Subject: Dear ImGui, bloat-free graphical user interface library for C++
Posted by zaktwist on Sat, 10 Oct 2020 09:56:23 GMT
View Forum Message <> Reply to Message

Hello everybody,
I have been using U ++ for some time now and I appreciate it in all its parts but I am not an expert ...
I want to share with you all an external integration experience in TheIDE ... I accidentally found ttps://github.com/ocornut/imgui Dear ImGui, bloat-free graphical user interface library for C ++, and it seems quite simple to manage and use.
I managed to start everything with VisualStudioCommunity, but the intent was to be able to start the example Win32DirectX11 with TheIDE and after various ups and downs I succeeded ..
Maybe for many of you it would have been easy to add libraries and be able to compile everything without problems but it wasn't for me ...
I didn't want to ask anything on the forum so as not to be ridiculous in maybe simple questions and in any case I managed to compile everything both in release and in debug.
Of course in the example code I was in the docking branch ...
However we accept advice to improve and simplify the project ...
I apologize for my english, but it's google translator's fault :lol:
Hello everybody

ttps://i.ibb.co/VVhqZhb/Cattura2.jpg
ttps://i.ibb.co/P5zhYRx/Cattura3.jpg
ttps://i.ibb.co/x5Lqts9/Cattura4.jpg
ttps://i.ibb.co/swC3zML/Cattura5.jpg
ttps://i.ibb.co/Pgh38b5/Cattura7.png

Long code snipset (Click to expand)

```
#pragma comment(lib, "d3d11.lib")

#include <Core/Core.h>
using namespace Upp;


#include "imgui.h"
#include "imgui_impl_win32.h"
#include "imgui_impl_dx11.h"

#define CY win32_CY_
#include <dshow.h>
#undef CY

#include <d3d11.h>
#define DIRECTINPUT_VERSION 0x0800
#include <dinput.h>
#include <tchar.h>
```

```
// Data
static ID3D11Device*            g_pd3dDevice = NULL;
static ID3D11DeviceContext*     g_pd3dDeviceContext = NULL;
static IDXGISwapChain*          g_pSwapChain = NULL;
static ID3D11RenderTargetView*  g_mainRenderTargetView = NULL;

// Forward declarations of helper functions
bool CreateDeviceD3D(HWND hWnd);
void CleanupDeviceD3D();
void CreateRenderTarget();
void CleanupRenderTarget();
LRESULT WINAPI WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam);


static bool show_app_dockspace = true;
static void ShowExampleAppDockSpace(bool* p_open);

// Helper to display a little (?) mark which shows a tooltip when hovered.
// In your own code you may want to display an actual icon if you are using a merged icon fonts (see docs/FONTS.md)
static void HelpMarker(const char* desc)
{
    ImGui::TextDisabled("(?)");
    if (ImGui::IsItemHovered())
    {
        ImGui::BeginTooltip();
        ImGui::PushTextWrapPos(ImGui::GetFontSize() * 35.0f);
        ImGui::TextUnformatted(desc);
        ImGui::PopTextWrapPos();
        ImGui::EndTooltip();
    }
}

static void ShowDockingDisabledMessage()
{
    ImGuiIO& io = ImGui::GetIO();
    ImGui::Text("ERROR: Docking is not enabled! See Demo > Configuration.");
    ImGui::Text("Set io.ConfigFlags |= ImGuiConfigFlags_DockingEnable in your code, or ");
    ImGui::SameLine(0.0f, 0.0f);
    if (ImGui::SmallButton("click here"))
        io.ConfigFlags |= ImGuiConfigFlags_DockingEnable;
