Document number: P0988
Date: 30/04/2018
Reply to: guy@hatcat.com
Audience: WG21

Ruminations on 2D graphics in the C++ International Standard

At Jacksonville in March the putative 2D graphics TS, P0267R7, was presented to a somewhat hostile audience and no consensus was reached on how to move it forward. This paper describes how the 2D graphics proposal got to this point and asks some questions about what to do next.

How did we get here?

September 2013, Chicago

A study group was convened to investigate the addition of a 2D graphics library to the C++ IS. This was SG13 Graphics, chaired by Herb Sutter. It was suggested to start with an existing library to get sight of a complete API and then to turn it into a modern C++ interface.

February 2014, Issaquah, SG13 Graphics

Michael McLaughlin presented the first version of a Cairo transformation. It was very well received and the author was encouraged to continue.

“Author direction welcomed: 8|2|1|0|0”

June 2014, Rapperswil, SG13 Graphics

By June, in Rapperswil, Michael had outlined a mechanical transformation which was again well received. There was considerable concern about rat-holing on the detail.

“This looks like a promising direction. We want to see wording for this shape of a proposal so we can move ahead in Urbana: 5|7|3|0|0”

November 2014, Urbana, SG13 Graphics

The transliteration was complete, and the quality of the wording was highly praised. There were no polls, but there was a strong “keep going” sentiment, and a fear of tiny changes filibustering the project.

February 2016, Jacksonville, LEWG

The original author spent a lot of time polishing the proposal. Fifteen months later it was presented to LEWG, where it was decided that the refresh rate should be user defined rather than “as fast as possible”.

June 2016, Oulu, LEWG

Surface and brush refinement was proposed, fonts were withdrawn to be added as a proposal against a working draft, and a decision was made to offer a dual error interface in the style of std::filesystem

February 2017, Kona, LEWG

[It was a few months before this meeting that this author joined the project.] There was further discussion about error handling, color stops and factory objects. The big advance was that the API was now stateless.

“Are we comfortable moving the paper to LWG for a graphics TS once these concerns are addressed and we’re happy with the wording? 1|8|3|1|1”

July 2017, Toronto, LEWG

Sadly, the notes are rather incomplete for this session. This author’s recollection is that the general gist was “still not quite ready”. However, the one recorded poll was “Do we want graphics support in a TS? 9|3|0|1|0”

March 2018, Jacksonville, LEWG

The implementation was completely abstracted away such that Cairo was a mere detail, as was the reliance on the float type for linear algebra. The authors believed the API and the accompanying wording was good to go forward to LWG, although it lacked text rendering and input.

However, some in the group were concerned that the paper was no longer appropriate for inclusion in the IS. The paper is large, at over 140 pages. The opportunity cost this represents to LWG is significant: it would comfortably swallow two committee meetings to verify the wording. This is at a time when both LWG and LEWG are processing fewer papers than they receive at each meeting.

There were also those who believed that there is no place for a 2D graphics API in the IS, along with those who believed that the API as it stands is not the right API, being stuck in the 1990s.

A poll was taken: “Should the paper be advanced to LWG? 1|3|3|3|2”

The effort was well and truly mired, without consensus on how to proceed, and no strong consensus to change the *status quo*. An evening session was scheduled for the next committee meeting in June 2018 at Rapperswil.

Do we still want 2D graphics support in the IS?

It depends who is in the room when you ask that question. The SG13 constituency (which is now called HMI rather than Graphics) are keen to see support but they are obviously a self-selecting group of people. During the Summer of 2017, LEWG were strongly in favour of 2D graphics support, but by Spring 2018 LEWG's support had weakened considerably.

This represents something of a problem. In excess of a thousand hours of volunteer time has gone into the graphics proposal: it is a considerable undertaking that does not fit well with attitudes blowing hot and cold.

This author spent much of 2017 visiting conferences and canvassing opinion. There was extensive interest in 2D graphics support, and paper P0669 consists mainly of the result of that investigation.

This author is not the only contributor. Besides Michael McLaughlin, the original author, two others have started implementing sample programs as well as rendering backends for other platforms using non-Cairo solutions. There are now versions for OS X, Windows and Linux; tests are being developed for other implementers and moves are afoot to complete the wording.

The wording is significant, running to over 140 pages as stated earlier. It does seem slightly unexpected that something for rendering paths, images and text should generate quite so much verbiage. There are six sections to the paper:

1. Color
2. Linear Algebra
3. Geometry
4. Paths
5. Brushes
6. Surfaces

As mentioned, text and input are yet to be specified but that would form two more significant sections. However, paths and surfaces alone take up 90 pages. There is a lot of text to describe how paths are rendered, how anti-aliasing works, how overlapping works and so on. These features have to follow well-defined mathematical rules.

Could this huge proposal be broken up?

The proposal draws together a number of large concepts which, independently, make significant contributions to the language without being specific to 2D graphics.

Canvas management

To render 2D graphics (or 3D graphics for that matter) you need a render surface, a canvas, hosted by the Operating System. There is a large variety of canvas hosting systems in use currently, per OS and per UI/graphics library e.g. Qt, SDL, SFML, so the job of picking one that every system can fit into is not trivial. However, such a system does not need to expose all the paraphernalia of a GUI (nor should it), merely somewhere to render to.

Colour

Who hasn't wanted to colour their output? Simply typing `ls` at the Bash prompt can yield a variety of hues on your terminal, each colour representing a different class of file. By defining colour in the standard you can improve your user feedback at the terminal.

