PeakIT 2021

Aplicații grafice moderne 3D cu Vulkan în C++

Autor: Dr. Sebastian Ichim

Email: ichim.v.sebastian@gmail.com

DIAMOND

SIEMENS

nagarro

Atos

Trimble.

Raiffeisen BANK
Banking așa cum trebuie

SILVER

accenture

endava

cegeka

PARTNERS

coresi

<ScriuCod/>

CODECAMP_

MEDIA PARTNERS

CAPITAL

BizBrasov
Stirile care conteaza in Brasov

start#up

ROCK FM
It Rocks!

Despre mine...

- Programare procedurală -> Programare Orientată pe Obiecte
- Desktop -> Mobile/Cloud
- Cursuri la MI:
  - Grafică 3D: DirectX 12.0, OpenGL ES, Modern OpenGL cu Shadere 4.6
  - Șabloane de proiectare
- Am venit la cursurile AgileHub în 2012
  - Despre Agile și Scrum

- Software Development Manager

- e-mail: ichim.v.sebastian@gmail.com
- Slack: https://peakit004.slack.com
- Linkedin: https://www.linkedin.com/in/sebastian-ichim-97354737/
- Github: https://github.com/ichimv/peakit04

Development environment

- Lunarg Vulkan SDK
  - Download
- GLFW library
  - GitHub
  - Download
- GLM
  - Github
  - Download

Agenda

- Introducere în Vulkan
- Instanță, dispozitiv, extensii, cozi
- Suprafețe, SwapChain, moduri de prezentare, ImageViews
- Pipeline, framebuffers, command pool, command buffers
- Desenarea unei clepsidre

OpenGL vs Vulkan

Oreon engine: https://www.youtube.com/watch?v=hvdAVsjrQRM&t=26s

Istoria Vulkan și OpenGL

- Ce este OpenGL?
  - OpenGL (Open Graphics Library) este un API grafic cross-platform
  - Alternativa pentru DirectX pe Windows
  - Din 2018 nu mai este suportat pe macOS și iOS (Metal de la Apple pentru grafica)

- Înainte de anunțul Vulkan comunitatea aștepta o continuare a OpenGL
  - "Next generation OpenGL initiative" sau “OpenGL next”

- În 2015 Khronos a anunțat Vulkan ca succesor al OpenGL
  - Este mult mai low-level decât OpenGL și Direct3D 12
  - Alternativă la Direct3D 12

- Vulkan 1.1 a fost publicat în 2018 - suport automat pentru multiple GPU

- Vulkan 1.2 a fost publicat în 2020 - semafoare pentru sincronizare

Ce este Vulkan API?