```

```
}



// Main code
int main(int, char**)
//CONSOLE_APP_MAIN
{
 ImGui_ImplWin32_EnableDpiAwareness();

 // Create application window
    //ImGui_ImplWin32_EnableDpiAwareness();
    WNDCLASSEX wc = { sizeof(WNDCLASSEX), CS_CLASSDC, WndProc, 0L, 0L,
GetModuleHandle(NULL), NULL, NULL, NULL, NULL, _T("ImGui Example"), NULL };
    ::RegisterClassEx(&wc);
    HWND hwnd = ::CreateWindow(wc.lpszClassName, _T("Dear ImGui DirectX11 Example"),
WS_OVERLAPPEDWINDOW, 100, 100, 1280, 800, NULL, NULL, wc.hInstance, NULL);


 // Initialize Direct3D
    if (!CreateDeviceD3D(hwnd))
    {
        CleanupDeviceD3D();
        ::UnregisterClass(wc.lpszClassName, wc.hInstance);
        return 1;
    }

    // Show the window
    ::ShowWindow(hwnd, SW_SHOWDEFAULT);
    ::UpdateWindow(hwnd);



    // Setup Dear ImGui context
    IMGUI_CHECKVERSION();
    ImGui::CreateContext();
    ImGuiIO& io = ImGui::GetIO(); (void)io;
    io.ConfigFlags |= ImGuiConfigFlags_NavEnableKeyboard;       // Enable Keyboard Controls
    //io.ConfigFlags |= ImGuiConfigFlags_NavEnableGamepad;      // Enable Gamepad Controls
    io.ConfigFlags |= ImGuiConfigFlags_DockingEnable;           // Enable Docking
    io.ConfigFlags |= ImGuiConfigFlags_ViewportsEnable;         // Enable Multi-Viewport / Platform
Windows
    //io.ConfigViewportsNoAutoMerge = true;
    //io.ConfigViewportsNoTaskBarIcon = true;
    //io.ConfigViewportsNoDefaultParent = true;
    //io.ConfigDockingAlwaysTabBar = true;
    //io.ConfigDockingTransparentPayload = true;
```

```
#if 1
    io.ConfigFlags |= ImGuiConfigFlags_DpiEnableScaleFonts;     // FIXME-DPI: THIS CURRENTLY DOESN'T WORK AS EXPECTED. DON'T USE IN USER APP!
    io.ConfigFlags |= ImGuiConfigFlags_DpiEnableScaleViewports; // FIXME-DPI
#endif


    // Setup Dear ImGui style
    ImGui::StyleColorsDark();
    //ImGui::StyleColorsClassic();

    // When viewports are enabled we tweak WindowRounding/WindowBg so platform windows can look identical to regular ones.
    ImGuiStyle& style = ImGui::GetStyle();
    if (io.ConfigFlags & ImGuiConfigFlags_ViewportsEnable)
    {
        style.WindowRounding = 0.0f;
        style.Colors[ImGuiCol_WindowBg].w = 1.0f;
    }

    // Setup Platform/Renderer bindings
    ImGui_ImplWin32_Init(hwnd);
    ImGui_ImplDX11_Init(g_pd3dDevice, g_pd3dDeviceContext);



    // Load Fonts
    // - If no fonts are loaded, dear imgui will use the default font. You can also load multiple fonts and use ImGui::PushFont()/PopFont() to select them.
    // - AddFontFromFileTTF() will return the ImFont* so you can store it if you need to select the font among multiple.
    // - If the file cannot be loaded, the function will return NULL. Please handle those errors in your application (e.g. use an assertion, or display an error and quit).
    // - The fonts will be rasterized at a given size (w/ oversampling) and stored into a texture when calling ImFontAtlas::Build()/GetTexDataAsXXXX(), which ImGui_ImplXXXX_NewFrame below will call.
    // - Read 'docs/FONTS.md' for more instructions and details.
    // - Remember that in C/C++ if you want to include a backslash \ in a string literal you need to write a double backslash \\ !
