

CGAME

Description:

→ `cgame` is a lightweight, header-only, CPU-based C and C++ framework built from the ground up to provide a foundation for high-performance graphical applications and custom GUI systems.

→ Developed over a focused two-week period for Windows, `cgame` represents a minimal yet powerful alternative to large-scale libraries such as Raylib, SDL, or SFML, with the goal of competing on performance, efficiency, and direct developer control – while remaining completely self-contained (no external dependencies beyond the C++ standard library).

→ Unlike frameworks that rely heavily on external backends, `cgame` itself is the framework – designed to define its own rendering layer, input handling, and GUI pipeline from scratch.

→ It is engineered to run entirely on the CPU, providing predictable behavior, consistent performance, and transparent control over every pixel drawn.

→ The motive behind `cgame`'s development was to create a pure C++ environment where developers can:

- Write graphical applications without linking to large, prebuilt binaries.
- Understand every component of the rendering and GUI process.

- Have a codebase that is portable, easy to read, and built on clarity rather than complexity.

→ cgame's architecture emphasizes:

- Zero hidden abstractions – every call leads directly to visible behavior.
- Header-only structure – easy integration and compilation across toolchains.
- Cross-compiler design – tested with both MinGW and MSVC, ensuring consistency.
- Cross-platform roadmap – while currently optimized for Windows, future versions aim for full Linux and macOS support.

→ Ultimately, cgame is not a wrapper – it is an evolving standalone framework built to compete with traditional graphics libraries while staying true to the philosophy of lightweight, open-source C++ development.

Developed by :M.Hassnain.K

Overview:

The library contains functions some of them are listed

below:

```
cgame.init (); // initializes the cgame lib
CGameScreen screen = cgame.display.set_mode (width (int), height (int), `
flags); // CGameScreen is a struct, cgame.display.set_mode function set
        the initial width, height of the screen plus some flags which are
        listed below:-
        CGAME_RESIZABLE // the window can be resized
        CGAME_DPI_AWARE // sets the dpi awareness/ correction
        CGAME_OPENGL    // graphics api to be used is opengl
```

```

        CGAME_VULKAN    // graphics api to be used is vulkan
        CGAME_D3D12     // graphics api to be used is directx 12
                        (only on msvc)

cgame.display.set_title (const char *title); // sets the title of the
window
cgame.display.set_icon  (const char *iconFilePath); // the icon of the
window should be .ico file
int event = cgame.event.get (); // in the main game loop, this function
gets the current event. The events are listed below:
    cgame.QUIT          // is the quit command given
    cgame.VIDEORESIZE   // is the windows currently resizing
    cgame.KEYDOWN       // is any key held down
    cgame.KEYUP         // is any key released
cgame.quit (); // quits the window

```

// DRAWING (main loop)

```

cgame.draw.rect (int x, int y, int width, int height, int red, int
green, int blue); // draws a normal rectangle(bordered) or outlined
cgame.draw.fill_rect (int x,int y, int width, int height, int red, int
green, int blue); // draws a filled rect
cgame.draw.rounded_rect (int x,int y, int width, int height, int radius,
int border_width, int red, int green, int blue); // draws a rounded
rectangle outlined
cgame.draw.rounded_fill_rect (int x,int y, int width, int height, int
radius, int red, int green, int blue); // draws a filled rounded
rectangle

```

// IMAGE

```

CGameImage image = cgame.image.load (const char *filePath); // loads
image and
cgame.image.draw (const CGameImage *img, int x, int y); // draws the
image
cgame.image.unload (CGameImage *img); // unloads the image
cgame.image.flip_horizontal (CGameImage *img); // flips the image
horizontally
cgame.image.flip_vertical  (CGameImage *img); // flips the image
vertically
cgame.image.resize (const CGameImage *img, int w, int h); // w, h new
dimensions
cgame.image.rotate (const CGameImage *img, float rotation); // rotates
the image

