A year in the Visual Class Libraries (VCL)

Michael Meeks
Pseudo Engineer, itinerant idiot
michael.meeks@collabora.com
mmeeks, #libreoffice-dev, irc.freenode.net

“Stand at the crossroads and look; ask for the ancient paths, ask where the good way is, and walk in it, and you will find rest for your souls...” - Jeremiah 6:16
Overview

- New & interesting stuff in VCL
  - Main-loop / Idle
  - Not much RenderContext (Kendy did it)
  - OpenGL
    - challenges & fun.
  - VclPtr
    - why, what, how, debugging ...
- TODO: Where next ?
  - RGBA vs. RGB + A
  - Cleanups …
New: Main-loop / idle
How it used to work ...

- VCL had (and still has) a single timer
  - This is a system-timer
  - Abstracted in SalTimer
  - We set the duration to the shortest timeout we have.
  - Wait for timeout, and emit any expired timeouts.
  - Start again ...
How it used to work #2

• So – to order events eg.
  • resize & layout window (50ms)
  • re-render window: Paint timer (30ms)
• A zoo of different length timeouts
  • 250+ timers of various sorts.
• Now fixed:
  • Timeouts (as before) & 'Idle' handlers.
  • Idle → “when nothing else is going on do it”
    - Prioritized to get sensible ordering ...
Now -much- better & cleaner.

- Can now get rapidly to the right idle handler
  - 'Invalidate' a widget → it will be re-rendered soon; not 30ms later …
- High resolution timers on Windows (Kendy)
  - Windows 'rounds-up' your timers to 10ms – unless another app on the system asked for high-resolution timers.
- Thanks to: Munich guys …
  - jennifer.liebel@muenchen.de, tobias.madl@muenchen.de, florian.haftmann@informatik.tu-muenchen.de
- https://wiki.documentfoundation.org/Development/LHM_LiMux/Main_Loop
Problem type

- Some code used to loop all the time
  - re-render something every 30ms – why not?
  - Writer / deferring spell/grammer checking for 'a bit' hoping for the next keystroke
    - Type “misspelledthin” (and don't press space) …
  - Beyond-lame 'Timer' to collect state and stash it into a struct – so it'll be ready on an event emission.
- Now we see this dumb-ness as a 100% CPU burning hot loop … <fix it>
  - For now: hacked a bit – low prio. idles throttled to 5ms: we should kill that on master …
Problem type:

- Starvation by high-prio Idle handler
  - This guy will always get the callback.
  - lower-prio. Idle handlers: no look in
- Effect:
  - If prio. higher than render: 100% CPU – and no re-rendering: but … otherwise usable app.
- Arguably better
  - previously priority / ordering not obvious – based on timeouts.
  - Races: Unix - configure_event vs. paint timer
    https://gerrit.libreoffice.org/#/c/16273/ - thanks Caolan.
VclPtr
VclPtr change ...

• Intended to be -minimal- not a complete fix, but getting the simple, basics in-place, avoid touching the tar-baby too hard etc.
  • ( its the way you tell them )
• Initial branch merge:
  • 276 commits thanks to: Noel Grandin & Myself.
  • 2635 files changed, 24679 insertions(+), 41344 deletions(-)
  • 'make check' passing (2x days before)
  • new unit tests written …
• But: VclPtr tracker bug (zero open)
  • As of today: 61 regression bugs tracked (~65 'vclptr' commits)
    – Great work from the QA team.
    – Left paranoid assertions on – no longer needed.
  • 5 bugs 'escaped'
    – 5.0.1 → 3 fixed.  5.0.2 → 2 fixed.
What was the change?

- Previously:
  - Window / OutputDevice classes had unusual lifecycles:
    - members, stack allocated, heap allocated.
  - Normal to have a heap allocated parent, with a ton of stack allocated children.
    - delete pParent → implicitly delete's memberChildren.

- Unfortunately:
  - Lifecycle reasonably impenetrable.
Impenetrable?

- Window's could have ref-counted UNO / toolkit peers:
  
  ```java
  WorkWindow aWin;
  Reference< awt::XWindowPeer > xWin = aWin.GetComponentInterface();
  ```

- Window's also got wrapped in boost
  
  ```cpp
  shared_ptr<WorkWindow> aWin( new WorkWindow() );
  ```

- Destruction semantics very unclear.
  
