xLights Edit Display Elements

Filter Performance Analysis

Author: Neil Heuer · Related PR: xLightsSequencer/xLights#6241

Overview

PR #6241 adds a real-time filter to the *Available* list in the Edit Display Elements dialog. Each keystroke triggers a rebuild of both list controls (Available + Added) so the filter is honoured. This document quantifies the per-keystroke cost, describes how to measure it on a real show, and lays out cheap mitigations to keep the experience smooth on slower hardware.

What each keystroke does

`OnNonModelsFilterText` → `PopulateModels()`. The path is:

```
DeleteAllItems on ListCtrlModels and ListCtrlNonModels
  -> loop sequenceElements (N items, ~50-500)
       for each: AddModelToList
                 (InsertItem + SetItem on ListCtrlModels)
  -> loop _xlFrame->AllModels (M items, ~50-2000)
       for each: ElementExists check
                 IsFilteredOutOfNonModels (lower + Find)
                 if not filtered:
                   InsertItem + SetItemPtrData + SetItem
                   on ListCtrlNonModels
  -> SetColumnWidth(wxLIST_AUTOSIZE) on column 0
  -> EnsureVisible (after the index clamp we landed)
```

Per keystroke the cost scales as **O(N + M)**, where M dominates on big shows. The expensive primitives are `wxListCtrl::InsertItem` / `SetItem` calls — each one nudges layout and triggers a repaint. On the macOS generic listctrl in particular, the per-row cost is noticeably higher than on the Windows native control.

Order-of-magnitude estimates

Gut estimates from generic-wxListCtrl behaviour, on a 5-year-old laptop. Real numbers depend on display scaling (HiDPI paints are slower) and CPU clock.

Show size (models)	ms per keystroke	Felt as
100	~5-10 ms	Instant
500	~30-60 ms	Slight lag, fine while typing
1000	~80-150 ms	Visible lag, queues keystrokes
2000+	200 ms+	Janky

Thresholds: <50 ms is fine, 50-100 ms is borderline, >100 ms warrants mitigation.

How to measure

Drop a timer around the handler. Cheap, accurate enough for triage:

```
void ViewsModelsPanel::OnNonModelsFilterText(wxCommandEvent& event)
{
    auto t0 = std::chrono::steady_clock::now();
    _nonModelFilter = TextCtrl_NonModelsFilter->
        GetValue().Lower().ToStdString();
    PopulateModels();
    auto ms = std::chrono::duration_cast<std::chrono::milliseconds>(
        std::chrono::steady_clock::now() - t0).count();
    spdlog::info("filter rebuild: {} ms (avail={}, view={})",
                 ms, ListCtrlNonModels->GetItemCount(),
                 ListCtrlModels->GetItemCount());
}
```

Then open a real big show, watch the log while typing. Optionally guard the timer behind a `#ifdef DEBUG_FILTER_TIMING` so it does not ship enabled but stays available for next time.

Mitigations ranked by effort / payoff

#	Mitigation	Effort	Payoff	Risk
1	Freeze() / Thaw() around the rebuild on both list controls. Tells wx to batch paints.	1 line	30-50% reduction on macOS generic listctrl	None
2	Debounce: start a wxTimer on each keystroke, restart on the next; fire PopulateModels only after ~150 ms pause.	~15 lines	Smooth, fast typing by skipping intermediate rebuilds	Low
3	Filter in place. Keep a master vector<Element*>, only insert/delete rows whose filter state changed (delta only).	~50 lines	Cuts work to O(delta) instead of O(N)	Medium - touches drag-drop, sele
4	Switch ListCtrlNonModels to wxLC_VIRTUAL. The list pulls only visible rows from a model.	Refactor	Near-instant at any size	High - touches drag-drop, selectio

Recommendation

For PR #6241 as it stands, do **#1 (Freeze/Thaw) + #2 (debounce)**. That is the 80/20: both are small and low-risk, and they cover the realistic worst case — a 1000-model show on a 2018-era Intel MacBook. If after that we get user reports of lag at 2000+ models, revisit #3 (in-place delta) before reaching for #4 (virtual listctrl).

Also worth committing the timer behind a `#ifdef DEBUG_FILTER_TIMING` so future perf questions take minutes, not hours.