GSoC 2026 Proposal

Complex Font Rendering in Haiku's app_server

Integrating HarfBuzz for Devanagari, Arabic, and Complex Script Support

Submitted by Anuj Billore | Haiku, Inc. | Google Summer of Code 2026

Full name	Anuj Billore
Timezone	IST (UTC+5:30)
Primary and Gerrit Email address	anujbillore2005@gmail.com
Mailing List email address	anujbillore3107@gmail.com
IRC username (oftc.net)	anujbillore
Trac username (dev.haiku-os.org)	anujbillore
Gerrit changes submitted	7 patches: #10446, #10447, #10448, #10449 (Merged), #10469, #10474, #10475 (Ready) at review.haiku-os.org Gerrit Page : https://review.haiku-os.org/q/owner:anujbillore2005@gmail.com
GitHub repository	
Emergency contact	
GSoC as full-time employment?	Yes
Hours per week	35 to 40 hours
Obligations that may take time away	None
University requirement?	No
How did you first hear about Haiku?	I first saw haiku way before 3 years; In the linkedIn Profile of my ‘24 Batch University Senior, who was GSoC’22 Student @Haiku I checked out his Final Report, stuff looked pretty cool at that time, It still does tbh .
Applied to other GSoC orgs?	No. Also, Since I’ve prepared two applications, I will prefer this proposal above the other one.
Estimated last day of classes/exams	8th May 2026
Estimated first day of autumn classes	28th August 2026

1. Introduction

My name is Anuj Billore, a 2027 undergraduate student studying BTECH at SGSITS, Indore, India. My primary areas of interest are systems programming, low-level software engineering, and graphics pipelines. I have been working with C++ for 3 years and am comfortable reading large unfamiliar codebases, working with Haiku's Jam-based build system, debugging native code, and submitting production-quality patches to open-source projects.

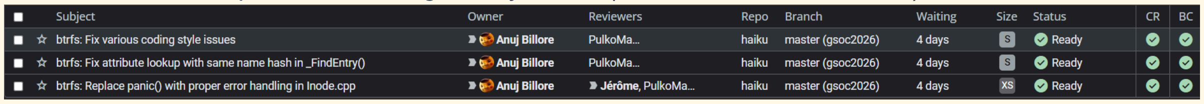
I am not a newcomer to Haiku. Before writing this proposal, I took deliberate steps to understand the codebase practically rather than just theoretically:

- **Ran Haiku nightly images on VirtualBox:** I explored the system as an end user and experienced its current text rendering limitations firsthand, seeing broken Devanagari and Arabic rendering directly, not just through screenshots.
- **Built Haiku from source:** I worked through the complete Jam toolchain, resolved build environment issues, and established the development setup I will use throughout GSoC.
- **Submitted 7 patches to Haiku's Gerrit instance** (review.haiku-os.org), all targeting the btrfs filesystem driver. These patches cover: fixing coding style issues, fixing attribute lookup with same-name hash collisions in _FindEntry(), replacing panic() with proper error handling in Inode.cpp, fixing name access and hash collision in directory lookup, adding validation checks in IsValid(), fixing pointer arithmetic in WriteSuperBlock(), and fixing code style in kernel_interface.cpp.

Of these 7 patches, **4 have already been merged into Haiku's master branch** (changes #10446, #10447, #10448, #10449). The remaining 3 (changes #10469, #10474, #10475) are reviewed and marked **Ready** for submission

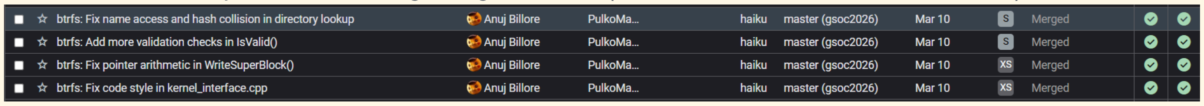
GERRIT SCREENSHOTS

Screenshot 1: three patches showing Ready status (#10469, #10474, #10475).



Subject	Owner	Reviewers	Repo	Branch	Waiting	Size	Status	CR	BC
btrfs: Fix various coding style issues	Anuj Billore	PulkoMa...	haiku	master (gsoc2026)	4 days	S	✓ Ready	✓	✓
btrfs: Fix attribute lookup with same name hash in _FindEntry()	Anuj Billore	PulkoMa...	haiku	master (gsoc2026)	4 days	S	✓ Ready	✓	✓
btrfs: Replace panic() with proper error handling in Inode.cpp	Anuj Billore	Jérôme, PulkoMa...	haiku	master (gsoc2026)	4 days	XS	✓ Ready	✓	✓

Screenshot 2: four patches showing Merged status (#10446, #10447, #10448, #10449).



btrfs: Fix name access and hash collision in directory lookup	Anuj Billore	PulkoMa...	haiku	master (gsoc2026)	Mar 10	S	Merged	✓	✓
btrfs: Add more validation checks in IsValid()	Anuj Billore	PulkoMa...	haiku	master (gsoc2026)	Mar 10	S	Merged	✓	✓
btrfs: Fix pointer arithmetic in WriteSuperBlock()	Anuj Billore	PulkoMa...	haiku	master (gsoc2026)	Mar 10	XS	Merged	✓	✓
btrfs: Fix code style in kernel_interface.cpp	Anuj Billore	PulkoMa...	haiku	master (gsoc2026)	Mar 10	XS	Merged	✓	✓

Caption: Pre-GSoC contributions to Haiku btrfs driver. 4 merged, 3 pending. Reviewer: PulkoMandy.

This pre-GSoC contribution work was intentional. I wanted to prove to myself and to the community that I can navigate Haiku's codebase, follow its strict coding standards, work with the Gerrit review workflow, and incorporate mentor feedback before asking to be trusted with a more complex project. The fact that PulkoMandy has already reviewed and approved my code means we enter GSoC with an established working relationship.

I chose this specific project because it sits at the intersection of the things I find most technically compelling: systems-level C++, Unicode text processing, graphics pipelines, and cross-cultural software accessibility. Rendering text correctly for Devanagari and Arabic is not a cosmetic problem. It is a prerequisite for Haiku to be genuinely usable by hundreds of millions of people who read and write in these scripts every day. That significance, combined with a precisely defined technical challenge and a mentor I have already collaborated with, makes this the right project for me.

