УДК 004.5:004.92

ЕЛЕМЕНТИ АРХІТЕКТУРИ ШЕЙДЕРНО-ОРІЄНТОВАНОЇ БІБЛІОТЕКИ КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ

Музикін А. М., Бовчалюк С. Я.
email: artem.muzykin@nure.ua, stanislav.bovchaliuk@nure.ua
Харківський національний університет радіоелектроніки, каф. ЕОМ,
м. Харків, Україна

An architecture has been developed for a shader-oriented, cross-platform user interface library that integrates the CPU-GPU pipeline, the rendering system, and the widget architecture. The generation of the visual output has been moved to the shader pipeline, reducing the load on the CPU and the volume of data exchange. The Render System includes a frontend, a command queue with sorting and batching, resource management, a shader subsystem, backend abstraction, and GPU-level execution. The library provides complex effects, adaptive scaling, batch processing of UI elements, and stable frame rates, demonstrating the benefits of a GPU-first approach.

Дослідження архітектури бібліотек користувацького інтерфейсу показують переважання CPU-орієнтованих підходів, де логіка інтерфейсу та обробка графіки здебільшого виконуються центральним процесором із подальшою передачею результатів до графічного конвеєра. Хоча така модель типова для програм загального призначення, вона обмежена у системах із високими вимогами до продуктивності, масштабованості та енергоефективності, особливо у вбудованих рішеннях. Розвиток програмованих GPU та шейдерних технологій дозволяє перенести формування інтерфейсу на графічний процесор, забезпечуючи складні ефекти, адаптивне масштабування та пакетну обробку без значного навантаження на CPU.

Прикладами retained-mode систем, де GPU виступає основним виконавчим ресурсом при збереженні деревоподібної структури інтерфейсу, є scene graph у Qt Quick [1] та tree-based pipeline у Flutter [2], із чітким розділенням етапів побудови та рендерингу. Як приклад GPU-first підходу з явним представленням сцени у вигляді списку команд (display list) та перенесенням композиції на спеціалізований GPU-конвеєр доцільно розглядати архітектуру Mozilla WebRender [3], що демонструє відхід від CPU-растеризації на користь командної моделі рендерингу.

Метою роботи є створення і подальша практична реалізація архітектурної моделі шейдерно-орієнтованої бібліотеки користувацького інтерфейсу із забезпеченням кросплатформеності, що базується на структурному поділі системи на платформонезалежне ядро, рівень абстракції операційного середовища та модуль рендерингу, орієнтований на використання шейдерів.

Альтернативою підходам, орієнтованим на центральний процесор (CPU-first), виступає шейдерно-орієнтована (GPU-first) модель, у межах якої формування візуального вигляду віджетів і композиція сцени здійснюються засобами програмованого графічного конвеєра, тоді як на CPU покладаються лише функції підготовки команд рендерингу та обмеженого обсягу даних. Водночас широке впровадження GPU-first підходу ускладнюється відсутністю чітко формалізованої архітектури бібліотеки, що одночасно визначала б структуру системи рендерингу (черги, пакетування, управління ресурсами, шейдерну підсистему) та забезпечувала кросплатформеність завдяки абстрагуванню від віконного середовища, графічного контексту й механізмів введення без необхідності змін у ядрі системи.

На підставі аналізу архітектур retained-mode інтерфейсів із використанням графа сцени в Qt Quick та багаторівневої організації Flutter, що передбачає розділення етапів формування інтерфейсу та його рендерингу (widget/rendering/engine), визначено основні вимоги до побудови бібліотеки:

- формування стабільної ієрархічної структури UI із чітко визначеними етапами компонування та створення рендер-представлення;
- наявність рендерингової підсистеми, що трансформує UI-опис у набір команд із підтримкою сортування та пакетної обробки;
- реалізація механізмів керування ресурсами (текстури, атласи, буфери) разом із шейдерною підсистемою, що включає кешування.