  - Impress a particularly good example.
  
  ```cpp
  shared_ptr + UNO / framework + destruction ordering \rightarrow extreme fragility.
  ```
Destruction ordering...

• Window extremely paranoid wrt. children being destroyed before parents.
  • We left this dbgutil assert / check in to help find dodgy places; no real need for it.
    - This was the majority of the regressions.
  • A child could easily be a member
    • We need to avoid FMR's at all costs => assert if child not destroyed.
[In]Correct coding ...

- Lots of code not robust vs. unexpected window destruction.
  - What you should (have) done:
    ImplDelData aDelData;
    ImplAddDel( &aDelData );
    ... call some user / virtual callback ...
    if ( aDelData.IsDead() )
      return;
    ... otherwise safe to use 'this' again ...
  - ie. 'this' would disappear under you.
    - Not-expected.
    - Dozens of code-paths unsafe in this regard.
    - Others use: Window::doLazyDelete ...
New world:

- Thanks for the update, refcounting Windows always seemed like a good idea - Philipp Lohman
  - The man with the previous VCL plan ...

- All children are VclPtr's
  - A VclPtr is either NULL, points to valid memory (perhaps a disposed object)
Reference cycles are implicit:

- Window has VclPtr's to parents, children, next windows (sometimes itself) etc.

No weak references. All strong.

So how do we free anything?
Disposing ...

'releasing all of this object's references'

Dispose Peter

Dispose Jane
New world:

- **Dispose:**
  - Frees backend resources – leaving a very small 'empty' placeholder.
  - Happens only once:
    - 'disposeOnce' makes this fly.
  - *Ideally* – methods called on a disposed object don't SEGV ...
    - We have belt & braces protected many dozens of methods.
  - So no need for dog-tag type code; we'll get some sane default behavior.
Problem type: destructors

- Moved all Window sub-class destructor code into ::dispose
  - ~Foo::Foo → ~Foo::dispose
  - ~Foo::Foo() { disposeOnce(); }
    - Clang plugin to check.

- Problem is:
  - There can be other members of Foo that are not Window sub-classes
    - They assume destruction ordering
      → need to add 'dispose' and call from ::dispose
Problem type: vtables ...

- As you destroy your C++ object:
  - The vtable is mapped to sub-classes
    ```cpp
class Base {
    virtual void doFoo() { print ("hello world"); }
    ~Base() { doFoo(); }
}