I have studied all prior work on this problem:
Deepanshu Sharma's 2017 GSoC blogs and demo repository (github.com/deepango/DemoHarfbuzzApp), the 2024 forum discussion between Azam and PulkoMandy tracing the exact rendering call chain, the 2024 to 2025 RTL language support community thread, and the relevant source files including `GlyphLayoutEngine.h`, `AGGTextRenderer.cpp`, `DrawingEngine.cpp`, `Painter.cpp`, `FontEngine.cpp`, and `ServerWindow.cpp`. I am applying with a precise, file-level understanding of where the integration must happen and why previous attempts stopped short of completion.

2. Project Proposal

2.1 Title

Integrating HarfBuzz into Haiku's app_server for Complex Script Rendering

2.2 Background and Problem Statement

Haiku uses **FreeType** in combination with **AGG (Anti-Grain Geometry)** for all text rendering. This pipeline works correctly for scripts following simple left-to-right, one-codepoint-to-one-glyph rules such as Latin and Cyrillic. However, it fundamentally fails for scripts that require **text shaping** before glyphs can be selected and positioned.

Text shaping is the process of converting a Unicode string into a correctly ordered, correctly selected, and correctly positioned sequence of glyphs by applying the OpenType layout rules embedded in the font file.

To understand why this matters, consider how Unicode and fonts relate. Unicode assigns a codepoint to every character: the Hindi letter क is U+0915, the vowel sign ि is U+093F. A naive renderer maps each codepoint directly to a glyph via FT_Get_Char_Index(). This works for Latin, where characters are largely independent. Complex scripts break this assumption in two ways:

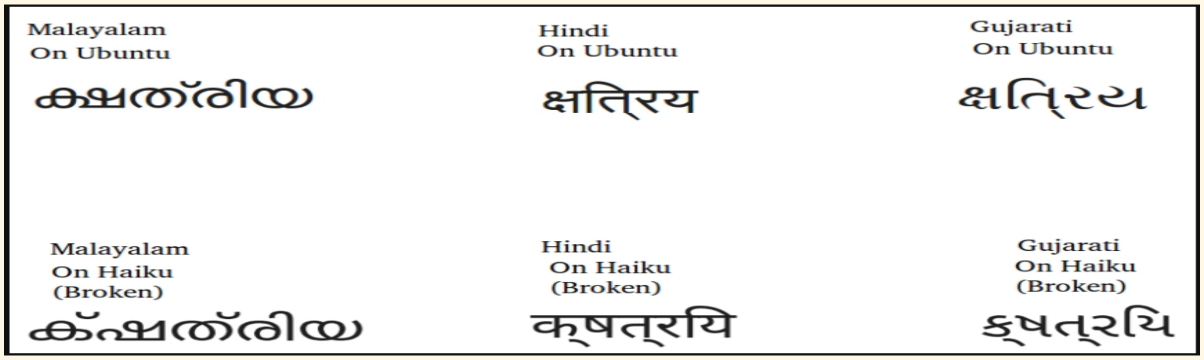
Glyph substitution (GSUB): The sequence क + ् + ष should render as the single conjunct ligature क्ष. A naive renderer draws three separate glyphs. Similarly, the Arabic letter ع has four different glyphs depending on its position in a word (isolated, initial, medial, final) - a naive renderer always returns the isolated form. These substitution rules are encoded in the font's GSUB OpenType table, which FreeType alone does not evaluate. See https://harfbuzz.github.io/what-is-harfbuzz.html for a full explanation of the shaping process, and https://learn.microsoft.com/en-us/typography/script-development/devanagari for Devanagari-specific rules.

Mark positioning (GPOS): Vowel marks in Devanagari and Arabic attach to base glyphs at precise positions defined in the font's GPOS table. Without evaluating GPOS, marks are placed at incorrect default positions.

HarfBuzz evaluates both GSUB and GPOS for a complete string in a single pass, producing final glyph IDs and precise x/y offsets. Without it:

- **Devanagari** (Hindi, Marathi, Nepali, Sanskrit, and over 120 other languages): conjuncts do not form and the output is unreadable. Trac ticket #17414.
- **Arabic** (Hebrew, Persian, Urdu): characters render left-to-right with only isolated letter forms. The output is semantically meaningless.
- **CJK scripts:** word-break rules are not applied correctly.
- Fonts with advanced OpenType tables such as Atkinson Hyperlegible render empty. Trac ticket #19268

Deepanshu's 2017 Blog Screenshot



The root cause is in `GlyphLayoutEngine.h`. The current implementation iterates over the input string one Unicode codepoint at a time using `UTF8ToCharCode()`, then requests the corresponding glyph from FreeType directly. This is correct for Latin but wrong for any script where a single glyph corresponds to multiple codepoints, or where the correct glyph depends on surrounding characters.

HarfBuzz is the industry-standard solution: an open-source OpenType text shaping engine maintained by Google and used in Chrome, Firefox, LibreOffice, GNOME, Android, and virtually every major modern rendering stack. As confirmed by PulkoMandy, HarfBuzz is already packaged for Haiku and in use in several places in the system. What does not yet exist is any integration between HarfBuzz and app_server's own rendering pipeline.

2.3 Prior Work and What Remains

2017 - Deepanshu Sharma (digib0y):

Deepanshu built a standalone demo application (github.com/deepango/DemoHarfbuzzApp) that successfully rendered Devanagari and Hindi text using HarfBuzz and FreeType directly, outside of app_server. His results were visually superior to Ubuntu 16.04's rendering of the same text at the time, a meaningful benchmark. His key technical decision was using **ICU** (already in Haiku's core) as the Unicode backend for HarfBuzz, avoiding new external dependencies. However, the project did not proceed to app_server integration. No changes were made to `GlyphLayoutEngine.h`, `AGGTextRenderer.cpp`, or any app_server source file.

2024 - Azam (forum discussion with PulkoMandy):

A 2024 GSoC applicant engaged PulkoMandy in detailed technical discussion that produced the most precise map of the integration site available. PulkoMandy traced the complete rendering call chain from application to screen (see Section 2.4.2). He explicitly identified `GlyphLayoutEngine.h` and its `UTF8ToCharCode()` loop as the primary integration point. This project also did not result in a completed implementation.

