Guidance for Reading P0267R9

Author: Michael B. McLaughlin
Target Audience: SG-13
Date: 2019-06-19

1. Overview/Goals of P0267

(Note: If you're already familiar with P0267, feel free to skip this section.)

The P0267 proposal specifies requirements for a standard interface that computer programs written in the C++ programming language may use to render and display 2D computer graphics. It includes the five fundamental operations in 2D graphics: painting, stroking, filling, masking, and text rendering.

P0267R9 is the 12th revision of the proposal, which is the result of several years of development, feedback, and revisions based upon that feedback. Once the rendering and display ("output") API has stabilized, an input API will follow, building upon this work.

When C++ was created, the dominant form of user interaction with computers was the terminal, also known as a console. Text was displayed to the user via a terminal and input was received from the user via the terminal in the form of text. Standard C++ fully supported this dominant form of user interaction and we believe that it was an important feature of the language.

Today, the display of 2D graphics to the user and input from the user in the form of keyboard, mouse, and touch based upon that graphical output are the dominant form of user interaction. Because Standard C++ does not presently support this, we assert that Standard C++ has lost an important feature. This proposal is intended to restore that feature by once again providing Standard C++ with an API to interact with users using the dominant form of user interaction.

For more information, read Clause 0 of P0267R9, available at: https://github.com/cpp-io2d/io2dts/blob/master/papers/P0267R9.pdf .

2. Important new features in P0267R9

To provide a better experience when approaching the proposal, Clause 0 has been changed from being just a Revision History to being an Introduction clause, containing a Background and Rational section as well as the Revision History. The intent of this section going forward is to provide someone who is approaching the proposal for the first time with sufficient information to gain familiarity with the intentions of the proposal and with its contents without the need to already have an advanced level of knowledge of the subject matter. This is by no means complete and its development will be an ongoing effort.

a. Text Rendering

The text rendering API contains an initial feature set. It is not complete. That said, I did not feel that the absence of the missing features in R9 would detract from the opportunity to get feedback on the design of the text API that is included in R9.

The text API's feature set was developed using an intersection of the features provided by Qt and Pango. The decision to focus on the features provided by Qt and Pango was made because they are libraries that are both broadly cross-platform. HTML/CSS, SVG, Core Text, and other libraries/specifications were also examined.

The specification of the text API relies heavily on ISO/IEC 14496-22 - Open Font Format (OFF) in addition to ISO/IEC 10646 - Universal Character Coded Set (needed for UTF-8 support). Both are Publicly Available Standards. You can access and download them for free from the ISO website here: https://standards.iso.org/ittf/PubliclyAvailableStandards/ . (OFF standardizes TTF, CFF, SVG, and bitmap glyphs within a single, unified format and is widely used on all major platforms.) Non-OFF fonts are not supported by the text API. To the extent that users need such support they can use the other 2D graphics APIs (this would primarily be needed for non-OFF bitmap fonts, which can easily be rendered using Fill or Mask operations).

We proceed now to an overview of the missing features.

i. Measurement of Text (Metrics)

Metrics are difficult in that users will generally want to know how much space text will take up when actually rendered but factors such as kerning and merging (using a similar font to render text that the chosen font does not provide glyphs for) make it difficult to determine this without actually rendering the text.

This is not an intractable problem, but there was not enough time to determine the capabilities that can be offered by this feature without significant performance implications. The fundamental text rendering API was subject to various revisions as I developed it and I simply ran out of time to develop this for R9.

ii. A Layout Type

Layouts (sometimes called Runs) are an important feature for complex text rendering. A layout type will provide users the ability to create an object that contains text to be rendered, the properties and other state data that should be used to render it, and information regarding its relative position and size. It is not, in my opinion, particularly difficult to address, but without a proper metrics API, key features would have been missing. Designing it without having some of its key features seemed inadvisable so its inclusion was deferred.

iii. Modification of spacing

In order to properly render HTML and SVG, the ability to modify the spacing between characters and between lines is necessary. I felt that determining how best to provide this feature would require implementing it in several environments using different back ends. There was insufficient time to do this. Qt does it on all of the major platforms, but I did not want to just take the Qt API and trust that it would be the best way to do it.

iv. BiDi and Vertical Alignment

The ability to properly render and display BiDi (scripts that are written right-to-left "RTL" as well as those written left-to-write "LTR") text is quite important. Arabic, Hebrew, and a number of other scripts are RTL. Determining how to properly place RTL text based on coordinates given and how to properly handle text that intermixes RTL and LTR scripts requires more time than I was able to devote to preparing R9.

The same is true for text that is rendered in a vertical alignment rather than a horizontal alignment. By way of example, both Japanese and Chinese can be written in a vertical alignment (in which case text is written from top to bottom, with each line being to the left of the previous line) in addition to horizontal alignment.

b. Command Lists

Command lists provide the ability to create a set of commands to perform the five rendering and composing operations as well as the ability to execute user-provided functions. They provide efficiency and reusability and, in the case of basic_image_surface objects, can be run on a separate thread. The efficiency gains are primarily expected to be used by back ends that use graphics hardware acceleration, such as a D3D12 or Vulkan back end. The reusability is helpful regardless of the back end. Additional information about the expected use cases for command lists and whether and how to choose between command lists versus the "direct" API (calling a surface's rendering and composing functions directly) can be found in the note on page 190 of P0267R9 (in the specification of the basic_image_surface::command_list member function (section 16.4.8)).

c. Miscellany

There were several other changes made between R8 and R9. These are documented in the Revision History and are minor changes.

The specification of the requirements for a GraphicsSurfaces back end, creation of which was a result of the abstraction of the implementation made in R7, continues. It will be completed in R10 as will full descriptions of basic_unmanaged_output_surface. Defining all front end classes in a uniform manner is also ongoing and is expected to be completed in R10.

3. Summary

The main additions to R9 are text rendering and command lists. Text rendering provides an initial feature set that will be added to and modified in subsequent revisions both as noted in this document and as a result of feedback gathered in Cologne and elsewhere. Command lists are also new, but are feature-complete. Any modifications to their design will come from feedback in Cologne, etc.