a* ***stroke color****,* ***shape****,* ***thinkness****, and* ***fill****.*”

Let's look at some application and toolkit



☑ Enabled

Banana

-20

0

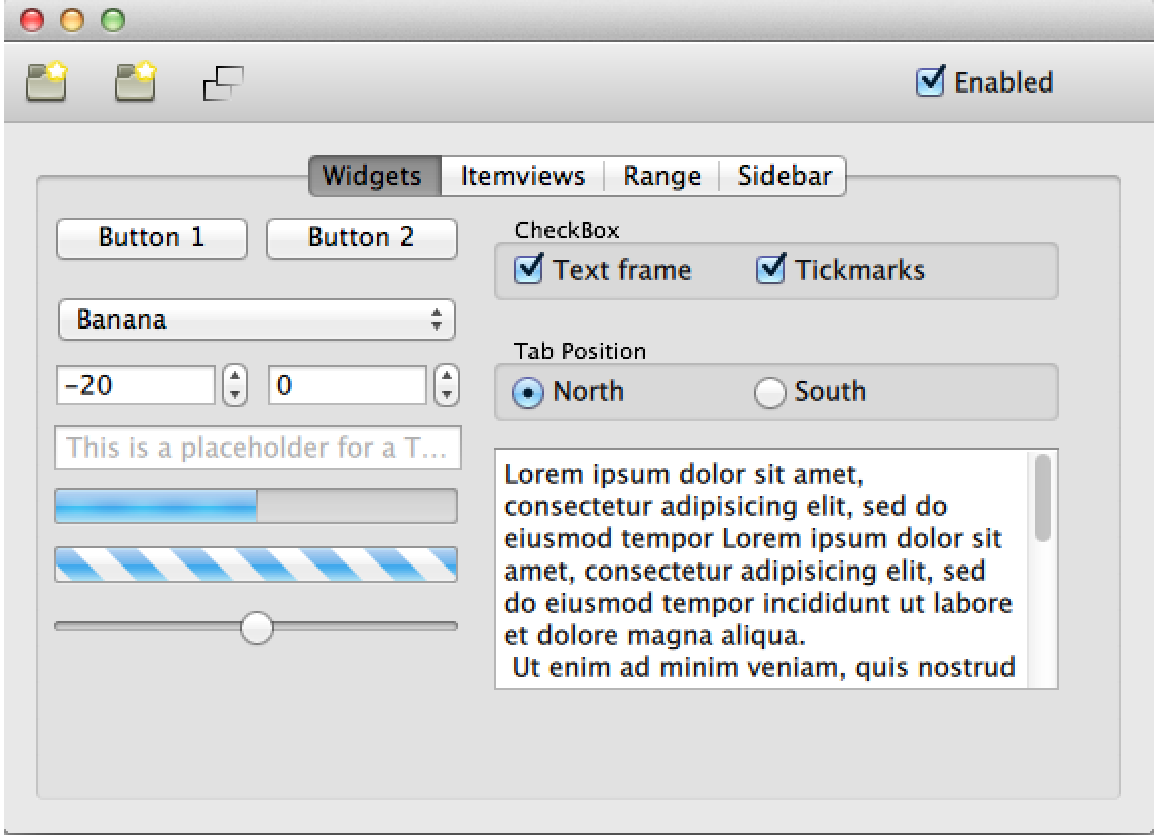
This is a placeholder for a TextField

☑ Text frame ☑ Tickmarks

● North ○ South

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo cosnsequat. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do

Let's look at some application and toolkit



☑ Enabled

Banana

-20

0

This is a placeholder for a T...