    //io.Fonts->AddFontDefault();
    //io.Fonts->AddFontFromFileTTF("../../misc/fonts/Roboto-Medium.ttf", 16.0f);
    //io.Fonts->AddFontFromFileTTF("../../misc/fonts/Cousine-Regular.ttf", 15.0f);
    //io.Fonts->AddFontFromFileTTF("../../misc/fonts/DroidSans.ttf", 16.0f);
    //io.Fonts->AddFontFromFileTTF("../../misc/fonts/ProggyTiny.ttf", 10.0f);
    //ImFont* font = io.Fonts->AddFontFromFileTTF("c:\\Windows\\Fonts\\ArialUni.ttf", 18.0f, NULL, io.Fonts->GetGlyphRangesJapanese());
    //IM_ASSERT(font != NULL);
```

```
    // Our state
    bool show_demo_window = true;
    bool show_another_window = false;
    ImVec4 clear_color = ImVec4(0.45f, 0.55f, 0.60f, 1.00f);

    // Main loop
    MSG msg;
    ZeroMemory(&msg, sizeof(msg));
    while (msg.message != WM_QUIT)
    {
        // Poll and handle messages (inputs, window resize, etc.)
        // You can read the io.WantCaptureMouse, io.WantCaptureKeyboard flags to tell if dear imgui wants to use your inputs.
        // - When io.WantCaptureMouse is true, do not dispatch mouse input data to your main application.
        // - When io.WantCaptureKeyboard is true, do not dispatch keyboard input data to your main application.
        // Generally you may always pass all inputs to dear imgui, and hide them from your application based on those two flags.
        if (::PeekMessage(&msg, NULL, 0U, 0U, PM_REMOVE))
        {
            ::TranslateMessage(&msg);
            ::DispatchMessage(&msg);
            continue;
        }

        // Start the Dear ImGui frame
        ImGui_ImplDX11_NewFrame();
        ImGui_ImplWin32_NewFrame();
        ImGui::NewFrame();


  ShowExampleAppDockSpace(&show_app_dockspace);

        // 1. Show the big demo window (Most of the sample code is in ImGui::ShowDemoWindow()! You can browse its code to learn more about Dear ImGui!).
        if (show_demo_window)
            ImGui::ShowDemoWindow(&show_demo_window);

        // 2. Show a simple window that we create ourselves. We use a Begin/End pair to created a named window.
        {
            static float f = 0.0f;
            static int counter = 0;

            ImGui::Begin("Hello, world!");                          // Create a window called "Hello, world!" and append into it.
```

```
            ImGui::Text("This is some useful text.");               // Display some text (you can use a
format strings too)
            ImGui::Checkbox("Demo Window", &show_demo_window);      // Edit bools storing our
window open/close state
            ImGui::Checkbox("Another Window", &show_another_window);

            ImGui::SliderFloat("float", &f, 0.0f, 1.0f);            // Edit 1 float using a slider from 0.0f to 1.0f
            ImGui::ColorEdit3("clear color", (float*)&clear_color); // Edit 3 floats representing a color

            if (ImGui::Button("Button"))                            // Buttons return true when clicked (most
widgets return true when edited/activated)
                counter++;
            ImGui::SameLine();
            ImGui::Text("counter = %d", counter);

            ImGui::Text("Application average %.3f ms/frame (%.1f FPS)", 1000.0f /
ImGui::GetIO().Framerate, ImGui::GetIO().Framerate);
            ImGui::End();
        }

        // 3. Show another simple window.