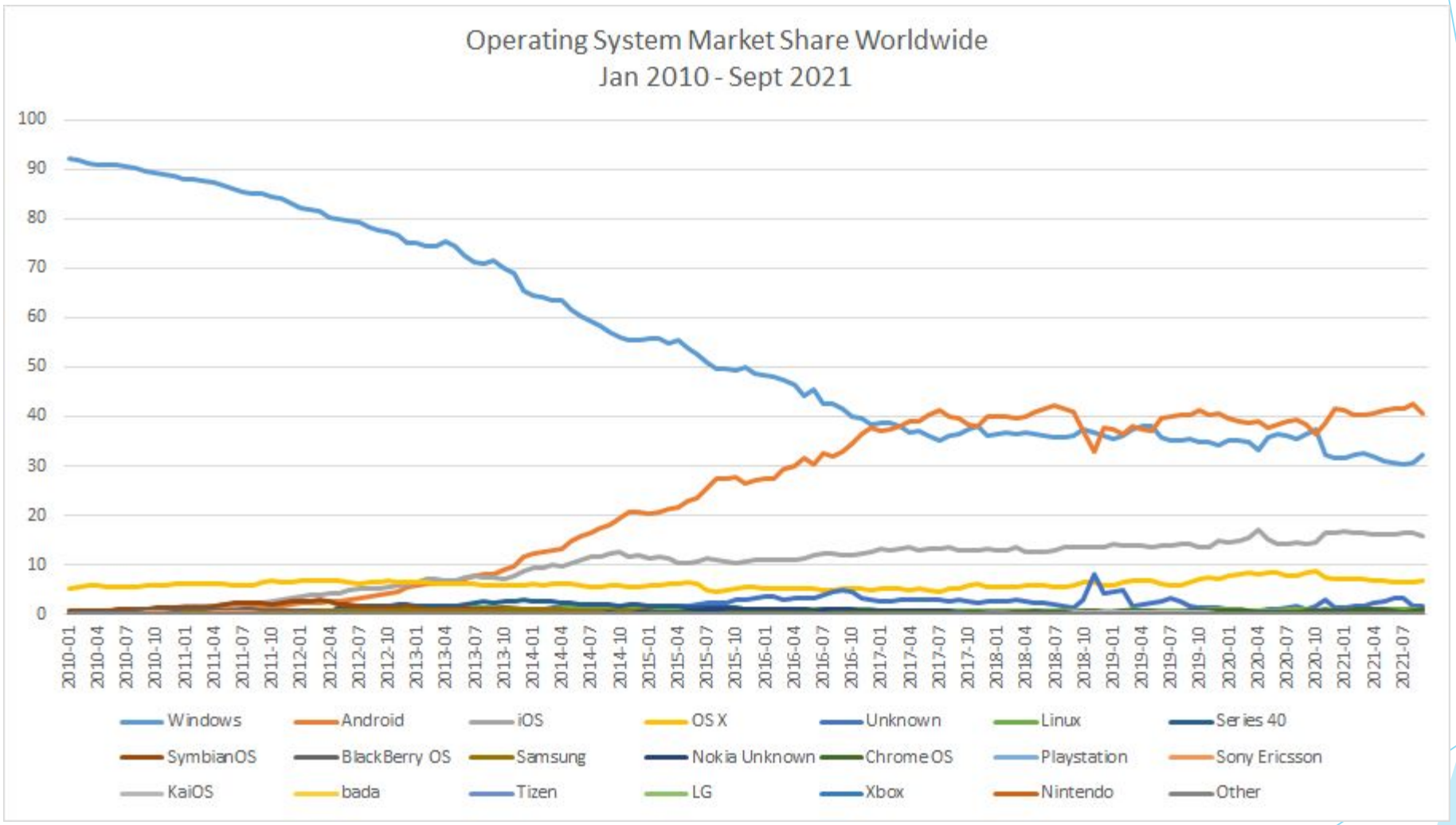
- API low-level, cross platform de grafică 3D și de calcul pe GPU

- Cross-platform - desktop/mobile
  - Windows și Linux/Android
  - MacOS/iOS necesită o mapare a apelurilor de Vulkan la framework-ul grafic Metal de la Apple
  - MoltenVK este bibliotecă gratuită open-source pe Github

- Grafică 3D
  - Procesare date grafică 3D pe GPU

- Calcul
  - Programare paralelă generică folosind paralelism bazat pe task-uri și date (OpenCL)

