Kicad View component

Draft Specification

CERN BE-CO-HT / Tomasz Włostowski

Table of Contents

1 Introduction

The goal of the View component is to decouple drawing from editing tools and start using the Graphics Abstration Layer as the main rendering backend.

1.1 Rationale

There are several reasons:

- Currently the burden of managing redraws in Kicad falls on editing tools. For example, while routing a track, the tool code must take care of erasing the old trace, painting over the new one and making sure that there will be no rubbish on the screen when the view is panned or zoomed.
- More complex tools (e.g. a Push and Shove router or a live DRC checker) modify the apprarance of the neighbourhood of the item being edited. We need a way of efficiently redrawing all affected objects without reinventing the wheel every time in each new tool.
- There is no display list concept in Kicad. Citing Dick: *When showing a subset of the full display list (such as less than a full set of layers), tests are done in the drawing routines rather than up front one time. This leads to tests *in the drawing routines* that test for layer and other characteristics such as net.*
- Drawing using XORs limits the possibility of defining custom color configurations and does not respect layer order. For example, the soldermask/copper/paste layers should be drawn over each other, but with XORing they just look the same regardless of the drawing order.
- Printing requires extra code (e.g. changing drawing modes from xor to copy). Same for PDF/SVG export.
- Make rendering independent from the UI toolkit. This way, Kicad DLLs/DSOs could be used as a rendering engine for boards/footprints/schematics/symbols in 3rd party applications without the UI burden (e.g. a Web component library).
- Current drawing code does not behave consistently across different platforms (e.g. OS X not supporting XORing).

The view component implements a simple, yet efficient graphical item view, similar to `QGraphicsView` in the Qt library. Since wxWidgets does not provide a similar component with required functionality, we need to write our own.

2 Architecture

The place of the View component in Kicad's graphics architecture is shown in Figure 2.1. The basic classes are the `VIEW` class, representing our View and the `VIEW_ITEM` interface, which is implemented by every type of item that can be added to a View (in case of pcbnew, all `BOARD_ITEM`s and some additional objects such as selections or editing harnesses). Drawing is done via a `PAINTER` class, which draws a given `VIEW_ITEM` on a given layer using the `GAL` primitive drawing routines. Handling user events controlling the view behaviour (such as zooming, panning, mouse grab) is controlled by a separate class `VIEW_CONTROLS`. This way, the view has no dependency on the UI toolkit at all.

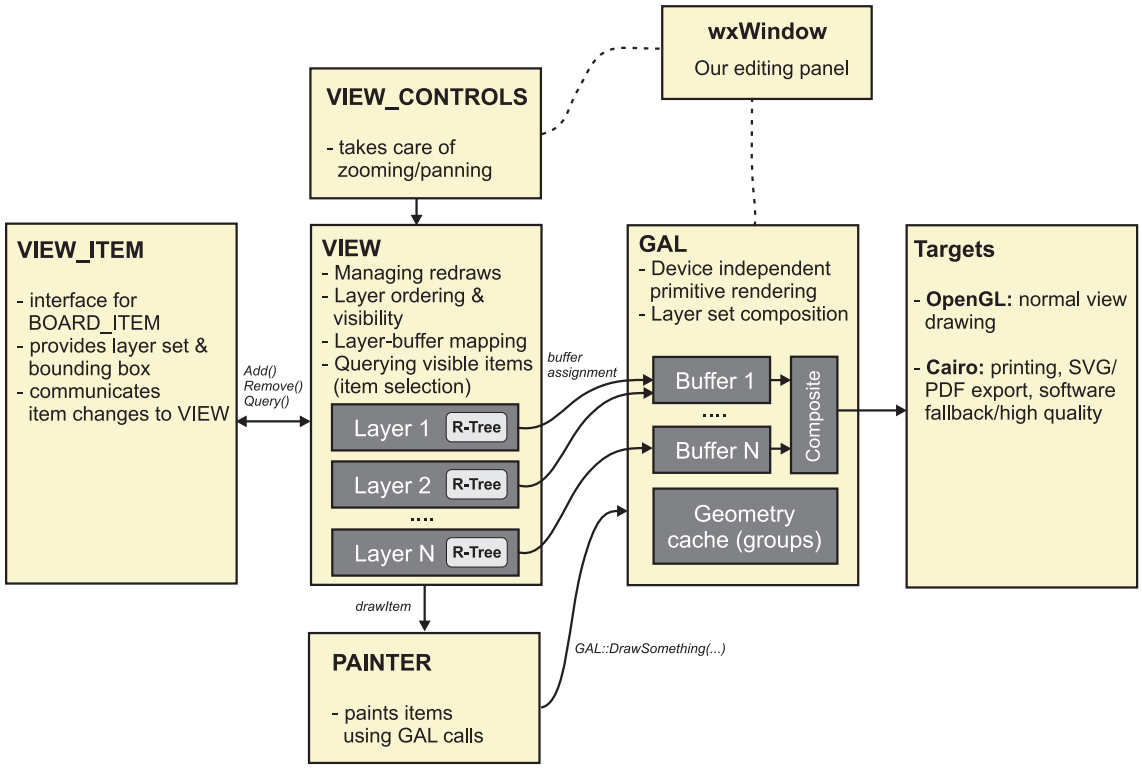

Figure 2.1: Architecture of View-based display engine.

Views exist in two flavors:

- **Dynamic** views, where each item holds a reference to the view it belongs to, therefore redrawing it is as simple as calling the `ViewUpdate` method. Items can be added/removed/modified at any time. The current proposal assumes that an item can be added to a single dynamic View. A dynamic view will be used for the main editing panel of pcbnew.
- **Static** views, which don't reference their items and don't allow changes once the view is built. An item can be added to any number of static views. Static views can be used for printing or image/PDF export.

Views are not containers, they don't own the items. When an item is deleted, it must be removed from its associated View.

3 Functional requirements

The `VIEW` class shall provide the following functionality:

- The obvious: adding & removing items,
- Viewport setting (viewport rectangle, center point, scale and mirror),
- Translation between world and screen coordinate systems,
- Adding layers,
- Setting layer rendering order and visibility,
- Assigning layer rendering buffer,
- Querying items that touch a given rectangle and are visible, according to their rendering order,
- Copying items from another view (static views only).

The `VIEW_ITEM` interface must provide:

- Bounding box of the item on a given layer,
- Set of layers the item can be displayed on,
- Item visible flag,
- (optionally) a drawing function, if drawing is not handled via the `PAINTER`.

4 Example implementation

The proposal for `VIEW` implementation lies in `view` subdirectory of the repository, with a demo in `tests/view_demo` subdirectory. The example renders a PCB and allows for selecting objects with the mouse. Test datasets are located in `data` subdirectory. Currently it compiles only under Linux, I'll port it to Windows as soon as I get Cairo to build on my Windows box.

Note that only class definitions are commented, the rest is not (sorry, I'm still working on it...).