March 2025 - Community exploratory work (rvense):

A community member named rvense posted results of exploratory work in March 2025. Using an approach of adding HarfBuzz to every Jamfile that referenced FreeType, rvense confirmed:

- `hb_buffer_create()` can be called from within app_server code
- HarfBuzz headers are findable from within app_server's build
- app_server links successfully against HarfBuzz, verified via `ldd` on the output binary

This is significant: the most common invisible obstacle in library integrations (will it even link?) is already answered affirmatively. The build system path is proven. What remains is the actual shaping integration.

2026 - This proposal:

This proposal begins with a complete and verified understanding of prior work, a precise file-level map of the integration points, a realistic scoped implementation plan addressing the known gcc2 architectural constraint, and a development methodology that maximises iteration speed using test_app_server. The goal is to deliver what all prior attempts did not: working complex script rendering inside app_server itself, submitted as mergeable patches on Gerrit.

2.4 Technical Design

2.4.1 How HarfBuzz Works

HarfBuzz operates as a shaping layer that sits between raw Unicode text and a font renderer. The API workflow for a single text run is:

```
hb_buffer_t* buf = hb_buffer_create();
hb_buffer_add_utf8(buf, text, length, 0, -1);
hb_buffer_set_direction(buf, HB_DIRECTION_RTL);   // or LTR
```

```
hb_buffer_set_script(buf, HB_SCRIPT_ARABIC);      // or DEVANAGARI, LATIN
hb_buffer_set_language(buf, hb_language_from_string("ar", -1));
hb_shape(hb_font, buf, NULL, 0);                  // shaping happens here

unsigned int glyph_count;
hb_glyph_info_t*     info = hb_buffer_get_glyph_infos(buf, &glyph_count);
hb_glyph_position_t*  pos = hb_buffer_get_glyph_positions(buf, &glyph_count);
```

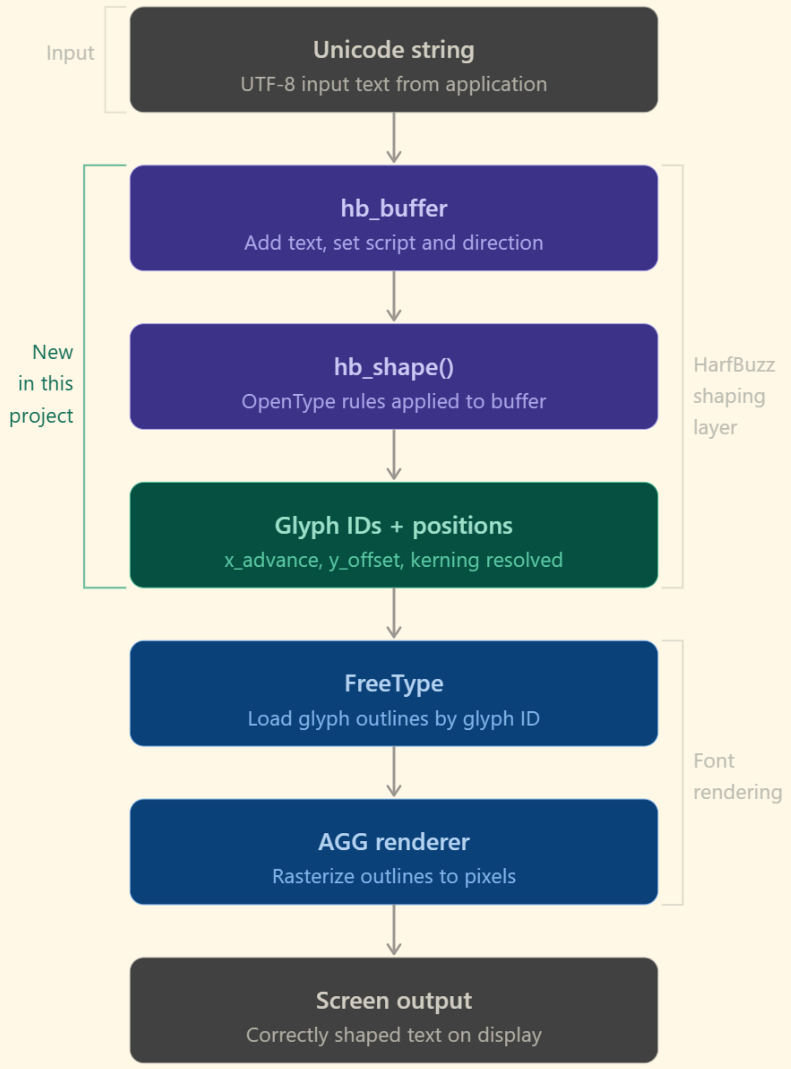
After `hb_shape()`, `glyph_info[i].codepoint` contains a **glyph ID** (an index into the font's glyph table, not a Unicode codepoint). `glyph_pos[i]` contains `x_advance`, `y_advance`, `x_offset`, `y_offset`, which are precise positioning values including kerning and mark placement.

HarfBuzz requires an `hb_font_t` object. In a FreeType-based pipeline, the cleanest way to create one is:

```
hb_font_t* hb_font = hb_ft_font_create(ft_face, NULL);
```

This wraps an existing `FT_Face` so that HarfBuzz and FreeType share the same font data with no duplication.

Harfbuzz Shaping Pipeline :



Input
Unicode string
UTF-8 input text from application
hb_buffer
Add text, set script and direction
hb_shape()
OpenType rules applied to buffer
Glyph IDs + positions
x_advance, y_offset, kerning resolved
FreeType
Load glyph outlines by glyph ID
AGG renderer
Rasterize outlines to pixels
Screen output
Correctly shaped text on display
New in this project
HarfBuzz shaping layer
Font rendering

