slughorn: What's Next?

Roadmap & Sponsorship

Contents

1 What's Next?

Even though slughorn is still in its infancy — we started the first experiments and proof-of-concepts in April, 2026 — we're fairly certain we're developing something unique: not only building on Eric Lengyel's recently open-sourced Slug algorithm, but applying it in other interesting (and potentially even *novel*) ways. We know there's potential here, and community support and interest will help push it even further.

2 Obvious Improvements

There are some improvements that we **know** are not only possible, but would benefit slughorn immediately (given the time and resources to dedicate towards them). The following simply need to be implemented, as they are all "solved problems" in one form or another.

2.1 HarfBuzz & Text Layout

Building a `slughorn/harfbuzz.hpp` backend would unlock a number of really compelling features: emoji sequences, bidirectional text, ligatures, kerning, and more. While slughorn doesn't **currently** venture into the domain of "text layout," adding HarfBuzz shaping support would make such an endeavor much more complete.

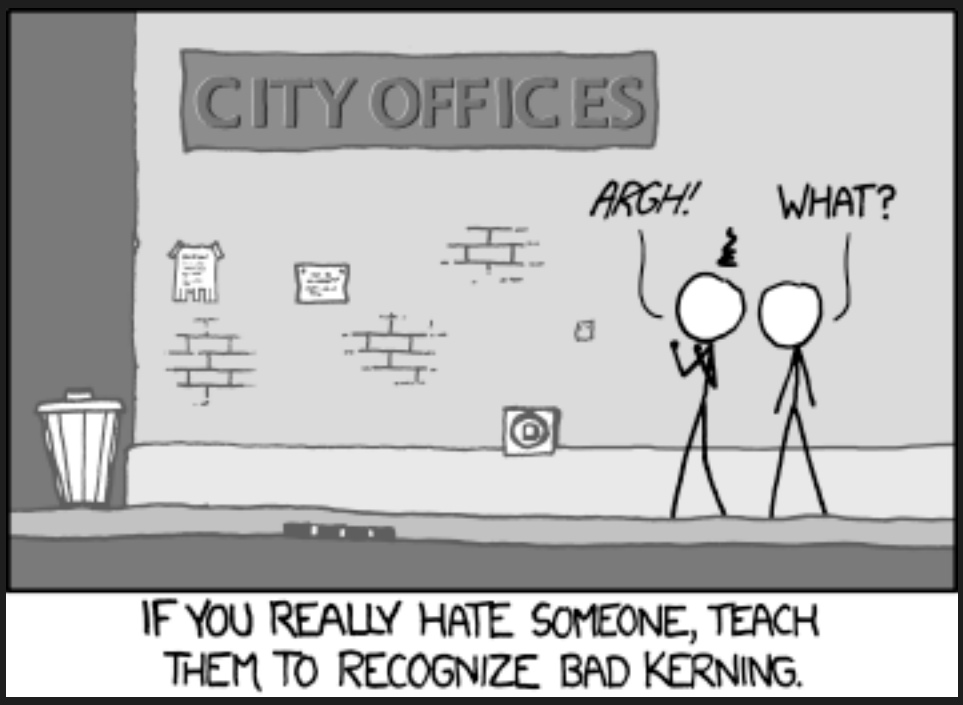

Figure 1: Kerning by xkcd

2.2 Stroke Caps & Joins

slughorn's current stroke support is mostly a proof-of-concept; it doesn't support any of the typical line caps or joins that most other vector libraries do. Adding these is entirely possible, as well as *additional* features like dynamic/realtime stroking with variable intensity, animation, and different stroke styles (dashed, dotted, or patterned with *any* `Shape` instance).

2.3 Patterns

Nothing prevents a frontend renderer from tiling slughorn `Shape` instances just like typical UV texture tiling (and we have osgSlug demos proving this). However, without a **unified** pattern primitive in slughorn itself, every renderer would be required to reimplement these features independently. slughorn can — and *should* — support patterns as a first-class primitive.