        if (show_another_window)
        {
            ImGui::Begin("Another Window", &show_another_window);   // Pass a pointer to our bool
variable (the window will have a closing button that will clear the bool when clicked)
            ImGui::Text("Hello from another window!");
            if (ImGui::Button("Close Me"))
                show_another_window = false;
            ImGui::End();
        }

        // Rendering
        ImGui::Render();
        g_pd3dDeviceContext->OMSetRenderTargets(1, &g_mainRenderTargetView, NULL);
        g_pd3dDeviceContext->ClearRenderTargetView(g_mainRenderTargetView,
(float*)&clear_color);
        ImGui_ImplDX11_RenderDrawData(ImGui::GetDrawData());

        // Update and Render additional Platform Windows
        if (io.ConfigFlags & ImGuiConfigFlags_ViewportsEnable)
        {
            ImGui::UpdatePlatformWindows();
            ImGui::RenderPlatformWindowsDefault();
        }

        g_pSwapChain->Present(1, 0); // Present with vsync
        //g_pSwapChain->Present(0, 0); // Present without vsync
    }
```

```
    // Cleanup
    ImGui_ImplDX11_Shutdown();
    ImGui_ImplWin32_Shutdown();
    ImGui::DestroyContext();

    CleanupDeviceD3D();
    ::DestroyWindow(hwnd);
    ::UnregisterClass(wc.lpszClassName, wc.hInstance);


    return 0;
}



void ShowExampleAppDockSpace(bool* p_open)
{
    static bool opt_fullscreen = true;
    static bool opt_padding = false;
    static ImGuiDockNodeFlags dockspace_flags = ImGuiDockNodeFlags_None;

    // We are using the ImGuiWindowFlags_NoDocking flag to make the parent window not
dockable into,
    // because it would be confusing to have two docking targets within each others.
    ImGuiWindowFlags window_flags = ImGuiWindowFlags_MenuBar |
ImGuiWindowFlags_NoDocking;
    if (opt_fullscreen)
    {
        ImGuiViewport* viewport = ImGui::GetMainViewport();
        ImGui::SetNextWindowPos(viewport->GetWorkPos());
        ImGui::SetNextWindowSize(viewport->GetWorkSize());
        ImGui::SetNextWindowViewport(viewport->ID);
        ImGui::PushStyleVar(ImGuiStyleVar_WindowRounding, 0.0f);
        ImGui::PushStyleVar(ImGuiStyleVar_WindowBorderSize, 0.0f);
        window_flags |= ImGuiWindowFlags_NoTitleBar | ImGuiWindowFlags_NoCollapse |
ImGuiWindowFlags_NoResize | ImGuiWindowFlags_NoMove;
        window_flags |= ImGuiWindowFlags_NoBringToFrontOnFocus |
ImGuiWindowFlags_NoNavFocus;
    }
    else
    {
        dockspace_flags &= ~ImGuiDockNodeFlags_PassthruCentralNode;
    }

    // When using ImGuiDockNodeFlags_PassthruCentralNode, DockSpace() will render our
background
    // and handle the pass-thru hole, so we ask Begin() to not render a background.
```

```
    if (dockspace_flags & ImGuiDockNodeFlags_PassthruCentralNode)
        window_flags |= ImGuiWindowFlags_NoBackground;

    // Important: note that we proceed even if Begin() returns false (aka window is collapsed).
    // This is because we want to keep our DockSpace() active. If a DockSpace() is inactive,
    // all active windows docked into it will lose their parent and become undocked.
    // We cannot preserve the docking relationship between an active window and an inactive
docking, otherwise
    // any change of dockspace/settings would lead to windows being stuck in limbo and never
being visible.
    if (!opt_padding)
        ImGui::PushStyleVar(ImGuiStyleVar_WindowPadding, ImVec2(0.0f, 0.0f));
    ImGui::Begin("DockSpace Demo", p_open, window_flags);
    if (!opt_padding)
        ImGui::PopStyleVar();

    if (opt_fullscreen)
        ImGui::PopStyleVar(2);

    // DockSpace
    ImGuiIO& io = ImGui::GetIO();
    if (io.ConfigFlags & ImGuiConfigFlags_DockingEnable)
    {
        ImGuiID dockspace_id = ImGui::GetID("MyDockSpace");
        ImGui::DockSpace(dockspace_id, ImVec2(0.0f, 0.0f), dockspace_flags);
    }
    else
    {
        ShowDockingDisabledMessage();
    }

    if (ImGui::BeginMenuBar())
    {
        if (ImGui::BeginMenu("Options"))
        {
            // Disabling fullscreen would allow the window to be moved to the front of other windows,
            // which we can't undo at the moment without finer window depth/z control.