CheckBox

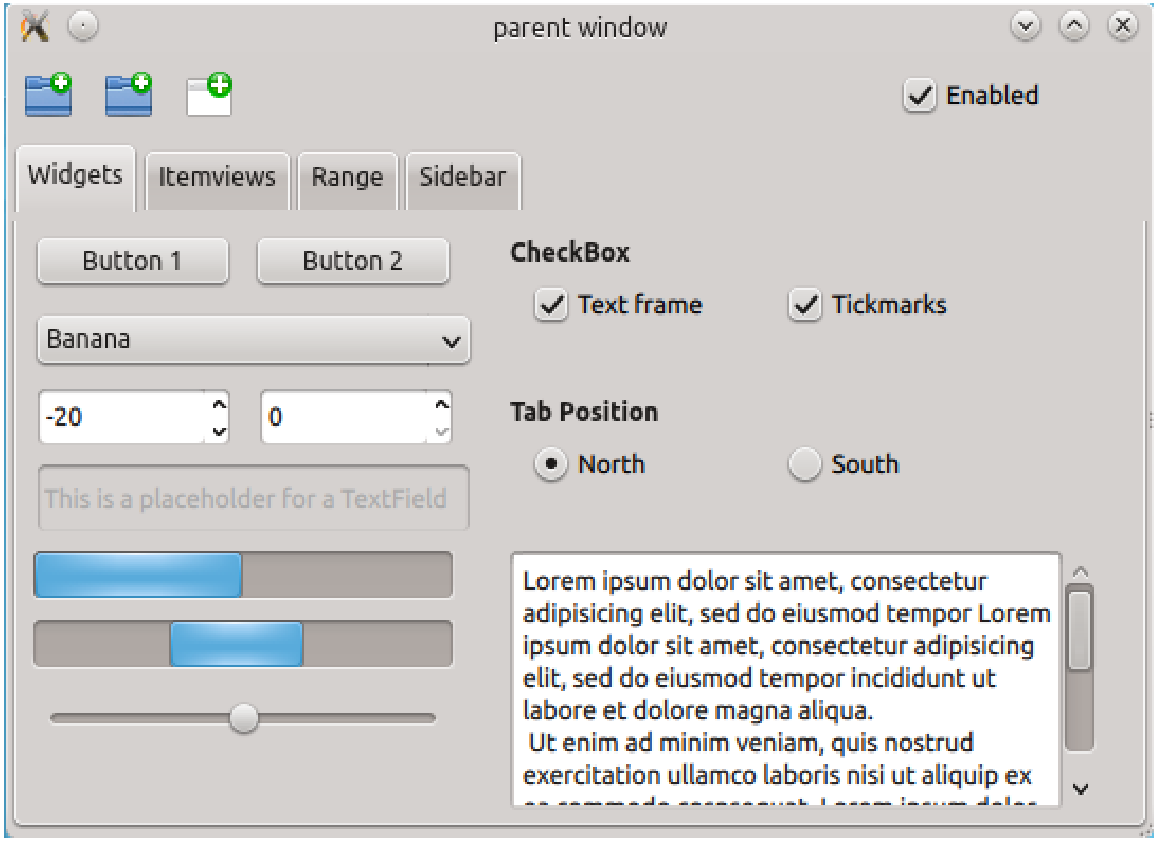
☑ Text frame ☑ Tickmarks

Tab Position

● North ○ South

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud

Let's look at some application and toolkit



☑ Enabled

Banana

-20

0

This is a placeholder for a TextField

CheckBox

☑ Text frame ☑ Tickmarks

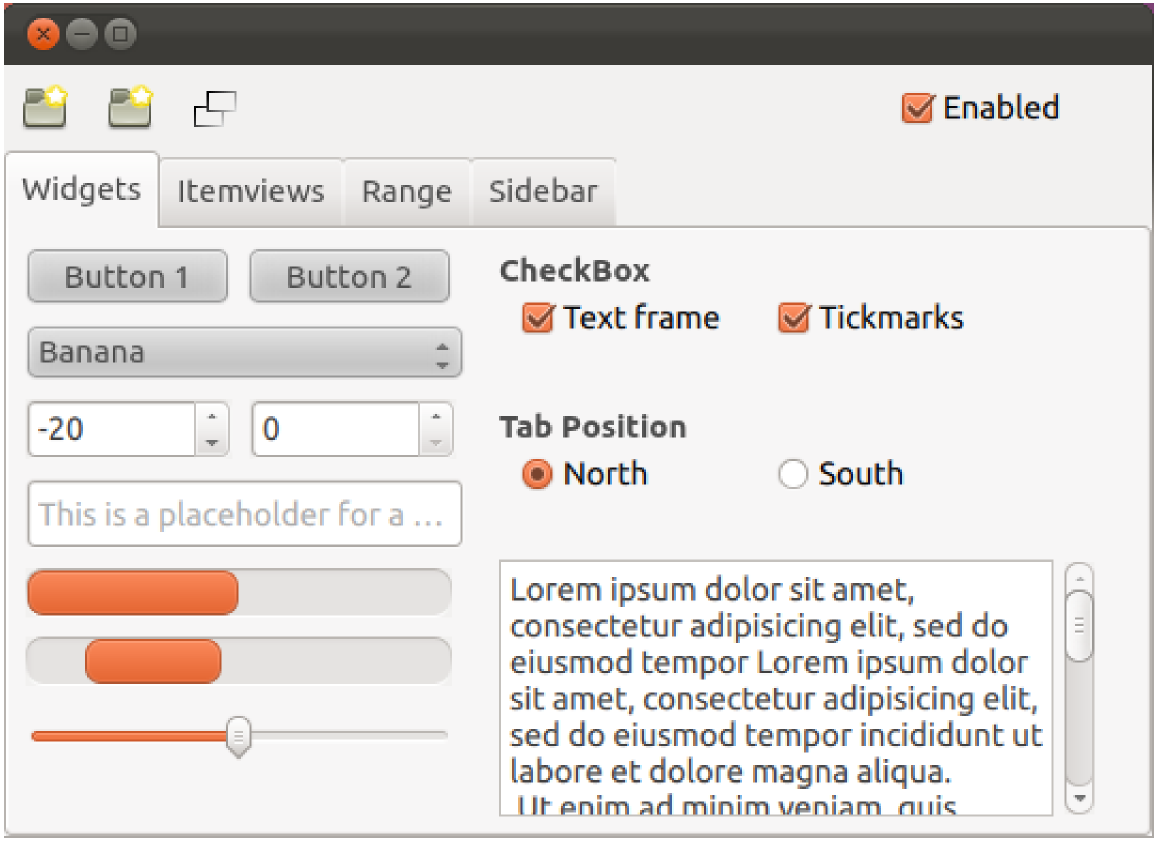
Tab Position

● North ○ South