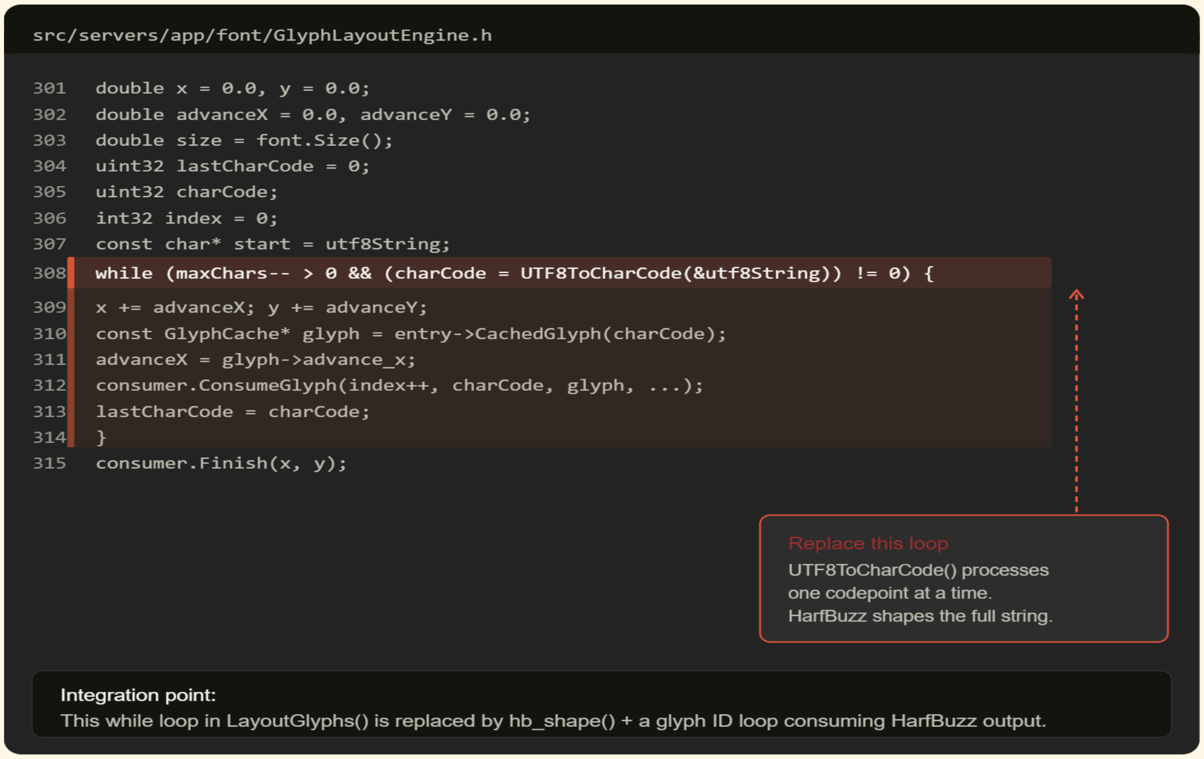
2.4.2 The Complete Rendering Call Chain

PulkoMandy traced the full path from application to screen in the 2024 forum discussion. This is the authoritative map of what needs to change:

```
Application calls BView::DrawString()
        |
        v
ServerWindow.cpp          <- receives IPC command from application
        |
        v
DrawingEngine::DrawString()    <- DrawingEngine.cpp
        |
        v
Painter.cpp               <- central drawing coordinator
        |
        v
AGGTextRenderer.cpp       <- text rendering logic
        |
        +-- StringRenderer           <- draws shaped glyphs to screen
        |
        +-- GlyphLayoutEngine.h      <- converts UTF-8 to glyphs   ** PRIMARY
TARGET **
                   |
                   v
            UTF8ToCharCode()         <- current naive per-codepoint loop
                   |
                   v
            FT_Get_Char_Index()      <- direct FreeType lookup, no shaping
```

HarfBuzz must be inserted between `GlyphLayoutEngine.h` and `FT_Get_Char_Index()`. The `UTF8ToCharCode` loop is replaced by a HarfBuzz shaping call. `StringRenderer` in `AGGTextRenderer.cpp` is updated to accept HarfBuzz's glyph IDs and position offsets rather than computing positions naively left-to-right.

Annotated Digram



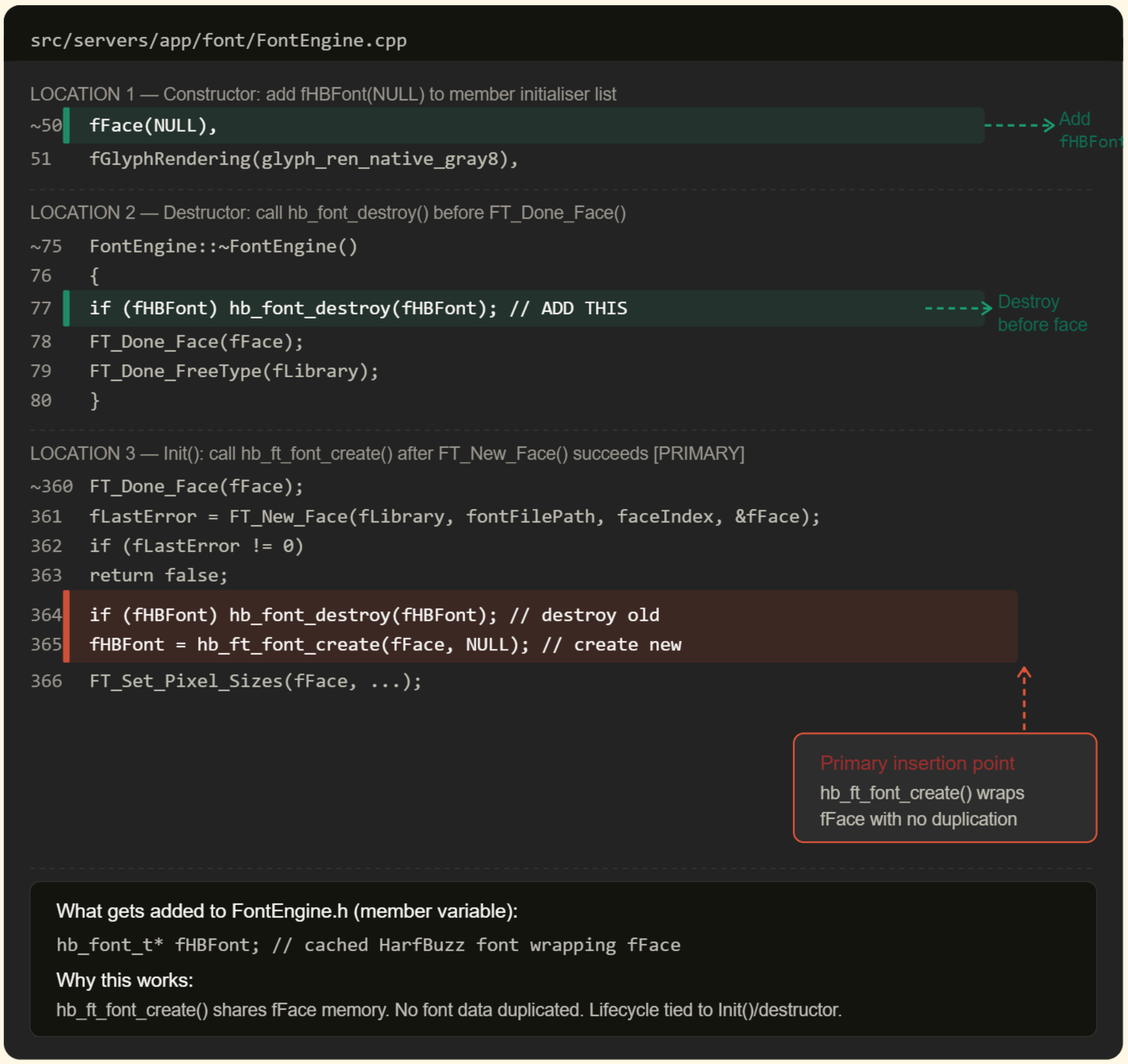
src/servers/app/font/GlyphLayoutEngine.h

```
double x = 0.0, y = 0.0;
double advanceX = 0.0, advanceY = 0.0;
double size = font.Size();
uint32 lastCharCode = 0;
uint32 charCode;
int32 index = 0;
const char* start = utf8String;
while (maxChars-- > 0 && (charCode = UTF8ToCharCode(&utf8String)) != 0) {
x += advanceX; y += advanceY;
const GlyphCache* glyph = entry->CachedGlyph(charCode);
advanceX = glyph->advance_x;
consumer.ConsumeGlyph(index++, charCode, glyph, ...);
lastCharCode = charCode;
}
consumer.Finish(x, y);
```

