Document number: P1200
Date: 06/10/2018
Reply to: Guy Davidson (guy@hatcat.com)
Reply to: (GB) Roger Orr (rogero@howzatt.demon.co.uk)
Reply to: (FI) Ville Voutilainen (ville.voutilainen@gmail.com)
Reply to: (ES) Jose Daniel Garcia Sanchez (josedaniel.garcia@uc3m.es)
Reply to: (FR) Jean-Paul Rigault (jpr@polytech.unice.fr)
Reply to: (RU) Antony Polukhin (antoshkka@gmail.com)
Reply to: (SI) Matevz Tadel (matevz.tadel@ijs.si)
Audience: WG21

High noon for the 2D Graphics proposal

Abstract

This proposal explains the rationale of the 2D graphics proposal and disputes some of the recent arguments against it, thus explaining why the push for the 2D graphics proposal is unwavering from several National Bodies. It proposes two ways forward for the committee to consider.

What is the 2D graphics proposal for, and what is it NOT for?

Let's start with the purpose of the 2D graphics proposal. Its goals are

- to provide straightforward graphics output
- to allow straightforward output of run-time data
- to do so portably
- to do so scalably

The significant non-goal of the proposal is maximal performance. Portability, scalability and out-of-the-box availability trump that (non-)goal.

Recently floated counter-arguments

Here is a collection of arguments against the 2D graphics proposal, with brief counter-arguments:

- The performance of an immediate 2D graphics API is bad, and it can't be implemented with good performance on modern graphics hardware. This means that serious applications and actual GUIs can't be built on top of it.
    - While an interesting claim, the existence of GNOME and various GTK applications using Cairo as their graphics engine, the foundation of the 2D graphics proposal, seems to dispute this immediately. Nor does the proposal necessarily imply either an immediate or retained API.
- The API is useless if it can't draw text.
    - We should add text support to this API, rather than stop working on it just because it doesn't draw text right now.
- We should go with an asset API instead.
    - The asset API draft shown to us can't draw lines or circles. That's very weak, much more so than an initial lack of text support. It's a different solution to a different

problem, and doesn't solve the problems that the 2D graphics proposal attempts to solve.

- We should go with a browser API instead.
    - Such a browser API is not a straightforward graphics API: it's an API to output strings, surrounded by HTML tags. That's not a drawing API. Again, it doesn't solve the problems that the 2D graphics proposal attempts to solve. Nor does it scale down to environments where running a browser engine is not feasible.
- We should go with a 2D graphics API that's designed from the ground up to work with OpenGL.
    - That seems like a promising idea, but only if you can make it work without OpenGL: OpenGL is not as portable as we want, nor does it scale down for the embedded developer community.
- We should go with something based on, for example, OpenGLES instead.
    - The proposal is API agnostic, and requires implementers to write a renderer using the most appropriate API for their environment. This could be a hardware-accelerated or software renderer. Also, OpenGL in all its forms is a vertex/texture/buffer management API rather than a drawing API. It is the incorrect level of abstraction.
- Graphics APIs change too frequently. WG21 can't keep up.
    - This is simply not the case. For example, Cairo has been quite stable for years, as have the user-facing APIs of Qt. We have existing APIs that have remained stable for half a decade or more and are surely candidates for standardisation of existing practice.
- This is too much work. It will take LWG a week to review the current proposal.
    - LWG knows how to arrange extra meetings if necessary, as evidenced by recent efforts.

Recently, those arguments, and variations thereof, have been made quite vocally. They all lack substance. There are better arguments:

- We should have a retained-mode API, closer to nv_path than Cairo.
    - Again, the proposal implies neither a retained nor immediate mode API. The final appearance of any drawing instructions will be the same to the user; how an implementer chooses to achieve this is for them to decide. Additionally, other rendering layers can be provided by hardware vendors to customise this very point, as specified by the proposal.
- We should solve this problem via package management.
    - That would indeed partially solve the availability problem; the user could just deploy CairoMM and use it, subject to the absence of regulatory impediment. Unfortunately, at the moment there is no schedule for delivery. Additionally, trying to solve this problem via package management certainly allows users to install whichever of the many different options are available, but it fails to provide an API that is ALWAYS available.

None of these arguments are convincing. They are based on a striving for maximal performance and the concern that the 2D graphics proposal is "too much" work; they also mitigate against something that's very much a bird-in-the-hand.