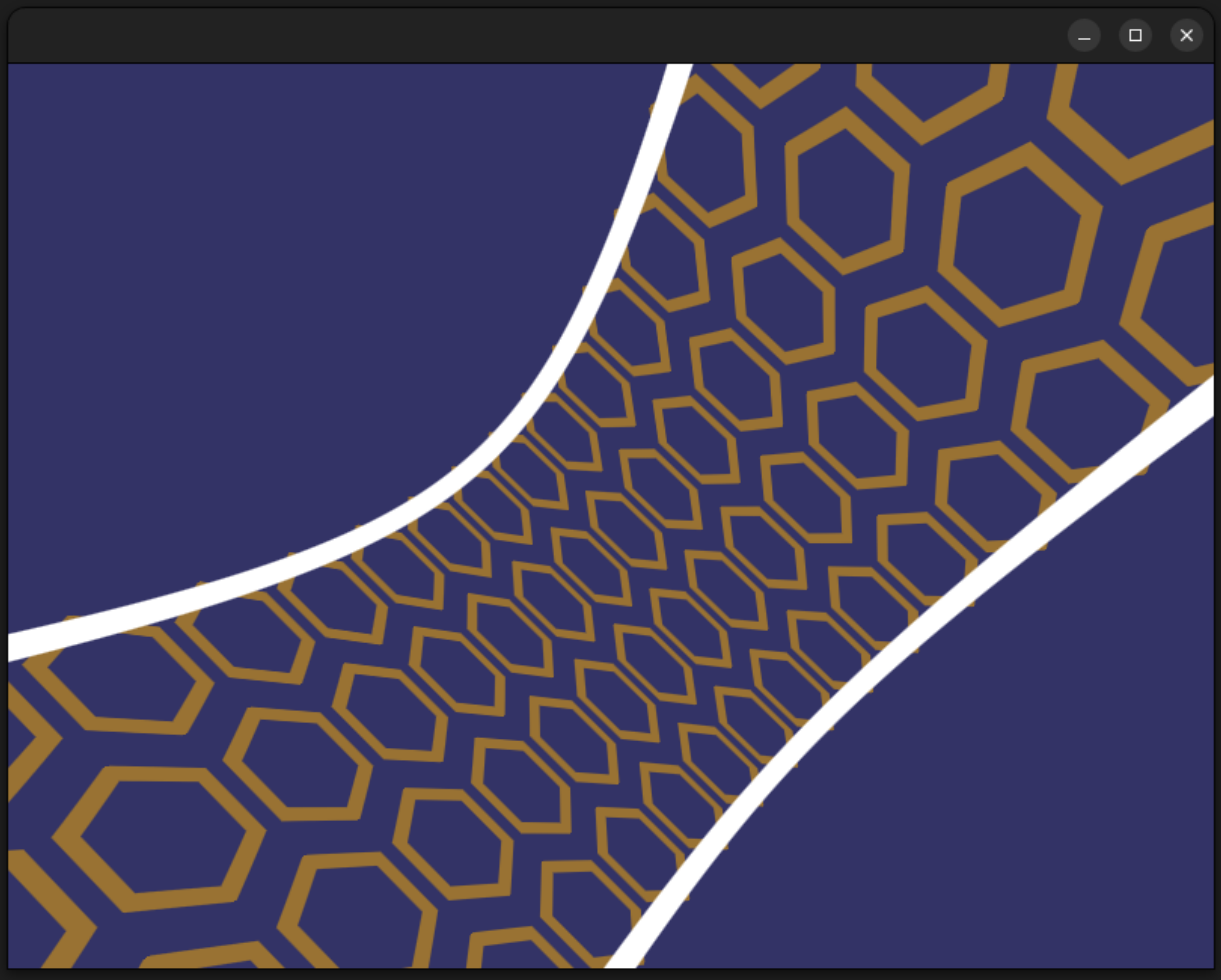

Figure 2: osgslug-hextiles

2.4 SVG/NanoSVG Improvements

We anticipate that the FreeType, NanoSVG, and native Canvas backends will likely see the *most usage* from future users. However, there remain a number of standard SVG features slughorn currently ignores, particularly strokes and text. Since slughorn is **capable** of handling those features in other backends, it's simply a matter of finding the time and support to make it happen.

2.5 WebAssembly

An Emscripten build would open up slughorn and the Slug algorithm to both browsers and Node.js. After that — *just like any other frontend* — you'd build a JavaScript renderer on **top** of it.

2.6 Offline Editor (slughorned)

The offline editor would allow users to interactively optimize the atlases created by slughorn, as well as providing additional methods for troubleshooting and debugging. A simple `PySide6` skeleton already exists, but a *full editor* would require significant additional work.