Replace this loop
UTF8ToCharCode() processes one codepoint at a time.
HarfBuzz shapes the full string.

Integration point:
This while loop in LayoutGlyphs() is replaced by hb_shape() + a glyph ID loop consuming HarfBuzz output.

FontEngine.cpp



```
src/servers/app/font/FontEngine.cpp
```

LOCATION 1 — Constructor: add fHBFont(NULL) to member initialiser list

```
~50  fFace(NULL),
51   fGlyphRendering(glyph_ren_native_gray8),
```

LOCATION 2 — Destructor: call hb_font_destroy() before FT_Done_Face()

```
~75  FontEngine::~FontEngine()
76   {
77   if (fHBFont) hb_font_destroy(fHBFont); // ADD THIS
78   FT_Done_Face(fFace);
79   FT_Done_FreeType(fLibrary);
80   }
```

LOCATION 3 — Init(): call hb_ft_font_create() after FT_New_Face() succeeds [PRIMARY]

```
~360 FT_Done_Face(fFace);
361  fLastError = FT_New_Face(fLibrary, fontFilePath, faceIndex, &fFace);
362  if (fLastError != 0)
363  return false;
364  if (fHBFont) hb_font_destroy(fHBFont); // destroy old
365  fHBFont = hb_ft_font_create(fFace, NULL); // create new
366  FT_Set_Pixel_Sizes(fFace, ...);
```

Primary insertion point
hb_ft_font_create() wraps fFace with no duplication

What gets added to FontEngine.h (member variable):

```
hb_font_t* fHBFont; // cached HarfBuzz font wrapping fFace
```

Why this works:

hb_ft_font_create() shares fFace memory. No font data duplicated. Lifecycle tied to Init()/destructor.

2.4.3 Current vs. Target Pipeline

Current pipeline (simplified):

```
while (string has more characters) {
    uint32 charCode   = UTF8ToCharCode(&string);
    uint32 glyphIndex = FT_Get_Char_Index(face, charCode);
    // position glyph, always advance cursor left-to-right
}
```

For Arabic, this always returns the isolated form of each letter regardless of context. For Devanagari, vowel marks are never combined with their base consonants. The output is incorrect for both scripts.

Target pipeline:

```
hb_buffer_t* buf = hb_buffer_create();
hb_buffer_add_utf8(buf, utf8_string, length, 0, -1);
hb_buffer_guess_segment_properties(buf); // auto-detect script and direction
hb_shape(hb_font, buf, NULL, 0);

unsigned int count;
hb_glyph_info_t*     info = hb_buffer_get_glyph_infos(buf, &count);
hb_glyph_position_t*  pos = hb_buffer_get_glyph_positions(buf, &count);
```

```
for (unsigned int i = 0; i < count; i++) {
    uint32 glyphId  = info[i].codepoint;
    float  xOffset  = pos[i].x_offset  / 64.0f;
    float  yOffset  = pos[i].y_offset  / 64.0f;
    float  xAdvance = pos[i].x_advance / 64.0f;  // negative for RTL
    // load glyph by glyphId, draw at (cursor + xOffset, baseline + yOffset)
    // advance cursor by xAdvance
}
hb_buffer_destroy(buf);
```

`hb_buffer_guess_segment_properties()` uses Unicode character properties to automatically detect script and direction from the string content, so app_server does not need explicit script metadata from the application for the common case.

2.4.4 Files to Modify

File	Location in Source Tree	Nature of Change
GlyphLayoutEngine.h	src/servers/app/drawing/Painter/font/	Replace UTF8ToCharCode loop with HarfBuzz shaping call. Primary change.
AGGTextRenderer.cpp	src/servers/app/drawing/Painter/	Update StringRenderer to consume HarfBuzz glyph IDs and position offsets.
FontEngine.cpp	src/servers/app/drawing/Painter/font/	Create and cache hb_font_t from existing FT_Face. Manage HarfBuzz font lifecycle. Related to Trac #18979.
DrawingEngine.cpp	src/servers/app/drawing/	Minor changes to pass direction context through the call chain if needed.
Jamfile (build system)	src/servers/app/	Declare HarfBuzz as a build dependency following the FreeType pattern.
build/jam/repositories/	Build system	Declare HarfBuzz package. Place in generated/downloads/ and build with -sHAIKU_NO_DOWNLOADS=1 for local testing.