            ImGui::MenuItem("Fullscreen", NULL, &opt_fullscreen);
            ImGui::MenuItem("Padding", NULL, &opt_padding);
            ImGui::Separator();

            if (ImGui::MenuItem("Flag: NoSplit",                "", (dockspace_flags &
ImGuiDockNodeFlags_NoSplit) != 0))                { dockspace_flags ^=
ImGuiDockNodeFlags_NoSplit; }
            if (ImGui::MenuItem("Flag: NoResize",               "", (dockspace_flags &
ImGuiDockNodeFlags_NoResize) != 0))               { dockspace_flags ^=
ImGuiDockNodeFlags_NoResize; }
            if (ImGui::MenuItem("Flag: NoDockingInCentralNode", "", (dockspace_flags &
```

```
ImGuiDockNodeFlags_NoDockingInCentralNode) != 0))  { dockspace_flags ^=
ImGuiDockNodeFlags_NoDockingInCentralNode; }
            if (ImGui::MenuItem("Flag: AutoHideTabBar",          "", (dockspace_flags &
ImGuiDockNodeFlags_AutoHideTabBar) != 0))          { dockspace_flags ^=
ImGuiDockNodeFlags_AutoHideTabBar; }
            if (ImGui::MenuItem("Flag: PassthruCentralNode",     "", (dockspace_flags &
ImGuiDockNodeFlags_PassthruCentralNode) != 0, opt_fullscreen)) { dockspace_flags ^=
ImGuiDockNodeFlags_PassthruCentralNode; }
            ImGui::Separator();

            if (ImGui::MenuItem("Close", NULL, false, p_open != NULL))
                *p_open = false;
            ImGui::EndMenu();
        }
        HelpMarker(
            "When docking is enabled, you can ALWAYS dock MOST window into another! Try it
now!" "\n\n"
            " > if io.ConfigDockingWithShift==false (default):" "\n"
            "   drag windows from title bar to dock" "\n"
            " > if io.ConfigDockingWithShift==true:" "\n"
            "   drag windows from anywhere and hold Shift to dock" "\n\n"
            "This demo app has nothing to do with it!" "\n\n"
            "This demo app only demonstrate the use of ImGui::DockSpace() which allows you to
manually create a docking node _within_ another window. This is useful so you can decorate your
main application window (e.g. with a menu bar)." "\n\n"
            "ImGui::DockSpace() comes with one hard constraint: it needs to be submitted _before_
any window which may be docked into it. Therefore, if you use a dock spot as the central point of
your application, you'll probably want it to be part of the very first window you are submitting to
imgui every frame." "\n\n"
            "(NB: because of this constraint, the implicit \"Debug\" window can not be docked into an
explicit DockSpace() node, because that window is submitted as part of the NewFrame() call. An
easy workaround is that you can create your own implicit \"Debug##2\" window after calling
DockSpace() and leave it in the window stack for anyone to use.)"