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex

Let's look at some application and toolkit



☑ Enabled

-20

0

This is a placeholder for a …

CheckBox

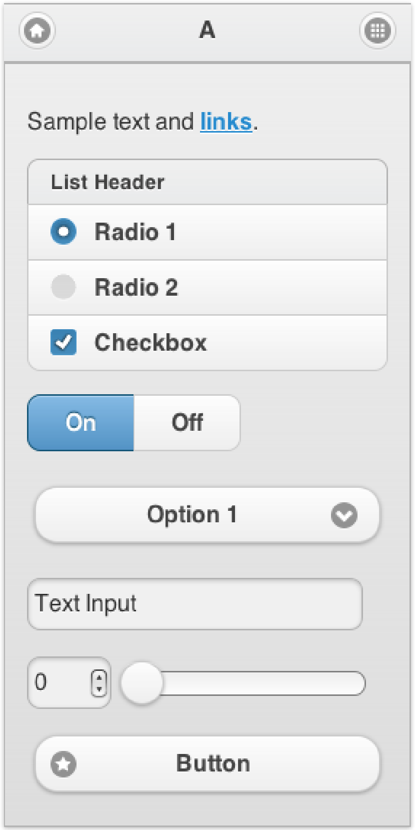
☑ Text frame ☑ Tickmarks

Tab Position

● North ○ South

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis

Let's look at some application and toolkit



Sample text and links.