2.4.5 The gcc2 Architectural Constraint

An important known constraint: Haiku's 32-bit x86 build of app_server is currently compiled with **gcc2**, an ancient compiler that HarfBuzz will not build with. This was confirmed by PulkoMandy in the RTL community thread. The solution path he outlined is:

- The initial implementation targets 64-bit x86 and other architectures that already use a modern compiler, where HarfBuzz builds without issue.
- Switching app_server to use gcc13 on the 32-bit build is a separate, solvable problem that does not block this GSoC project.
- Once the proof-of-concept is working on 64-bit, the gcc2 migration can be addressed as a follow-on task.

This constraint is understood and planned for. The GSoC deliverables target 64-bit Haiku, with 32-bit compatibility tracked as a post-GSoC task.

2.4.6 HarfBuzz Font Caching

Creating `hb_font_t` from an `FT_Face` on every `DrawString()` call would be a performance regression. Since app_server already manages a cache of `FT_Face` objects in `FontEngine.cpp`, the `hb_font_t` objects will be cached alongside them, one per cached `FT_Face`, and destroyed when the face is evicted from cache. This directly addresses the class of font loading inefficiencies tracked in Trac ticket #18979.

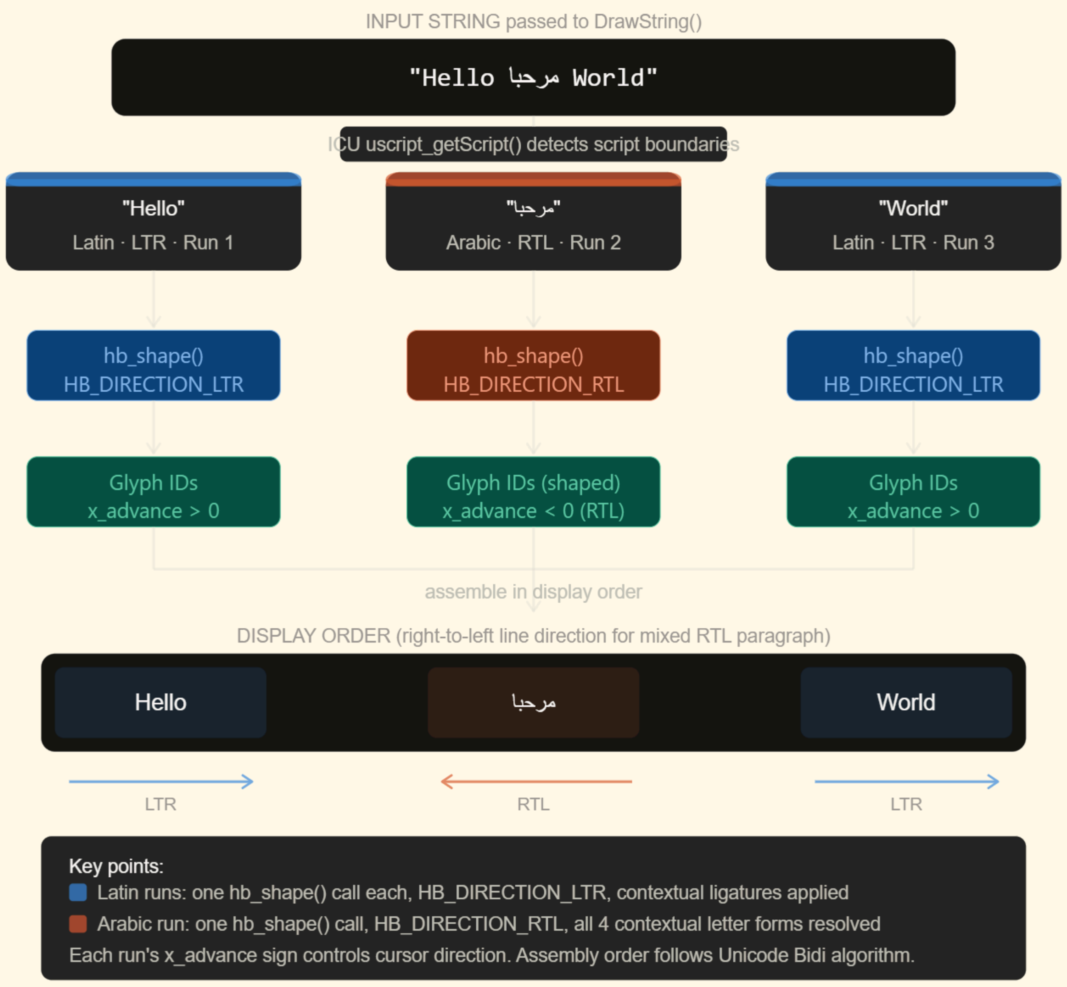
2.4.7 Text Run Segmentation

A single string passed to `DrawString()` may contain mixed scripts. For example, an English label with an embedded Arabic word. HarfBuzz must receive homogeneous runs (one script, one direction per buffer). The strategy is:

1. Scan the UTF-8 input string and detect script boundaries using ICU's `uscript_getScript()`. ICU is already in Haiku's core and is the same dependency Deepanshu used in 2017.
2. Split the string into text runs: contiguous sequences sharing the same script and direction.
3. Feed each run to HarfBuzz separately.
4. Assemble the shaped output runs in correct display order.

For the initial implementation I will begin with single-script strings to validate the core pipeline, then add run segmentation as the second major step.

How we handle mixed Scripts



Key points:
🟦 Latin runs: one hb_shape() call each, HB_DIRECTION_LTR, contextual ligatures applied
🟥 Arabic run: one hb_shape() call, HB_DIRECTION_RTL, all 4 contextual letter forms resolved
Each run's x_advance sign controls cursor direction. Assembly order follows Unicode Bidi algorithm.