Available implementations

Linux/Cairo
Win32/Cairo
OS X/Cairo
OS X/CoreGraphics
iOS/CoreGraphics

In response to D1225R0

At the beginning of October 2018, JF Bastien posted a paper numbered D1225R0 to the lib-ext mailing list outlining Apple's response to the 2D Graphics Proposal. Following is a response to points raised therein.

The author lists several things Apple would do

- Support Multiple output devices. The API is output device independent, able to support multiple devices simultaneously. It is down to the implementer to provide a renderer, and the user to provide additional renderers they might require. It doesn't have to be a GPU driver or a software rasteriser; there is already a browser renderer (incomplete, through emscripten/WebGL) and rendering to PDF or SVG is similarly feasible. FP16 support is available through customising the maths at the specified customisation point. There is indeed alpha channel support.
- Anti-aliasing is implemented by the renderer. The API merely forwards requests for it.
- We plan to do text. Something that is missing from all the conversation is that the desire to publish a TS is motivated by the need to create a checkpoint, a work in progress, rather than a big bang of All The Things at some point in the future.
- Hardware support would be available through the renderer customisation point.
- Colour spaces and gamma support is not available. This would be ideal material for a further proposal against the TS.

Moving on to the teaching/plotting arguments

- The teaching argument is not about teaching graphics programming. It is about teaching C++ more immediately by having drawing output in addition to text output. However, it could be used as part of a graphics syllabus given additional material which is beyond the scope of this TS.
- Similarly with plotting: this is not the raison d'etre of the API, it is an application. Just as there is no single killer app for lambdas, neither is there a single killer app for drawing, it's just a useful tool to have available.

Regarding the higher capabilities

- The list is comprehensive and mostly observable in the reference implementation sample set:
- Obtaining a window object is achieved through the GraphicsSurfaces class template, and is a matter of implementation choice.
- Loading, transforming and drawing asset files is demonstrated in the sprites sample, as is stacking geometric transforms, and clipping/scissoring, and complex raster image support..
- User-implemented primitives have not been supported.
- Complex line drawing is shown in the Life sample.
- User input and text are unavailable at the moment.
- All of this should be available in shaders: Charley Bay is writing a Vulkan renderer at the moment which should demonstrate this.

The initial version list is significant

- Buffering is implementable by being aware of when the draw callback starts and finishes. It is up to the renderer implementation to decide what to do with it, whether to retain and draw or immediately pass to the frame buffer.
- DPI independence, modern colour spaces and gamma are all possibilities awaiting a concrete proposal.
- Nothing prevents you from addressing finer points than display points, again, this is up to the renderer: ultimately, everything has to be put onto the screen.
- Check the sprites sample for animation options.
- The proposal does specify in the GraphicsSurface specification support for PNG, JPEG and TIFF.
- Generating PDF, SVG and raster output are all renderer options.

Regarding the aesthetic feel of the library

- Dual error handling is clunky. We expect to rebase the TS as the standard progresses. There will be opportunities to exploit contracts and static exceptions which should remove error-prone functions.
- The APIs are indeed function orientated. We're open to suggestions for alternatives, but this was developed from Cairo, chosen as a rock solid, battle hardened, stable API in very wide use throughout the Linux world. The mission of the committee is to standardise existing practice as directed by the National Bodies, not innovate new practice.
- The path is the only container object in the API. It is a convenience type implemented with a vector. It would be trivial to add iterator/range support.
- Linear algebra is already the subject of additional standardisation effort. Paper D1166 (unpublished) will cover this. Geometry will become available conceptually once that hits the standard, expected for C++2x. Again, the TS can be rebased afterwards.
- Also, to be clear and to correct an error, both X11+Cairo and Cocoa+CoreGraphics are equally supported on OS X. There is an option to use the X11+Cairo renderer, but the CoreGraphics renderer is preferred. It is up to the user to choose in the CMake configuration.

So what should we do next?

The co-authors of this paper include some of those National Bodies whose official position is that they want to see a TS published based on P0267R8. We propose advancing the proposal the usual way, with LEWG design approval, then LWG review, and then a plenary poll for “direct the convenor to ask SC22 for a new work item for a TS with the contents of P0267Rn as its initial content”.