Cota de piață a OS la nivel mondial



Android	41%
Windows	32%
iOs	16%
OS X	7%
Unknown	2%
Chrome OS	1%

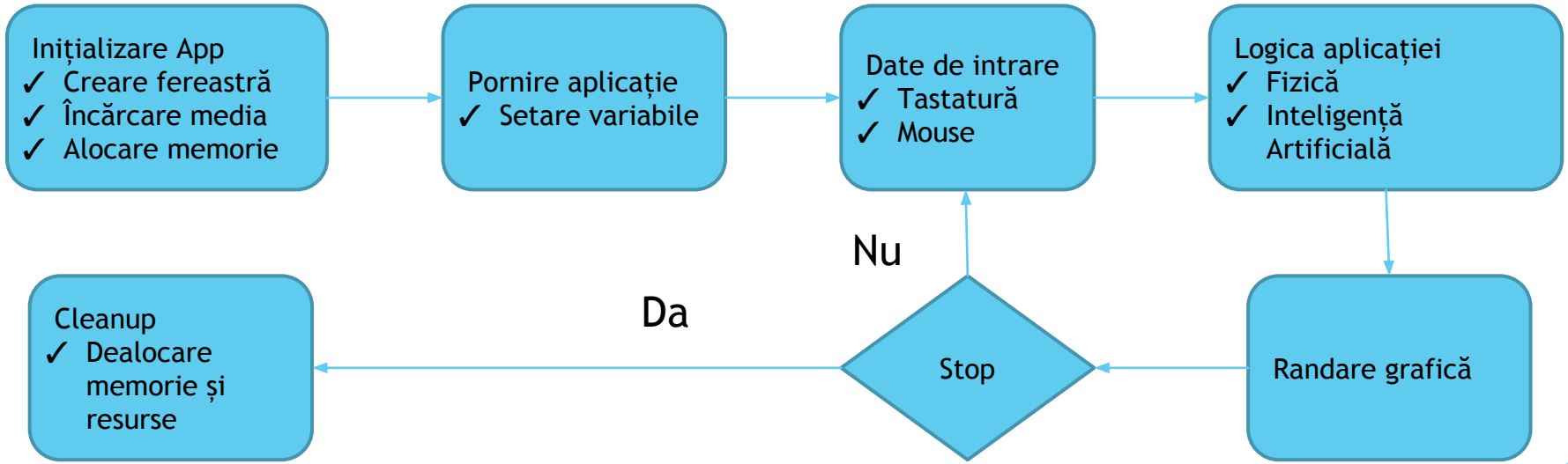
Sursa: https://gs.statcounter.com/os-market-share#monthly-201001-202109

Vulkan este foarte “stufos”

- Vulkan este foarte detaliat
  - Control asupra GPU
  - Spre deosebire de OpenGL, tratează aproape toate procesele GPU
  - Spre deosebire de Direct3D , rulează pe majoritatea platformelor
- Necesită o mulțime de cod
  - Ca să desenezi un triunghi scrii 1000 de linii de cod
- GPU este complex și Vulkan oferă acces la toată această complexitate
  - Mai mult control înseamnă mai multă optimizare
- Majoritatea codului setează valori în GPU și crează obiecte
  - Tot codul este simplu, logic și simetric, dar este foarte mult cod...

Structura unei aplicații Vulkan

```
void run() {
    initWindow();
    initVulkan();
    mainLoop();
    cleanup();
}
```



Pasul 1: Instanța Vulkan

- O instanță Vulkan este o referință la un context Vulkan
  - Aici se specifică versiunea Vulkan și capabilitățile
  - Toate aplicațiile Vulkan încep prin a crea o instanță Vulkan

- Pe instanță se setează informații despre aplicație
  - Nume
  - Versiunea Vulkan

- Extensiile accesibile instanței sunt enumerate și este verificat suportul

```
VkInstanceCreateInfo createInfo{};
createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;

VkInstance instance;
if (vkCreateInstance(&createInfo, nullptr, &instance) != VK_SUCCESS) {
        throw std::runtime_error("failed to create instance!");
}
```

Ce este un dispozitiv Vulkan?

- Dispozitivul fizic este o referință a GPU, similar cu
  - Contextul OpenGL
  - Dispozitivul DirectX

- Resursele
  - Fizice
  - Logice

- **Dispozitiv fizic**
  - este un GPU, are resurse precum memoria sau coada de comenzi pentru țeava programabilă (pipeline)
  - Nu se poate utiliza direct

- **Dispozitiv logic**
  - Interfața cu dispozitivul fizic
  - Se face setup-ul GPU

Selectarea unui dispozitivul fizic

- **Resurse ale dispozitivului fizic**
  - Memorie: alocare de memorie
  - Cozi: procesarea comenzilor pentru GPU în ordine FIFO

- **Cum se obține**
  - Nu poate fi creat

  - Se obține prin alegerea unui dispozitiv fizic din lista tuturor dispozitivelor fizice disponibile pe instanță și care răspund nevoilor aplicației

    - Dispozitivul fizic suportă cozile necesare

if (queueFamily.queueFlags & **VK_QUEUE_GRAPHICS_BIT**)

Dispozitiv logic

- Interfața pentru un dispozitiv fizic
- Utilizat/transmis intens în apelurile funcțiilor Vulkan
- Majoritatea obiectelor Vulkan se crează pe dispozitiv logic
- Folosim referința la dispozitivul logic atunci când sunt create obiectele Vulkan
- Crearea dispozitivului logic
  - Se definesc extensiile pe care dispozitivul le va utiliza
  - Se definesc familiile de cozi atașate la dispozitivul logic
  - Se definesc caracteristicile care se doresc a fi disponibile (geometry shader, linii groase)