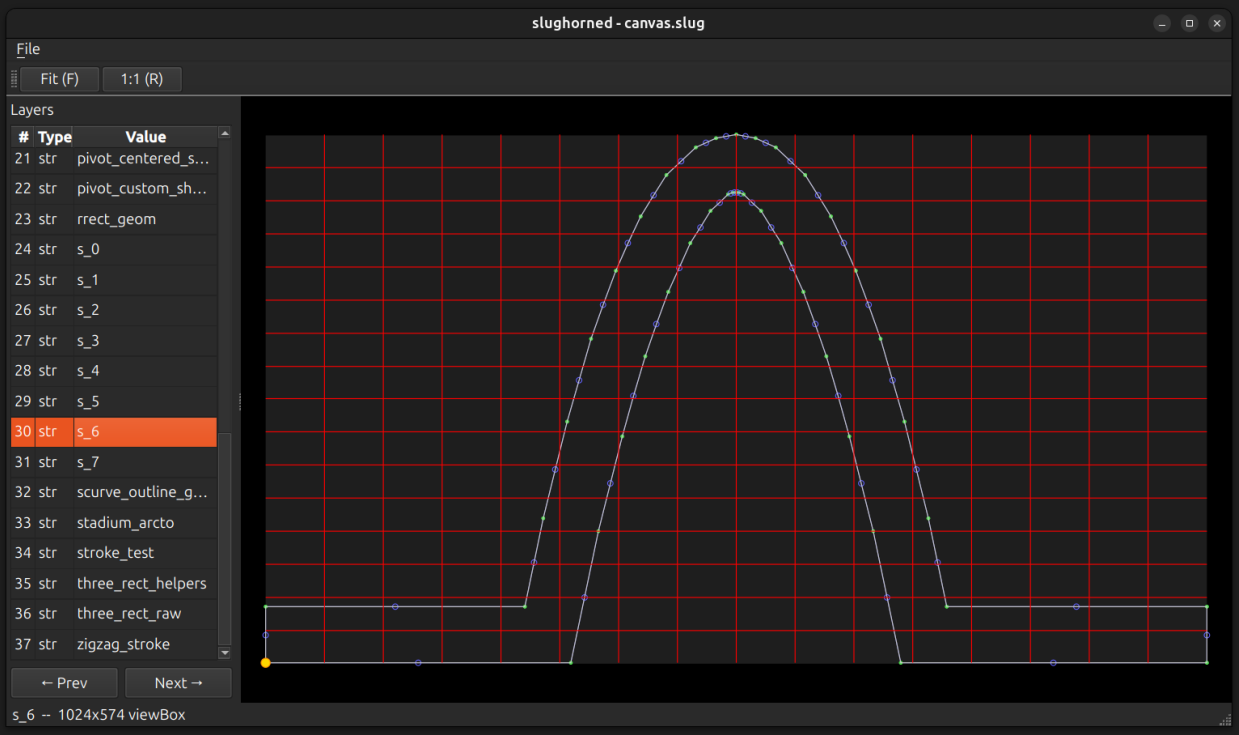

Figure 3: slughorned

2.7 Additional Renderers

osgSlug was developed alongside slughorn, acting as its default "proof-of-concept renderer." Other libraries — such as VulkanSceneGraph — could support slughorn with virtually **no friction at all**, since the core data slughorn produces (curves, atlases, layer data) is renderer-agnostic by design.

2.8 2D Text Fallbacks

Even though Slug was originally developed for dynamically rendering text glyphs at any size or in any 3D perspective, nothing can compete with the quality or *speed* of pre-rasterized, cached glyphs at a small, known pixel size. Eventually, slughorn could support both of these techniques, allowing the user to toggle between them based on *how* the text is currently being rendered — for

example, switching to pre-rasterized glyphs for 2D screen-space UI elements while keeping Slug for world-space or perspective text.

> People have been trying for **years** to match the hinted, antialiased quality of pre-rasterized glyphs in a *truly 3D* setting; needless to say, it's not an easy problem to solve. Slug gets very, **very** close, but rasterizers like FreeType have a significant advantage at small pixel sizes: they run on the CPU and know **exactly** what their pixel — and subpixel — constraints are beforehand, letting them apply stem hinting, grid-fitting, and LCD subpixel rendering that simply cannot be replicated once you lose a fixed screen-space pixel grid.

3 Research & Experimental Improvements

In addition to the previous "obvious" improvements, there are also areas of slughorn we're certain **could** be improved, but would require a bit more research and development to implement properly.

3.1 Performance

We're firm believers in avoiding both "overengineering" and "premature optimization." In its current phase, slughorn (and osgSlug) focus purely on *accurate visualization* and quality. This does **not**, however, mean performance isn't a concern — in fact, we have a handful of concrete ideas we're confident could boost performance significantly, including fragment occlusion optimization, better band placement heuristics, per-shape meshes (instead of quads), and seamless SDF/MSDF integration (see below).

3.2 GDAL Backend

GIS visualization and vector graphics go hand-in-hand, and a true "no-tessellation, GPU-only" approach to what GDAL calls `Features` is something that could provide enormous benefits to the GIS community. GDAL itself performs a lot of the hard work here; all slughorn needs to do is simplify the process of getting true dynamically-generated content **onto** the map being rendered. We're just scratching the surface of what *could be possible* here.

3.3 SDF/MSDF Fallbacks

slughorn already includes `msdfgen` as a submodule and is able to convert *any* `Shape` instance into either an SDF or MSDF tile. With some additional work, these could serve as more performant fallbacks for normal slughorn `Shape` rendering (think **vector mipmapping**), trading rendering *quality* for additional *speed* at a distance or small size.

3.4 Interactivity Layer

slughorn's primary focus is *preparing* data — baking curves and layer information into optimized GPU buffers. That said, the library already exposes primitives that hint at interactivity: the `effectData` field (available in both the GL4/SSBO and GL3 paths), per-shape user-defined origins, and more. A natural companion library could build on top of these primitives, providing hit-testing, selection, animation, and other dynamic behaviors — reducing the burden on each frontend to reinvent them independently.

3.5 UI Widgets (ImGui)

Integrating slughorn as the rendering backend for immediate-mode UI libraries like ImGui would allow them to transition into true 3D, perspective-agnostic rendering. Immediate-mode libraries

already handle the hard parts — state management and geometry generation — so the integration path is straightforward: hook into the render pipeline and substitute slughorn-compatible drawing primitives where tessellated geometry currently lives.

4 Community Input

We'd love to hear from users and potential contributors: what would *you* like to see from slughorn? What use cases are we missing? What would make it immediately useful for your project? Feedback — whether filed as an issue, discussed in a forum, or raised at a conference — directly shapes our roadmap.