class Inherit : public Base {
    virtual void doFoo() { print ("whatever"); }
}
```
  - As we destroy `~Inherit` → “hello world”
  - BUT → if we move all destructors to:
    - virtual dispose() methods – this is not so.
    - Nasty re-ordering issues ... fixed a large number of these.
Other benefits

• Can implement UNO interfaces directly in a Window
  • No helpful templates yet, but – shouldn't need a separate proxy object
    → de-bloat accessibility peers.
• Ultimately more stable, reliable & defensive code.
  • Multiple disposeAndClear's are fine.
  • found & fixed a good number of leaks while porting.
• Everything is now my fault …
  • Previously it was Caolan's (wrt. layout work)
  • cf. VclPtr regression from ~LibreOffice 4.2 …
VclDemo
Had no good, small test-app

$ bin/run vcldemo

- Built as part of the OpenGL work …
OpenGL rendering ...
Lots of work here ...

- 2.1 man years in the last year from Collabora
- Where is the code?
  - `vcl/source/opengl`
    - OpenGLContext – management ...
  - `vcl/opengl/*`
    - Implements VCL / GDI backend etc.
  - `vcl/opengl/win/` or `unx/`
    - Windows / Unix specific platform / GL backends
- Why so long?
OpenGL's API is appalling ...

● Global (variable) state ...
   ● Is a complete nightmare.
   ● Encourages the most horrible programming
   ● Hides the most vicious bugs.

● Getting performance is 'fun'.

● Heavily using framebuffers to avoid GL context switching.

● Thanks so much to:
  ● Louis-Francis Ratté-Boulianne, Markus Mohrhard, Tomaž Vajngerl, Luboš Luňák, Jan Holesovsky, Tor Lillqvist, Marco Cecchetti, Miklos Vajna, Lubosz Sarnecki and others ...
Using modern OpenGL

• **Shaders**
  - Mini C-like programs → run on a very odd processor per-pixel.
  - Arbitrary Alpha blending algorithms eg.
  - Almost 'free' work …

• **Texture atlas (Tomaz)**
  - Managing a single large texture for our icons – handing out patches of pixels.
Using modern OpenGL

- Shaders – clever hacks for Anti-aliased lines:
  - Thanks to Chris Tsang:
  - Essentially rendering gradients along the edge of lines.
  - And Lubos Lunak for implementing this.
Reducing texture up-loads

- Font rendering ... (Tor)
  - Splitting UniScribe font rendering into:
    - Render (runs of) glyphs to bitmaps
    - Compose these into a single OpenGL texture.
  - Rendering text is:
    - foreach-glyph:
      - “alpha-blend <x,y,w,h> → there”
  - Improves performance nicely.
  - Creates space for Glyphy
    - → glyph rendering mostly on the GPU, on-demand.
GPU for CRC calculation

- To de-duplicate images internally
  - We use(d) a CRC32 → not ideal collision-wise
  - Switched to CRC64T (ish)

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Mpix</td>
<td>4.8ms</td>
<td>2.8ms</td>
</tr>
<tr>
<td>6Mpix</td>
<td>37ms</td>
<td>10ms</td>
</tr>
</tbody>
</table>

Leaves image data on the GPU until the last minute.

- 2x pass successive 16x reduction → CPU.
- CRC unit tests added
- Thanks to Marco Cecchetti

Consider a document with 100x images: 37ms → 3.7s
Gtk3 ... 
(Caolan's talk tomorrow)
VCL: the future ... at least my (un-funded) ideas ...
Cleanups

• Killing split alpha:
  • RGB + (1bit ? 8bit A) → RGBA everywhere
    – Avoid hitting all these odd driver paths.
    – Drastically simplify & improve performance of code.
  • Code-wise; a big change:
    – audit all BitmapEx usage
    – remove BitmapEx vs. Bitmap.
• Kill 1bit pixel formats …
• Kill under-window store / restore …
Wider Cleanups

• Push classes down into VCL
  • Lame VCL widget → less lame svx widget
  • no-one else uses VCL → we should avoid pointless inheritance for specialization

• Slideshow re-work
  • Lots of horrible hacks-around old VCL problems
  • Now we have high-resolution timers
    → bin the custom thread & main-loop there.
  • De-UNO-ise some of the fluff in here ...
More changes

• Continued Idle re-work
  • Unifying sources & timeouts/idle with priorities
  • Currently main-loop special-cases some input / event posting means.
  • Moving more things to low prio. Idle from timeouts. eg. sfx2 / toolbar updates.

• LibreOfficeKit / CloudSuite
  • Pre-initializing → glyph caching / pre-render etc.
  • Stubbing font / layout → unit testing ...
OpenGL work ...

- VirtualDevice – construction: always a 1x1
  - Reducing texture up-loads
- Glyphy → Markus has great plans ...
  - Beautiful SDF text rendering with transforms.
- Keeping geometry on the GPU
  - Currently re-tesselate & re-up-load constantly.
- Move 'image insensitization' to GPU ...
- Double buffering → get it on.
- Reduce duplicate working left & right ....
- glFlush optimization ...
Conclusions

• VCL:
  • A huge amount of work in the last year.
    - please be patient as it beds down ...
  • Major performance wins & long-term swamp draining
  • Fitting us for the deep cross-platform future.
• All executed alongside the community
  • Without you, it couldn't be done & it wouldn't be fun.
• Interested in helping out
  • Mail me / grab me on IRC ...

Oh, that my words were recorded, that they were written on a scroll, that they were inscribed with an iron tool on lead, or engraved in rock for ever! I know that my Redeemer lives, and that in the end he will stand upon the earth. And though this body has been destroyed yet in my flesh I will see God, I myself will see him, with my own eyes - I and not another. How my heart yearns within me. - Job 19: 23-27