Vulkan queues

- Dispozitivele fizice pot avea multiple tipuri de cozi
- Tipurile de cozi se mai numesc “Familii de cozi”
- O familie poate avea mai multe cozi
- Exemple de familii de cozi
  - **Grafice**: procesare comenzi grafice
  - **Computaționale**: procesare comenzi generice (cod sub formă de shadere)
  - **Transfer**: procesare operațiunilor de transfer de date
- Când parcurgem dispozitivele fizice trebuie să alegem dispozitivul care deține familiile de cozi necesare aplicației

Extensiile

- În Vulkan nu există noțiunea de fereastră
  - Vulkan este cross platform
  - Nu suportă nativ afișarea imaginilor într-o fereastră
  - Fiecare platformă definește diferit fereastra
- Vulkan folosește extensii pentru utilizarea ferestrelor
  - Afișarea imaginilor într-o fereastră
  - Swap-ul imaginilor prin mecanisme duble/triple buffer
- Extensiile vin la pachet cu Vulkan
  - Swapchain: listă de imagini prezentabile asociată cu o suprafață
  - Surface: o interfață între fereastră și o imagine din swapchain
- Utilizarea extensiilor
  - Manual
  - Utilizând librării (precum GLFW)

GLFW

- GLFW - “Graphics Library Framework”

- Inițial a fost creată pentru OpenGL, dar a fost ajustată să suporte și Vulkan

- Permite crearea de ferestre independent de platformă și interfața directă cu Vulkan/OpenGL

- Are funcții de identificare ale extensiilor Vulkan și returnează o listă a acestor extensii
  - **glfwGetRequiredInstanceExtensions**(...)

- Vulkan se configurează utilizând corect aceste extensii

- Crearea unei ferestre
  - GLFWwindow* window = **glfwCreateWindow**(WIDTH, HEIGHT, "Vulkan", nullptr, nullptr);

Suprafața ferestrei

- Interfață între fereastră și imagine din Swapchain

- Suprafața se creează specific pentru fereastra din sistem

- GLFW oferă o funcție pentru crearea unei suprafețe

```
VkInstance instance;
GLFWwindow* window;
VkSurfaceKHR surface;
if (glfwCreateWindowSurface(instance, window, nullptr, &surface) != VK_SUCCESS) {
        throw std::runtime_error("failed to create window surface!");
}
```

Coada de prezentare

- Prezintă o imagine nouă din SwapChain pe suprafața ferestrei
- Coada grafică de obicei are funcționalitatea de prezentare
- Coada grafică și coada de prezentare este de obicei aceeași coadă

```
VkQueue graphicsQueue;
vkGetDeviceQueue(device, indices.graphicsFamily.value(), 0, &graphicsQueue);

VkQueue presentQueue;
vkGetDeviceQueue(device, indices.presentFamily.value(), 0, &presentQueue);
```

Obținerea indexilor cozilor de familii

```
struct QueueFamilyIndices {
        std::optional<uint32_t> graphicsFamily;
        std::optional<uint32_t> presentFamily;
        bool isComplete() {
        return graphicsFamily.has_value() && presentFamily.has_value(); } };

QueueFamilyIndices findQueueFamilies(VkPhysicalDevice device) {
    QueueFamilyIndices indices;
    uint32_t queueFamilyCount = 0;

vkGetPhysicalDeviceQueueFamilyProperties(device, &queueFamilyCount, nullptr);
    std::vector<VkQueueFamilyProperties> queueFamilies(queueFamilyCount);

vkGetPhysicalDeviceQueueFamilyProperties(device, &queueFamilyCount, queueFamilies.data());
 int i = 0;
for (const auto& queueFamily : queueFamilies) {
        if (queueFamily.queueFlags & VK_QUEUE_GRAPHICS_BIT) {
            indices.graphicsFamily = i;
        }
        VkBool32 presentSupport = false;
        vkGetPhysicalDeviceSurfaceSupportKHR(device, i, surface, &presentSupport);
        if (presentSupport) {
            indices.presentFamily = i;
        }
        if (indices.isComplete()) {
            break;
        }
        i++;
    }
    return indices;
}
```