        );

        ImGui::EndMenuBar();
    }

    ImGui::End();
}



// Helper functions

bool CreateDeviceD3D(HWND hWnd)
{
    // Setup swap chain
```

```
    DXGI_SWAP_CHAIN_DESC sd;
    ZeroMemory(&sd, sizeof(sd));
    sd.BufferCount = 2;
    sd.BufferDesc.Width = 0;
    sd.BufferDesc.Height = 0;
    sd.BufferDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
    sd.BufferDesc.RefreshRate.Numerator = 60;
    sd.BufferDesc.RefreshRate.Denominator = 1;
    sd.Flags = DXGI_SWAP_CHAIN_FLAG_ALLOW_MODE_SWITCH;
    sd.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;
    sd.OutputWindow = hWnd;
    sd.SampleDesc.Count = 1;
    sd.SampleDesc.Quality = 0;
    sd.Windowed = TRUE;
    sd.SwapEffect = DXGI_SWAP_EFFECT_DISCARD;

    UINT createDeviceFlags = 0;
    //createDeviceFlags |= D3D11_CREATE_DEVICE_DEBUG;
    D3D_FEATURE_LEVEL featureLevel;
    const D3D_FEATURE_LEVEL featureLevelArray[2] = { D3D_FEATURE_LEVEL_11_0,
D3D_FEATURE_LEVEL_10_0, };
    if (D3D11CreateDeviceAndSwapChain(NULL, D3D_DRIVER_TYPE_HARDWARE, NULL,
createDeviceFlags, featureLevelArray, 2, D3D11_SDK_VERSION, &sd, &g_pSwapChain,
&g_pd3dDevice, &featureLevel, &g_pd3dDeviceContext) != S_OK)
        return false;

    CreateRenderTarget();
    return true;
}

void CleanupDeviceD3D()
{
    CleanupRenderTarget();
    if (g_pSwapChain) { g_pSwapChain->Release(); g_pSwapChain = NULL; }
    if (g_pd3dDeviceContext) { g_pd3dDeviceContext->Release(); g_pd3dDeviceContext = NULL; }
    if (g_pd3dDevice) { g_pd3dDevice->Release(); g_pd3dDevice = NULL; }
}

void CreateRenderTarget()
{
    ID3D11Texture2D* pBackBuffer;
    g_pSwapChain->GetBuffer(0, IID_PPV_ARGS(&pBackBuffer));
    g_pd3dDevice->CreateRenderTargetView(pBackBuffer, NULL, &g_mainRenderTargetView);
    pBackBuffer->Release();
}

void CleanupRenderTarget()
{
```

```
    if (g_mainRenderTargetView) { g_mainRenderTargetView->Release();
g_mainRenderTargetView = NULL; }
}

#ifndef WM_DPICHANGED
#define WM_DPICHANGED 0x02E0 // From Windows SDK 8.1+ headers
#endif

// Forward declare message handler from imgui_impl_win32.cpp
extern IMGUI_IMPL_API LRESULT ImGui_ImplWin32_WndProcHandler(HWND hWnd, UINT
msg, WPARAM wParam, LPARAM lParam);

// Win32 message handler
LRESULT WINAPI WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    if (ImGui_ImplWin32_WndProcHandler(hWnd, msg, wParam, lParam))
        return true;

    switch (msg)
    {
    case WM_SIZE:
        if (g_pd3dDevice != NULL && wParam != SIZE_MINIMIZED)
        {
            CleanupRenderTarget();
            g_pSwapChain->ResizeBuffers(0, (UINT)LOWORD(lParam), (UINT)HIWORD(lParam),
DXGI_FORMAT_UNKNOWN, 0);
            CreateRenderTarget();
        }
        return 0;
    case WM_SYSCOMMAND:
        if ((wParam & 0xfff0) == SC_KEYMENU) // Disable ALT application menu
            return 0;
        break;
    case WM_DESTROY:
        ::PostQuitMessage(0);
        return 0;
    case WM_DPICHANGED:
        if (ImGui::GetIO().ConfigFlags & ImGuiConfigFlags_DpiEnableScaleViewports)
        {
            //const int dpi = HIWORD(wParam);
            //printf("WM_DPICHANGED to %d (%.0f%%)\n", dpi, (float)dpi / 96.0f * 100.0f);
            const RECT* suggested_rect = (RECT*)lParam;
            ::SetWindowPos(hWnd, NULL, suggested_rect->left, suggested_rect->top,
suggested_rect->right - suggested_rect->left, suggested_rect->bottom - suggested_rect->top,
SWP_NOZORDER | SWP_NOACTIVATE);
        }
        break;
    }
```

```
    return ::DefWindowProc(hWnd, msg, wParam, lParam);
}
```