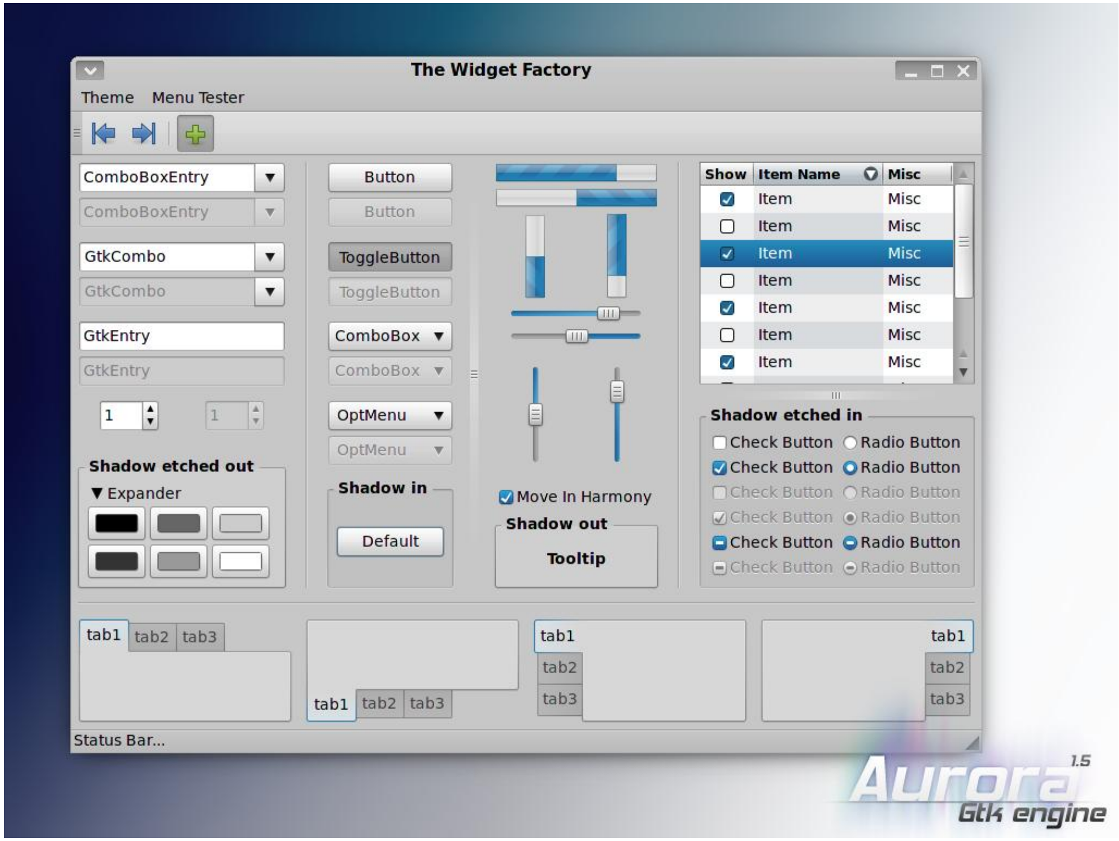
List Header

● Radio 1

○ Radio 2

☑ Checkbox

Let's look at some application and toolkit



Show	Item Name	Misc
☑	Item	Misc
☐	Item	Misc
☑	Item	Misc
☐	Item	Misc
☑	Item	Misc
☐	Item	Misc
☑	Item	Misc

Observation

- Common pattern
  - Same shape
  - Same gradient
  - Same color
- Interface are consistent !
- When we raster image, we reuse the same image everywhere
  → Let's do the same with vector graphics !

Designing a modern rendering pipeline for vector graphics

What should we cache ?

- Caching CPU intensive information
- Minimize amount of memory needed to keep that information
- Minimize memory bandwidth needed to replay that information
- Vector graphics are done in 3 stages :
  - Computing the spans lines (CPU intensive)
  - Filling that spans lines (Depend on operation, but mostly cache bound)
  - Compositing the spans lines (Memory bound operation)
- Cache the spans lines and math, not the generated texture
  → Cache the CPU intensive information without increasing the memory use
- Caching texture result in higher cost during animation

Let's talk a little bit about modern device

- Multi core with different characteristic (big/little)
- Some kind of GPU
- Constrained by memory, because of multi tasking
- Constrained by memory bandwidth
- Constrained by battery
- Everyone expect great and reactive user experience whatever the device
- Everyone want weeks of battery life !

Evas - Scene Graph

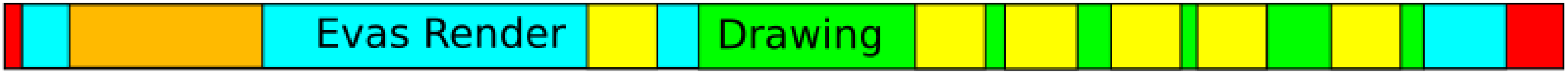
- A basic scene graph rendering ascends from the bottom to the top of the tree
- Possible optimizations
  - Reorder GL operations to limit Texture and Shader changes
  - Partial updates
  - Cutout opaque areas
  - Complete drawing operations in another thread
  - Efficiently cache costly CPU operations between frames
  - Deduplicate objects

Now what can we do with vector graphics in a scenegraph ?

- Possible optimizations :
  - Reorder GL operations to limit Texture and Shader changes
  - Partial updates
  - Cutout opaque areas
  - Complete drawing operations in another thread
  - Efficiently cache costly CPU operations between frames
  - Deduplicate objects

