Chapter 10: 📽 So far, The result

```cpp
#include "amGHOST_System.hh"
#include "amVK_Instance.hh"
#include "amVK_Device.hh"

#include "amGHOST_VkSurfaceKHR.hh"
#include "amVK_Surface.hh"

#include "amVK_SwapChain.hh"
#include "amVK_ColorSpace.hh"
#include "amVK_RenderPass.hh"
#include "amVK_RenderPass_Descriptors.hh"
#include "amVK_CommandPoolMAN.hh"

int main(int argumentCount, char* argumentVector[]) {
    REY::cout << "\n";

    // ------------------------- amGHOST ----------------------------
        amGHOST_System::create_system();

        amGHOST_Window *W = amGHOST_System::heart->new_window_interface();
        W->create(L"Whatever", 0, 0, 500, 600);
    // ------------------------- amGHOST ----------------------------

    REY_LOG("");
    REY_LOG("");
    // --------------------------- amVK -----------------------------
            REY_LOG("");
        amVK_Instance::EnumerateInstanceExtensions();
        amVK_Instance::EnumerateInstanceLayerProperties();
        amVK_Instance::addTo_1D_Instance_Layers_Enabled("VK_LAYER_KHRONOS_validation");
        amVK_Instance::addTo_1D_Instance_EXTs_Enabled("VK_KHR_surface");
        amVK_Instance::addTo_1D_Instance_EXTs_Enabled(amGHOST_System::get_vulkan_os_surface_ext_name());
        amVK_Instance::CreateInstance();

            REY_LOG("");
        VkSurfaceKHR  VK_S = amGHOST_VkSurfaceKHR::create_surface(W, amVK_Instance::vk_Instance);

            REY_LOG("");
        amVK_Instance::EnumeratePhysicalDevices();
        amVK_GPUProps  *GPUProps = amVK_InstanceProps::GetARandom_GPU();
                        GPUProps->GetPhysicalDeviceQueueFamilyProperties();
                        GPUProps->EnumerateDeviceExtensionProperties();
                        GPUProps->REY_CategorizeQueueFamilies();

        amVK_Device* D = new amVK_Device(GPUProps);
            D->addTo_1D_GPU_EXTs_Enabled("VK_KHR_swapchain");
            D->CreateDevice(1);
            D->GetDeviceQueues();

            REY_LOG("")
        amVK_Surface   *S  = new amVK_Surface(VK_S);
            S->GetPhysicalDeviceSurfaceInfo();
            S->GetPhysicalDeviceSurfaceCapabilitiesKHR();
        // --------------------------- SwapChain, RenderPass, FrameBuffers ----------------------------
            REY_LOG("")
        amVK_SwapChain *SC = new amVK_SwapChain(this->S, this->D);;
            SC->konf_ImageSharingMode(VK_SHARING_MODE_EXCLUSIVE);
            SC->konf_Images(
                amVK_IF::RGBA_8bpc_UNORM,    // VK_FORMAT_R8G8B8A8_UNORM
                amVK_CS::sRGB,               // VK_COLOR_SPACE_SRGB_NONLINEAR_KHR
                amVK_IU::Color_Display       // VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
            );
            SC->konf_Compositing(
                amVK_PM::FIFO,               // VK_PRESENT_MODE_FIFO_KHR
                amVK_CC::YES,                // Clipping:- VK_TRUE
                amVK_TA::Opaque              // VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR
            );
            SC->sync_SurfCaps();             // refresh/fetch & set/sync ---> latest SurfCaps

            SC->CI.oldSwapchain     = nullptr;
            SC->CreateSwapChain();

        amVK_SwapChainIMGs *SC_IMGs = new amVK_SwapChainIMGs(this->SC);
            SC_IMGs->   GetSwapChainImagesKHR();
            SC_IMGs->CreateSwapChainImageViews();



        amVK_RenderPass *RP = new amVK_RenderPass(this->D);
            amVK_RPADes::ColorPresentation.format = SC->CI.imageFormat;

            RP->AttachmentInfos.push_back(amVK_RPADes::ColorPresentation);
            RP->SubpassInfos   .push_back(amVK_RPSDes::ColorPresentation);
            RP->Dependencies   .push_back(amVK_RPSDep::ColorPresentation);

            RP->sync_Attachments_Subpasses_Dependencies();
            RP->CreateRenderPass();

        amVK_RenderPassFBs *RP_FBs = PR->create_FrameBuffers_interface();
            RP_FBs->CreateFrameBuffers();
        // --------------------------- SwapChain, RenderPass, FrameBuffers ----------------------------



        amVK_CommandPoolMAN  *CPMAN = PR->create_CommandPoolMAN_interface();
                              CPMAN->init_CMDPool_Graphics();

CPMAN->CreateCommandPool_Graphics(amVK_Sync::CommandPoolCreateFlags::RecordBuffer_MoreThanOnce);
                              CPMAN->AllocateCommandBuffers1_Graphics(1);

        amVK_CommandBufferPrimary *CB = new amVK_CommandBufferPrimary(CPMAN->BUFFs1.Graphics[0]);
    // --------------------------- amVK ----------------------------
    REY_LOG("");
    REY_LOG("");


    // ------------------------- CleanUp & ExportJSON ----------------------------
        REY::cin.get();     // wait for terminal input
            amVK_InstancePropsEXT::Export_nilohmannJSON_EXT();
                destroy_everything_serially();         // Last Chapter, copy code from there
                W->m_amGHOST_VkSurface->destroy();
                amVK_Instance::DestroyInstance();
            W->destroy();
    // ------------------------- CleanUp & ExportJSON ----------------------------

    REY::cout << "\n";
}
```