2.4.8 RTL Cursor Advancement

For Arabic and other RTL scripts,PulkoMandy's feedback clarifies that there are two distinct concepts involved, and they require separate treatment:

The pen position (app_server side, in scope for this project): app_server maintains a pen position that advances after every `DrawString()` call. For RTL scripts, this advancement must be leftward rather than rightward. This project handles pen position correctly by reading `hb_buffer_get_direction()` on the shaped buffer and inverting the advancement direction accordingly. A known limitation for the MVP is that applications which draw text in small fragments rather than complete strings may encounter edge cases with bidirectional text, since app_server advances the pen sequentially without awareness of the overall paragraph direction. This is a known constraint of the current `DrawString()` API and is documented as a follow-on problem.

The visible caret and text editing in BTextView (out of scope for this project): BTextView's caret requires separate and more complex work: correct left/right arrow navigation in RTL text, correct text selection behaviour, and correct copy/paste handling. Additionally, because HarfBuzz combines multiple codepoints into a single glyph (such as a Devanagari conjunct), it may not be visually possible to position the caret between individual characters of a combined glyph. Inserting a new character into shaped text may also require redisplaying the entire word rather than just appending a new glyph, since surrounding context changes the shaping output. These complexities belong to a BTextView redesign and are explicitly tracked as post-GSoC work.
This project delivers correct pen advancement at the app_server level. BTextView caret behaviour is a larger and separable problem that builds on top of the shaping foundation this project establishes.

2.5 Trac Tickets This Work Addresses

Ticket	Title	Relationship to This Project
#17414	Ligatures support for fonts	Direct: HarfBuzz integration enables correct ligatures for all scripts
#19268	Text rendered empty with Atkinson Hyperlegible font	Direct: a proper OpenType shaping pipeline resolves this class of bug
#18979	Inefficiencies in font loading on startup	Partial: hb_font_t caching in FontEngine.cpp reduces redundant font operations

2.6 Goals

Primary (must deliver):

- HarfBuzz build system integration: declared in `build/jam/repositories/`, linked in app_server Jamfile
- `hb_font_t` creation and caching in FontEngine.cpp
- HarfBuzz shaping pipeline in `GlyphLayoutEngine.h` replacing the `UTF8ToCharCode` loop
- Updated glyph drawing in `AGGTextRenderer.cpp` consuming HarfBuzz output
- Correct Devanagari rendering via DrawString() in a test application
- Correct Arabic rendering with proper RTL pen position advancement in app_server
- Latin and Cyrillic rendering not regressed
- All changes submitted as patches on Gerrit for review

Secondary (target if primary is complete):

- Mixed-script text run segmentation using ICU
- Correct rendering in StyledEdit and other standard Haiku applications

Stretch (if time permits):

- Investigation and design document for BFont/BView API changes needed for explicit RTL support at the application level
- Initial investigation of gcc2 migration path for 32-bit x86 compatibility

3. Timeline

Community Bonding Period (May 1 to May 24)

The bonding period will be used for concrete preparation, not passive reading.

Week 1 (May 1 to May 11)

- Follow PulkoMandy's precise build system instructions: obtain the HarfBuzz package, place it in `generated/downloads/`, declare it in `build/jam/repositories/` following the FreeType pattern, and confirm it builds with `-sHAIKU_NO_DOWNLOADS=1`.
- Clone Deepanshu's 2017 demo repository (github.com/deepango/DemoHarfbuzzApp) and attempt to build it on current Haiku. Document exactly what compiles, what fails, and why. Share findings with my mentor.

Week 2 (May 12 to May 24)

- Read `GlyphLayoutEngine.h` and `AGGTextRenderer.cpp` in full. Write detailed inline comments on each section. Share the annotated version with PulkoMandy for correction.
- Reproduce rvense's March 2025 finding: confirm `hb_buffer_create()` can be called from app_server and the binary links correctly. This validates the build setup before real integration work begins.
- Write a minimal standalone C++ program outside of Haiku that uses HarfBuzz and FreeType to shape and render a Devanagari string.If Deepanshu's 2017 demo repository (github.com/deepango/DemoHarfbuzzApp) fails to build or run on current Haiku, I will write a minimal standalone C++ program using HarfBuzz and FreeType directly to shape and render a Devanagari string, establishing hands-on API familiarity before touching app_server. If the demo runs cleanly, this step is skipped and the time is redirected to annotating GlyphLayoutEngine.h and AGGTextRenderer.cpp in preparation for integration work.
- Set up the `test_app_server` development workflow as recommended by PulkoMandy. Confirm a test application can call `DrawString()` and display results without rebooting the OS.
- Establish a weekly sync schedule with Mentor.

Pre-Midterm Coding Period (May 25 to July 10)

Week 1 (May 25 to May 31)

- Implement `hb_font_t` creation in `FontEngine.cpp` using `hb_ft_font_create()`. Implement caching: one `hb_font_t` per cached `FT_Face`, destroyed on face eviction and on each `Init()` call.
- Write a test confirming the font object is created correctly from a font file and survives cache cycles.

Week 2 (June 1 to June 7)

- Modify `GlyphLayoutEngine.h`: replace the `UTF8ToCharCode` loop with a HarfBuzz shaping call, initially with script and direction hardcoded to LTR Latin. This validates the new pipeline without touching script detection yet.
- Confirm all existing Latin text rendering continues to work correctly inside `test_app_server`. Document any regressions and fix them before proceeding.

Week 3 (June 8 to June 14)

- Enable HarfBuzz shaping for Devanagari. Create a BView-based test application that calls `DrawString()` with a Hindi string. Run inside `test_app_server`.
- Debug glyph ID extraction and position offset application in `AGGTextRenderer.cpp`. Document all issues and share observations in a forum progress post.

Week 4 (June 15 to June 21)

- Achieve correct visual rendering of at least one Devanagari string in the test application. Validate conjunct consonants and vowel mark placement.
- Fix any remaining Devanagari rendering issues. Run full Latin regression tests to confirm nothing is broken.

Week 5 (June 22 to June 28)

- Extend HarfBuzz integration to Arabic. Validate that `hb_buffer_guess_segment_properties()` correctly auto-detects Arabic direction and script.
- Implement correct RTL pen position advancement in `AGGTextRenderer.cpp`, ensuring the pen advances leftward after each RTL glyph based on HarfBuzz output direction.

Week 6 (June 29 to July 5)

- Test Arabic rendering in the test application. Validate all four contextual forms (initial, medial, final, isolated) are being selected correctly.
- Compare output against a reference rendering from Firefox or LibreOffice on Linux to verify correctness.