```

// CONTROLS (keyboard and mouse)

if (cgame.key.pressed (cgame.K_..) // key is held down

if (cgame.key.just_pressed (cgame.K_..) // pressed once

if (cgame.key.just_released (cgame.K_..) // key is released

THE KEYS ARE:-

cgame.K_a

cgame.K_b

cgame.K_c

cgame.K_d

cgame.K_e

cgame.K_f

cgame.K_g

cgame.K_h

cgame.K_i

cgame.K_j

cgame.K_k

cgame.K_l

cgame.K_m

cgame.K_n

cgame.K_o

cgame.K_p

cgame.K_q

cgame.K_r

cgame.K_r

cgame.K_s

cgame.K_v

cgame.K_w

cgame.K_x

cgame.K_y

cgame.K_z

cgame.K_0

cgame.K_1

cgame.K_2

cgame.K_3

cgame.K_4

cgame.K_5

cgame.K_6

cgame.K_7

cgame.K_8

cgame.K_9

```
cgame.K_SPACE
cgame.K_RETURN
cgame.K_ESCAPE
cgame.K_LEFT
cgame.K_RIGHT
cgame.K_UP
cgame.K_DOWN
cgame.mouse.pressed      (..) // mouse button held down
cgame.mouse.just_pressed (..) // mouse button pressed once
cgame.mouse.just_released (..) // mouse button released
    THE BUTTONS ARE:
    CGameButtonLeft
    CGameButtonRight
    CGameButtonMiddle
```

Developer Motive & Design Philosophy

→ The development of **cgame** was driven by the need for a **pure, dependency-free, low-level GUI and rendering framework** — one that gives developers the same creative freedom as coding directly with system APIs, but with a cleaner structure and game-engine-style control.

→ Over the span of two intense weeks, **cgame** was designed, built, and tested entirely on **Windows**, written from scratch in **C++**, and structured as a **header-only library** to make integration seamless across compilers.

→ The goal was not to wrap existing frameworks, but to **create one from the ground up** — built around clarity, transparency, and performance. Every function in **cgame** is direct; there is no hidden layer or abstraction that separates the developer from the system.

→ Unlike most libraries that rely on *GPU* acceleration or *OS-level GUI* systems, **cgame runs entirely on the CPU**, making it deterministic and predictable. This design choice also makes it an ideal environment for developers interested in low-level rendering concepts and direct framebuffer manipulation.

→ The library's architecture is designed around three principles:

1. **Simplicity** — easy to read, easy to modify, and easy to extend.
2. **Performance** — real-time responsiveness without the weight of heavy engines.

3. **Transparency** — every pixel, event, and draw call is fully visible and controllable in code.

→ In its current state, cgame is focused on **Windows** with verified builds under both **MinGW** and **MSVC (Visual Studio 2022)** compilers. However, its design remains cross-compatible at a source level, ensuring that future releases can target **Linux** and **macOS** with minimal changes.

→ The long-term vision for cgame is to evolve into a **complete, cross-platform rendering and GUI framework** that matches the flexibility of modern engines while preserving its lightweight nature and header-only simplicity.

MinGW WINDOWS:-

Makefile example on mingw with cgame:-

```
CXX      = g++
CXXFLAGS = -Iinclude
LDFLAGS  = -lopengl32 -lgdi32 -lgdiplus -lws2_32 -lmsimg32 -municode
SRC      = src/main.cpp
TARGET   = main

all: $(TARGET)

$(TARGET): $(SRC)
    $(CXX) $(CXXFLAGS) $(SRC) -o $(TARGET) $(LDFLAGS)
clean:
    rm -f $(TARGET).exe
```

MSVC (Visual Studio 2022) :-

On visual studio the libraries to use are listed below:-

- opengl32.lib
- gdi32.lib
- gdiplus.lib
- msimg32.lib

→ ws2_32.lib

EXAMPLE CODE:

// Basic example: creating a window and drawing a rectangle

```
#include <cgame/cgame.h>
int main() {
    cgame.init();
    CGameScreen screen = cgame.display.set_mode(800, 600, CGAME_RESIZABLE);
    cgame.display.set_title("Hello from cgame!");

    while (true) {
        if (cgame.event.get() == cgame.QUIT) break;
        cgame.display.set_bgcolor (245,245,250);
        cgame.display.clear();
        cgame.draw.fill_rect(100, 100, 200, 150, 255, 0, 0);
        cgame.display.flip();
    }

    cgame.quit();
}
```