This guide is the master technical manual for **LibreSandbox**, an engine designed for the "Hardware-Direct" era. Whether you are developing on a vintage 2009 workstation or a modern console, LibreSandbox provides a high-performance framework by stripping away the abstraction layers found in modern AAA engines.

🏗️ 1. Architecture: The "Lean-Core" Philosophy

LibreSandbox is built on the principle of **Mechanical Sympathy**—designing software that works *with* the hardware rather than fighting it.

The Component Stack

- **The Heart (C11):** A high-speed, single-threaded core. It handles the window lifecycle (GLFW), memory management, and the main loop.
- **The Brain (LibreC):** A line-based interpreter that bridges game logic to the C core without the overhead of a virtual machine like C# or Python.
- **The Build (Zig):** Uses the Zig toolchain to manage C dependencies and cross-compile to different architectures (x86, ARM, i686) with a single command.

📜 2. The Universal Syntax (LibreC)

LibreC is designed to be readable by humans and lightning-fast for the parser. It is whitespace-sensitive and uses no semicolons.

🌍 World & Spawning

- `load [mesh.obj]`: Pre-calculates vertex data and sends it to the GPU.
- `tex [image.png]`: Registers a texture into VRAM.
- `spawn [mesh] [x y z]`: Places the mesh in 3D space.
- `move [id] [x y z]`: Translates an object in real-time.

🖥️ UI & Interaction

- `ui_box [x y w h] [color]`: Creates a flat UI element.
- `ui_btn "Label" [x y] [script.lbc]`: Defines a clickable area that triggers logic.
- `ui_text "String" [x y]`: Draws text using the engine's internal bitmap font.

⚡ 3. Performance & Optimization Strategies

LibreSandbox achieves a **30MB-50MB RAM footprint** through three key technical strategies:

A. Static Memory Allocation

Unlike modern engines that use "Dynamic Allocation" (which causes stuttering during memory cleanup), LibreSandbox reserves a fixed block of RAM at startup.

> **Benefit:** Zero "Garbage Collection" lag. The frame time remains perfectly flat.

B. The VRAM Bridge (VBOs)

The engine utilizes **Vertex Buffer Objects**. Instead of the CPU telling the GPU what to draw every frame, the engine "uploads" the model once.

C. Texture Atlasing

For "Demake" style games, developers should combine multiple small textures into one "Atlas." This reduces **Draw Calls**, the single biggest bottleneck on older CPUs like the Core 2 Duo.

🛠️ 4. Multi-Platform Porting Guide

Because the engine is written in pure C, it can be ported to almost anything that has a screen.

Target	Architecture	Rendering API	Difficulty
PC	x86_64	OpenGL 3.3	1/10
Old Laptop	x86 / i686	OpenGL 2.1/3.3	2/10
Consoles	x86_64	DirectX / Native	4/10

Legacy Hardware	i386	Legacy DX / nxdk	7/10
Mobile	ARM64	OpenGL ES 3.0	6/10

🚀 5. Getting Started: Your First Scene

To create a fully functional world, follow this directory structure:

1. **`/asset`** : Place your `.obj` and `.png` files here.
2. **`/src/master.lb`** : The entry point for your game.
3. **The Code**:

```
// Setup Environment
play background_noise.mp3 loop
cam_pos 0 10 20

// Load Assets
load world_map.obj
tex stone_floor.png
spawn world_map.obj 0 0 0

// Add UI
ui_text "LibreSandbox v1.0" 10 10
```

📦 6. Compiling with Zig

Zig is the secret weapon for LibreSandbox. It is not just a language; it is a **C compiler** that handles cross-compilation better than almost anything else.

Installing Zig

1. **Version:** Use **Zig 0.13.0** or **0.14.0 (Master)** for the best C compatibility in 2026.
2. **Windows:** Download the `.zip` from `ziglang.org`, extract it, and add the folder to your **Path** environment variable.
3. **Verify:** Open a terminal and type `zig version`.

The `build.zig` Script

Instead of a complex Makefile, use a `build.zig` file to compile your C engine:

```
const std = @import("std");

pub fn build(b: *std.Build) void {
    const target = b.standardTargetOptions(.{});
    const optimize = b.standardOptimizeOption(.{});

    const exe = b.addExecutable(.{
        .name = "LibreSandbox",
        .target = target,
        .optimize = optimize,
    });

    // Link C source and libraries
    exe.addCSourceFile(.{ .file = b.path("src/engine_core.c"), .flags = &[_][]const u8{"-std=c11"} });
    exe.linkLibC();

    // Example: Link GLFW for windowing
    exe.linkSystemLibrary("glfw3");

    b.installArtifact(exe);
}
```

Compilation Commands

- **Debug Build:** `zig build` (Fast compile, includes debug info).
- **High Performance Build:** `zig build -Doptimize=ReleaseFast` (This is what you want for the Core 2 Duo).
- **Cross-Compile to Windows (from Linux):** `zig build -Dtarget=x86_64-windows-gnu`.

🏆 Summary: Why Use LibreSandbox?

- **Stability:** It is immune to the "bloat" that slows down modern systems.
- **Portability:** Use the Zig compiler to build for hardware from 2005 to 2026.
- **Control:** No hidden background tasks, no telemetry, no mandatory accounts.