Week 6 Buffer and Midterm Prep (July 6 to July 10)

- Fix any outstanding Arabic rendering issues from the previous week.
- Prepare midterm evaluation materials: document all modified files, all design decisions made, current rendering quality for Latin, Devanagari, and Arabic, and any open questions.
- Submit midterm evaluation to Mentor by the hard deadline of **July 10 at 18:00 UTC**.
- Midterm deliverable: Devanagari and Arabic both rendering correctly via `DrawString()` inside `test_app_server`, with Latin regression confirmed clean.

Post-Midterm Coding Period (July 10 to August 16)

Week 1 (July 10 to July 16)

- Implement text run segmentation for mixed-script strings using ICU's `uscript_getScript()`.
- Test with a mixed-script string such as "Hello مرحبا World". Each segment must render in its correct script and direction.

Week 2 (July 17 to July 23)

- Test rendering inside standard Haiku applications: StyledEdit and any application that uses `BView::DrawString()` with user-provided text.
- Fix any regressions. Document cases where rendering remains incorrect and file Trac tickets for them.

Week 3 (July 24 to July 30)

- Comprehensive regression testing across all scripts: Latin, Cyrillic, Greek, Devanagari, Arabic. Produce a testing matrix documenting results for each script in each tested application.

- Fix all remaining rendering issues identified during application testing.

Week 4 (July 31 to August 6)

- Code cleanup. Ensure all new code follows Haiku's coding style. Add comments explaining HarfBuzz-specific logic for future maintainers, particularly the relationship between shaped glyph IDs and FreeType's expectations.
- Begin submitting patches to Gerrit incrementally. Start with the build system changes and `FontEngine.cpp`, then `GlyphLayoutEngine.h`, then `AGGTextRenderer.cpp`.

Week 5 (August 7 to August 16)

- Write a technical document explaining the new pipeline: how HarfBuzz is integrated, why each design decision was made, and what a future developer needs to know to extend it or tackle the 32-bit gcc2 migration.
- Begin stretch goal if primary goals are complete: investigate what BFont and BView API changes would be required for applications to explicitly set text direction and language. Write a design document for this as a concrete proposal for a future contributor.
- All coding work complete by **August 16**. No new code after this date.

Final Submission Week (August 17 to August 24)

This week is reserved entirely for submission and documentation. No new code is written.

- Respond to mentor and community feedback on all open Gerrit patches. Gerrit patch submission is not deferred to the final week. Patches are submitted incrementally throughout the summer as each component reaches a mergeable state. Build system changes and `FontEngine.cpp` are submitted as soon as they are working and clean. `GlyphLayoutEngine.h` changes are submitted once Latin regression is confirmed. Each patch is iterated on immediately based on reviewer feedback before the next component is started. The goal is that by the final submission week, all patches are already merged or in final review, not newly submitted.
- Write the final GSoC report and blog post for the Haiku website: documenting what was built, every design decision, what remains for future contributors, and a practical guide for anyone continuing this work.
- Submit final work product and final mentor evaluation.
- Hard deadline: **August 24 at 18:00 UTC**.

After Google Summer of Code

This project does not end at the GSoC deadline.

- **Gerrit patch completion:** Ensure all submitted patches are reviewed, revised, and merged into the Haiku tree.
- **32-bit gcc2 migration:** Assist with or drive the task of switching app_server's 32-bit x86 build to gcc13, enabling HarfBuzz on the remaining architecture. This was identified by PulkoMandy as a separable follow-on task.
- **BTextView integration:** The MVP deliberately excludes BTextView. Post-GSoC I intend to investigate how cursor positioning, selection, and text editing interact with RTL text in BTextView, and contribute design proposals.
- **Interface Kit API proposals:** BFont and BView do not have APIs for setting text direction or language. A proper API design modelled on how Qt, GTK, and other toolkits handle this would be a meaningful contribution as a design document even before implementation.

- **Continued maintenance:** As Haiku users with non-Latin scripts begin using the system, bug reports will arrive. I intend to remain involved in triaging and fixing these.

4. Expectations from Mentors

I expect the following from Haiku mentors:

Technical Guidance

- Clarification on Haiku-specific conventions in app_server that are not obvious from reading the code, particularly around memory management, threading, and the locking model in the rendering pipeline.
- Review of design decisions before significant implementation effort is committed to an approach, to catch architectural mistakes early rather than late.
- Specific, actionable feedback on Gerrit patch submissions.

Scope Management

- Help identifying when a discovered problem is in scope for GSoC versus a separate project that should be tracked as a future Trac ticket. The font rendering space is large and it is easy to pursue rabbit holes.I will treat the project goals as a direction rather than a rigid contract. If the scope needs to shift during implementation, I will discuss it with you before changing course rather than pursuing rabbit holes unilaterally.

Communication

- Weekly synchronous check-in via IRC or the forum to discuss progress and blockers.
- Reasonable availability to answer questions on IRC. I understand mentors are volunteers and will flag blockers promptly so they do not stall progress for days.

Evaluation Honesty

- If my implementation approach is fundamentally wrong, I would rather know at week 2 than week 8. Frank feedback, even when it requires significant rework, is more valuable than encouragement that leads to a late-stage dead end.

I will maintain a public development log via forum posts after each major milestone throughout the summer. This serves as a communication mechanism, a progress record for the community, and documentation for any future contributor who looks at this problem and wants to understand what was tried, what worked, and what remains.

Pre-GSoC Contributions Reference

Change	Title	Status
#10446	btrfs: Fix code style in kernel_interface.cpp	Merged
#10447	btrfs: Fix pointer arithmetic in WriteSuperBlock()	Merged
#10448	btrfs: Add more validation checks in IsValid()	Merged
#10449	btrfs: Fix name access and hash collision in directory lookup	Merged
#10469	btrfs: Fix various coding style issues	Ready
#10474	btrfs: Fix attribute lookup with same name hash in _FindEntry()	Ready

Change	Title	Status
#10475	btrfs: Replace panic() with proper error handling in Inode.cpp	Ready

Reviewer on all patches: PulkoMandy
Demo link: https://review.haiku-os.org/c/haiku/+/10446

Submitted for Google Summer of Code 2026 - Haiku, Inc.