Rendercontext & Double-Buffering

By Jan Holesovsky

@JHolesovsky <kendy@collabora.com>

VCL changes...

- VCL (Visual Class Library)
  - LibreOffice's graphics toolkit
  - ~20 year history
  - Undergoing a major upgrade to allow modern features like OpenGL support
- Attend the Michael's VCL talk
  - The rendercontext is just part of the entire picture

When do we draw?

- Before the RenderContext rework started, Paint() methods were called just at any time
  - When painting (that's OK of course)
  - But also in event handlers (key press, mouse over effect, ...)
  - Triggered by timer
  - Any other random time (eg. in Writer – the debug rectangle at the top left when layout finishes)

Ideal state

- Painting triggered in a controlled way
  - Only the Paint() methods paint
  - Only VCL triggers the paint
    - Consequently it can control the conditions of the paint – various setups / tear downs etc.
  - Everything else only invalidates the area
    - And VCL decides when to paint, and what
- Painting de-coupled from vcl::Window
  - vcl::Window becomes more abstract

RenderContext – what's that?

- RenderContext: class that implements the drawing
  - At the moment, vcl::Window inherits from OutputDevice which allows all the painting at random points of time
    - That's what we want to avoid
- Instead, RenderContext is an implementation of the OutputDevice
  - And is passed as a param of the Paint() method
  - vcl::Window paints only in Paint()

Problems with direct paints

- Direct paints are problematic, because the render context is not available
  - The code that previously called Paint() directly now has to use Invalidate()
  - Invalidate()s are fast now – thanks to the Idle work
- Rework to use Invalidate() has to be done carefully though
  - Danger of Invalidate() loops

Double-buffering

- Easy once RenderContext is used everywhere
  - vcl/source/window/paint.cxx responsible for the rendering in the right order
  - For double-buffering, additionally:
    - Buffer set up before calling paint (VirtualDevice)
    - Then call the Paint()s (as before)
    - Copy the buffer to the screen when done

Rendercontext rework

- Easy parts
  - Adding the RenderContext parameter (via clang plugin)
- Hard parts
  - Everything else :-)
- Implemented by Tomaž Vajngerl and Miklos Vajna
  - Laszlo Nemeth and others nailed down many bugs – thank you!

Hard parts of the work

- Direct paints stateful in many cases
  - Background set once in a constructor, instead of the Paint method
- OutputDevice cached
  - Many places just try to remember the OutputDevice, and paint to it later
- Blinking cursor
  - Currently it just inverts what is on the screen
- Size of the window vs. size of the rendercontext confusion

Current status

- Currently
  - Most of the classes modified to paint only in the Paint() methods
  - StartCenter completely double-buffered
  - Writer mostly double-buffered
    - Except text cursor – needs inverting still – and some deep pieces
- Try yourself:
  - export VCL_DOUBLEBUFFERING_FORCE_ENABLE=1

DEMO

TODO

- Text cursor
  - Inverting not convenient; should we have it as a flat rectangle? [as in Firefox etc.]
- Switch it on for StartCenter and Writer
- Cleanup
  - Get rid of the code paths that are not needed for double-buffering
- Implement it for Calc, Impress and Base

And further...

- Switch all the drawing to tiled rendering
  - Paint methods would not paint the entire screen, but only 256x256 'tiles'
- Currently used on Android & LibreOffice On-Line
  - Adding Desktop would make it one code path again
  - Would allow extremely fast OpenGL scrolling / panning / zoom

Questions?

Thanks for listening!