- Vector graphics will always be less efficient than just image rasterizing

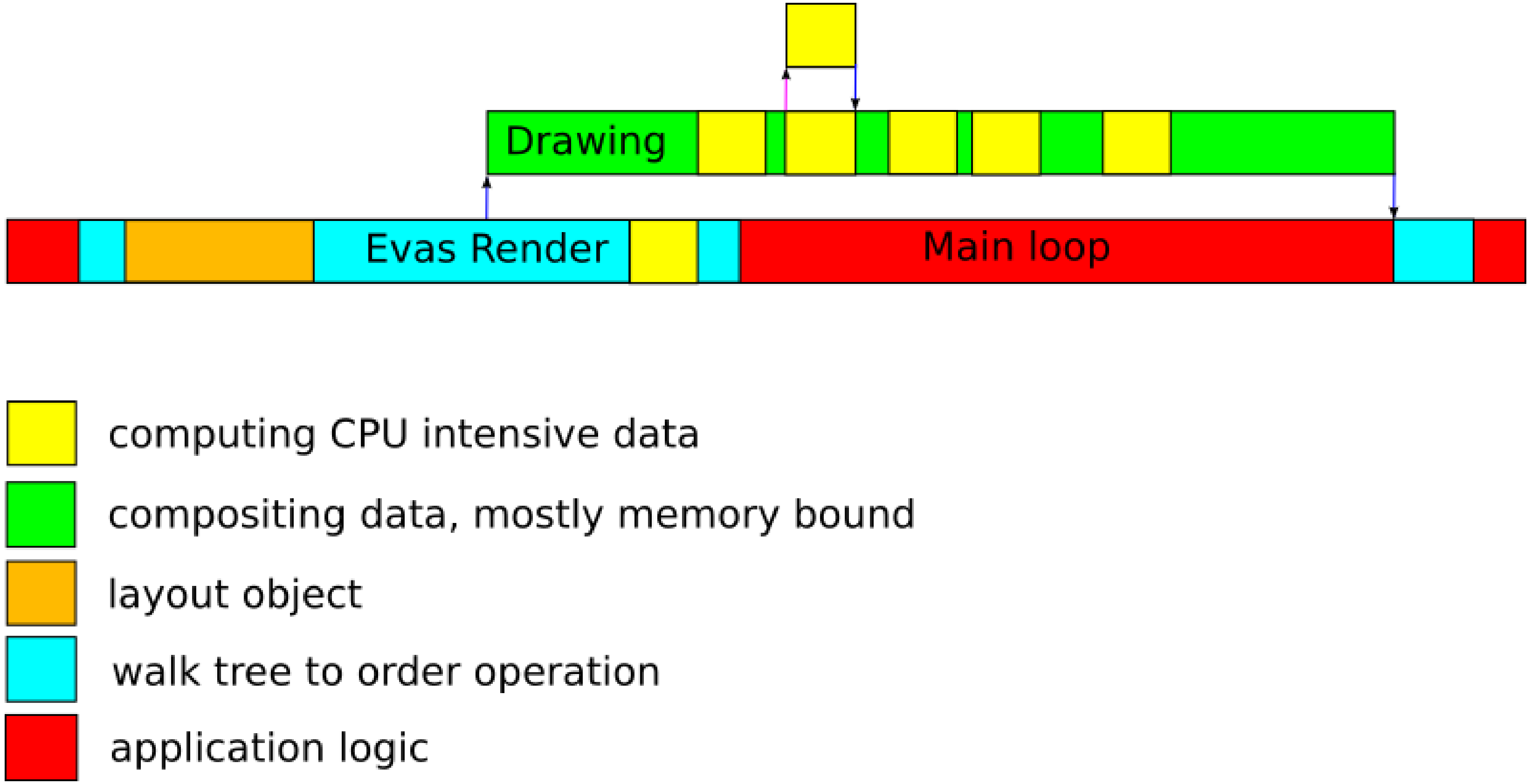
Rendering Pipeline – Where we started



- 🟨 computing CPU intensive data
- 🟩 compositing data, mostly memory bound
- 🟧 layout object
- 🟦 walk tree to order operation
- 🟥 application logic