З урахуванням концепції GPU-first, додатково формулюється вимога зменшення залежності від CPU-растеризації, скорочення обміну даними між CPU і GPU та досягнення передбачуваної продуктивності шляхом перенесення композиції й ефектів у програмований шейдерний конвеєр.

Рендерингова система UI-бібліотеки складається з взаємопов’язаних підсистем, що перетворюють високорівневий опис інтерфейсу у послідовність GPU-операцій із контрольованим часом кадру: фронтенд, черга команд (render queue), керування ресурсами, шейдерна підсистема, бекенд-абстракція, GPU-рівень виконання. Головна особливість шейдерно-орієнтованого підходу полягає в тому, що базові елементи інтерфейсу задаються як параметризовані примітиви, а не попередньо растеризовані поверхні. Їх обчислення виконується безпосередньо у вершинних та фрагментних шейдерах і охоплює геометричні форми (прямокутники, заокруглені кути, обводки), процедурні заповнення (градієнти), операції композиції (маскування, кліпінг), постобробку (розмиття) та анімаційні деформації, що не потребують CPU-растеризації при зміні стану.

У системах інтерфейсу текст потребує високої масштабованості та якості згладжування. Ефективним GPU-орієнтованим рішенням є використання distance field представлення гліфів, що дозволяє масштабування та часткові ефекти у шейдері з мінімальним обсягом даних.

Бібліотека віджетів реалізована як retained-mode модель, де базовий віджет визначає єдиний контракт життєвого циклу: зберігає стан, інтегрується в дерево (parent/children), керує властивостями з повідомленням про зміни, інвалідацією (mark dirty) та забезпечує відновлення узгодженості через проходи layout і підготовки рендерингу.

Практична реалізація інтеграції вимагає визначення правил перетворення високорівневих властивостей віджетів у GPU-параметри: кожен віджет має набір “рендер-атрибутів” (геометрія, трансформація, кліп, стиль, стан), які після нормалізації упаковуються у стандартизовані структури для шейдерів. Щоб мінімізувати зайві перерахунки, інвалідацію доцільно розділяти на layout-dirty і render-dirty, що дозволяє перераховувати лише геометрію або лише візуальні параметри за потреби. Адаптер “віджет → draw commands” забезпечує узгоджений порядок накладання, вкладені кліпи та підтримку окремих рендер-проходів (ефекти, текст) без порушення інкапсуляції віджетів і залежності від конкретного графічного API.

Висновки. Розроблено архітектуру шейдерно-орієнтованої кросплатформеної бібліотеки UI, що узгоджує три ключові компоненти: CPU-GPU конвеєр формування кадру, систему рендерингу та архітектуру віджетів. Модель усуває обмеження CPU-орієнтованих підходів – надлишкову CPU-растеризацію та високий трафік CPU-GPU, за рахунок перенесення формування візуального результату на шейдерний конвеєр і зведення ролі CPU до підготовки впорядкованої черги команд та мінімальних параметрів рендерингу. Сформовано узагальнену Render Systemь модель, що включає фронтенд для створення рендер-представлення, чергу команд із сортуванням і пакетуванням, керування ресурсами, шейдерну підсистему, бекенд-абстракцію графічного API та GPU-рівень виконання з підтримкою багатопрохідного рендерингу. Така структура дозволяє зменшити кількість draw calls, мінімізувати перемикання стану та стабілізувати час кадру завдяки батчингу та уніфікації “матеріалів” віджетів.

Список використаних джерел:

1. The Qt Company. (n.d.). Qt Quick Scene Graph. Qt Documentation. URL: https://doc.qt.io/qt-6/qtquick-visualcanvas-scenegraph.html (дата звернення: 20.02.2026).

2. Flutter Team. (2025, Dec 8). Flutter architectural overview. Flutter documentation. https://docs.flutter.dev/resources/architectural-overview (дата звернення: 20.02.2026).

3. Nical (Mozilla Gfx Team). (2018, Nov 29). WebRender newsletter #32. Mozilla Gfx Team Blog. URL: https://mozillagfx.wordpress.com/2018/11/29/webrender-newsletter-32/ (дата звернення: 20.02.2026).