Linear algebra

Maths is under-served by the standard. A linear algebra library is long overdue, and can be implemented in a variety of ways to get the best out of hardware. Using SIMD instructions for vector and matrix multiplication will shortly be feasible in standard C++ with the inclusion of P0214 in the IS.

Geometry

Standard formal definitions of geometric primitives such as line-segments, triangles, rectangles, ellipses, squares and circles (and possibly their 3D equivalents) become feasible with standard geometry objects. This is useful for a broad range of applications: not just drawing applications but mathematical and scientific applications.

Input

There is no support for hardware controller input. std::cin is the mechanism by which you retrieve keyboard input, translated by the host for you into individual keypresses. There are a variety of common input devices available now in addition to the keyboard: button and position input can be retrieved from a mouse, game controllers and trackballs. An input proposal would need to enumerate available controllers, map identifiers to each button or key and respond to queries for press states and position states.

Text

SG16 was convened at Jacksonville to deal with a variety of Unicode text processing issues. Putting text on a surface requires font enumeration and glyph rendering. While this is useful for the teletype, it's a short hop and a skip to rendering on a canvas.

However, Bjarne Stroustrup often repeats the advice "don't design systems in isolation". Delivering all these systems in a single paper amplifies cohesion by identifying their edges and offering immediate use cases. This can't be overstated: this paper delivers so much more than simply a graphics API: it also advances the standard in several other directions.

The first four of these areas, delivered as individual papers, may take at least as much time as the single paper, and possibly more. The paper is a coherent whole which can be reviewed as a single piece by LWG. Breaking it up would extend this effort as each paper would need to be self-contained and voted into the standard before the next paper could be reviewed. This could add several years to the effort. As it stands, if we want to introduce input and text into the IS at the same time as the rest of the paper we will have a tight race on our hands. This author believes that breaking up the paper may undermine the effort as a whole, or at least incur considerable additional time cost.

Additionally, the linear algebra and geometry components, although incomplete, do form a solid foundation for extension to fully equipped libraries. Trying to make that extension *in vacuo* can only be a formidable task: it would be better to extend them from a position of already being in use.

Is this the right 2D API?

Cairo is quite an old API. It has been around for over 15 years now, and it was chosen as the API model because of its stability and battle-hardened state. Several new APIs have emerged since, some at a higher level of abstraction. Cairo is used by GTK applications and the Gnome desktop; Chrome and Firefox also used Cairo before moving on to bespoke libraries. This makes it a very well used library.

The original author's process was to mechanically transform the Cairo API from C to C++, introducing RAII along the way. Additionally, the API is now stateless and the Cairo implementation is now a customisation point rather than a requirement. P0267 now describes a modern C++ interface for 2D graphics suitable for publication as a TS. Wording polish aside, it is ready to receive input from the outside world. Whether or not it is the right API is moot until another API is offered for comparison.

A grander objection to this API, and indeed to a 2D graphics library, is that C++ isn't FOR this sort of thing. There should be no room beneath C++ for something else: it should be optimal and unimprovable. There is a belief held by some in the committee that there should be NO such API because it is impossible to design without compromising somewhere on performance.

However, large chunks of the C++ standard library could be described as a performance compromise for the sake of a good API. This author is part of the SG14 constituency, and spends his days writing high-performance games for Windows machines. It is vital that not a cycle be wasted. As a result of this, there is a lot of bypass going on. Exceptions are disabled, so the standard containers are eschewed for hand-rolled, exception-unsafe versions. File IO is handled by exotic OS manipulation involving memory-mapping rather than using std::fstream. RTTI is disabled so std::function is discarded in favour of a hand-rolled non-RTTI version. The list is significant.

This author contends that 2D graphics fits in the standard library in the same space as std::fstream. It is ideal for cross platform solutions for throwing up telemetry feedback to developers, charts, graphs, dialogs, debug output, mobile phone apps and yes, even some games. If the library is insufficiently performant, the programmer can look elsewhere.

Additionally, the Direction paper P0939 lists simple graphics as a priority for C++ 20 if time allows, and a medium term priority in any case. The sample programs implemented using P0267 reference implementation include an SVG renderer, an implementation of Conway's cellular automaton Life and a clone of Atari's Asteroids game. There is a simple real-time profiler example in development and, together with the other samples, it seems fair to say that the requirement of a simple graphics library has been delivered *in excelsis*.

If this is not the right API, another is required, and soon. This API is a bird in the hand, and the committee is not in the habit of waiting for another proposal while one is ready to go; nor should it be, especially in this case. New graphics APIs turn up periodically: by the time a new proposal is polished, yet another graphics API could be ready to take its place as the "superior" choice. We have been content with character IO for decades: why are we suddenly keen for the latest new thing?

So what should the committee do now?

There are a number of ways forward.

1. Nuke it from orbit, it's the only way to be sure. Buy this author a beer and goto end.
2. Confirm that the committee still wants standard 2D graphics support somehow and either.
    a. Wait for package management to manifest and put the (by then) finished proposal in a committee approved package.
    b. Decide on an incremental roadmap and build a TS piecemeal.
    c. Take what exists, publish a working draft and invite appending and amending proposals against it.
    d. Solicit a counter-proposal with a different, more popular approach.
    e. Some combination of the above.

This author is keen to hear other options, but in his opinion he believes the publication of a working draft prior to a TS is the starting point of any continuation. A TS does not imply addition to the IS or any putative package manager, it favours velocity in development of the standard, and it gives the committee a stake in the ground from which to look forward and plan ahead.