Historical rendering pipeline for Evas

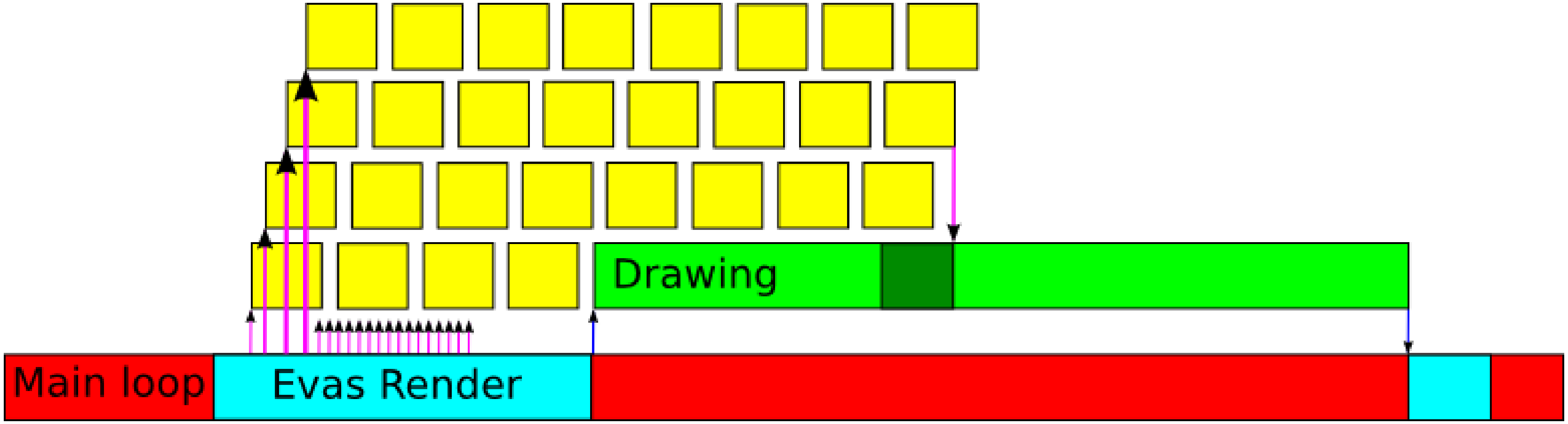
Rendering Pipeline – Where we are



- 🟨 computing CPU intensive data
- 🟩 compositing data, mostly memory bound
- 🟧 layout object
- 🟦 walk tree to order operation
- 🟥 application logic

Current rendering pipeline for Evas

Rendering Pipeline – Where we are going



Future rendering pipeline for Evas

Ector – Retained rendering library

- The idea of retained rendering come from Enesim (http://www.enesim.org)
- Didn't want to reinvent everything, so we have multiple backend :
  - Freetype
  - Cairo
- Ector is used by Evas for the drawing

Ector – Freetype backend

- Freetype provide an API to get spans lines easily
- Freetype provide in fact a source code you can include in your project
- It's fast, tested and support all the primitive we need to generate all shape
- Also we can make it match our retained API well
- We do already have better performance than we expected
- Still missing
  - Deduplicating shape
  - Asynchronous computation of shape and gradient information
  - GL backend

Ector – API

- Surface object per backend with a renderer factory
- Renderer just draw one primitive into the surface that created them
- Renderer can be moved at no cost (Ease reuse)
- Renderer have 3 functions :
  - Prepare
  - Render
  - Fill
- Renderer :
  - Shape
  - Gradient Linear
  - Gradient Radial

Ector – Where are we going ?

- Improve testing
- Understand why we have so much difference with Cairo
- Experiment with different GL backend design
  - Classic Loop & Blinn Approach
  - Use a texture filled with span information to fill
- Vulkan backend once we have some driver to play with
- Replacement of all Evas immediate rendering code by Ector
  - Filters
  - Text
  - Image
  - GL

Evas – Integration with the scene graph

- Evas work with Evas_Object primitive :
  - Rectangle
  - Image
  - Text
- We just added a Vector graphics object that is handle as a transparent object
- Contain a tree of primitives
- Tree's :
  - Can be disconnected from the canvas
  - Can be duplicated
  - Can be interpolated (following w3c SVG specification)

Evas – What come next ?

- Add SVG file loading/saving support
- Add EET (binary file format for theme) file loading/saving support
- Add more primitive (Likely order) :
  - Filter
  - Text
  - Image

Edje – Theme integration proposal

- SVG animation and interaction definition are “tricky”
- Most tool generate heavy animation instead of simpler one
- Keep it simple :
  - Starting point defined by SVG
  - End point defined by another SVG
  - Interpolate in between them
- Vector graphics part
  - State defined by a SVG
  - Program define rules for interpolation

EFL – Vector graphics cheat sheet

- Vector graphics will always be slower than just image rendering
- Can still be made fast and usable for real time user interface component
- Require to rethink how we do rendering
- Retained rendering is likely to open a lot of possibility in the futur
- EFL introduce 3 new components :
  - *Ector*: Retained rendering library
  - *Evas_Object_VG*: Vector graphics scene graph object
  - *VECTOR*: part in Edje theme

Questions?

We're Hiring!

Twitter: @SamsungOSG
Email: jobs@osg.samsung.com
Slides: http://www.slideshare.net/SamsungOSG