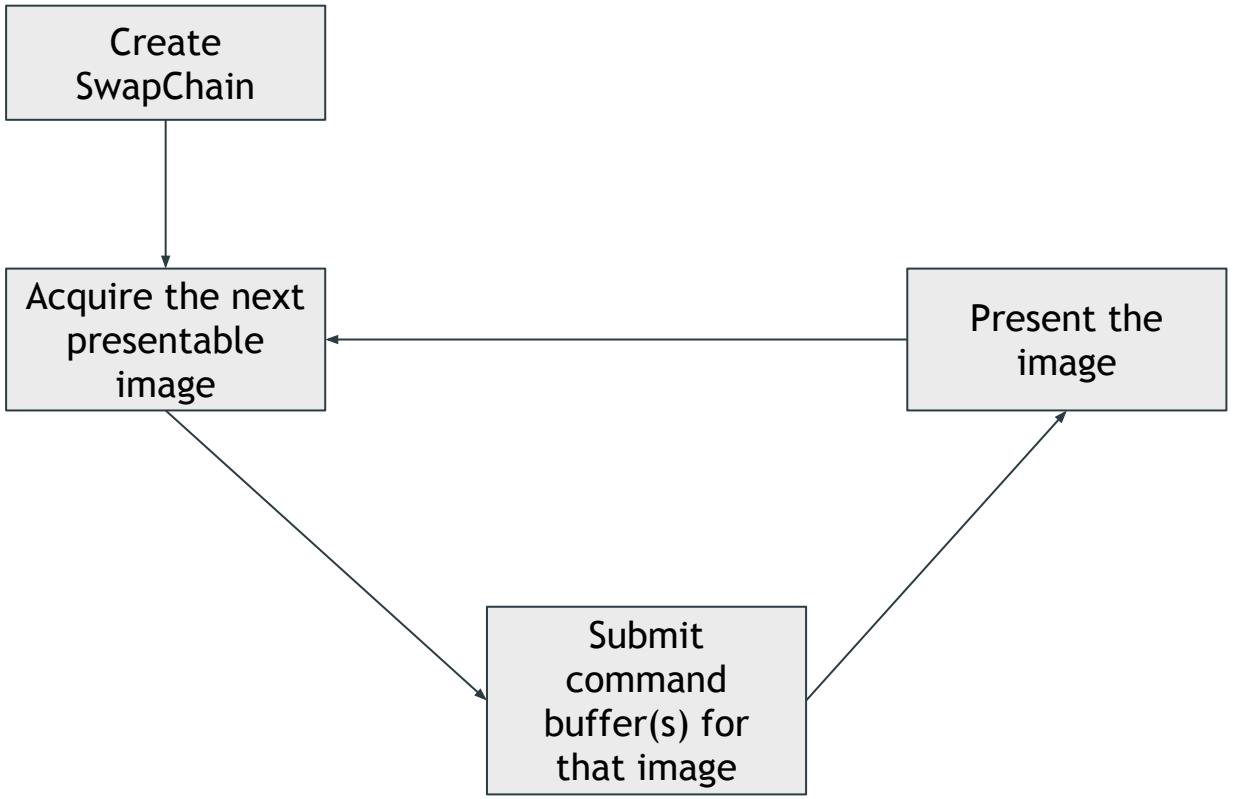
SwapChain

- Este un grup de imagini care pot fi desenate și apoi prezentate pe ecran

- Părțile unui SwapChain
  - **Surface capabilities**: ce suprafață poate fi manipulată (dimensiunea imaginii)
  - **Surface formats**: ce format poate avea suprafața (RGBA)
  - **Presentation modes**: în ce ordine imaginile sunt prezentate pe ecran

```
struct SwapChainSupportDetails {
    VkSurfaceCapabilitiesKHR capabilities;
    std::vector<VkSurfaceFormatKHR> formats;
    std::vector<VkPresentModeKHR> presentModes;
};
```

Vulkan Frame Loop

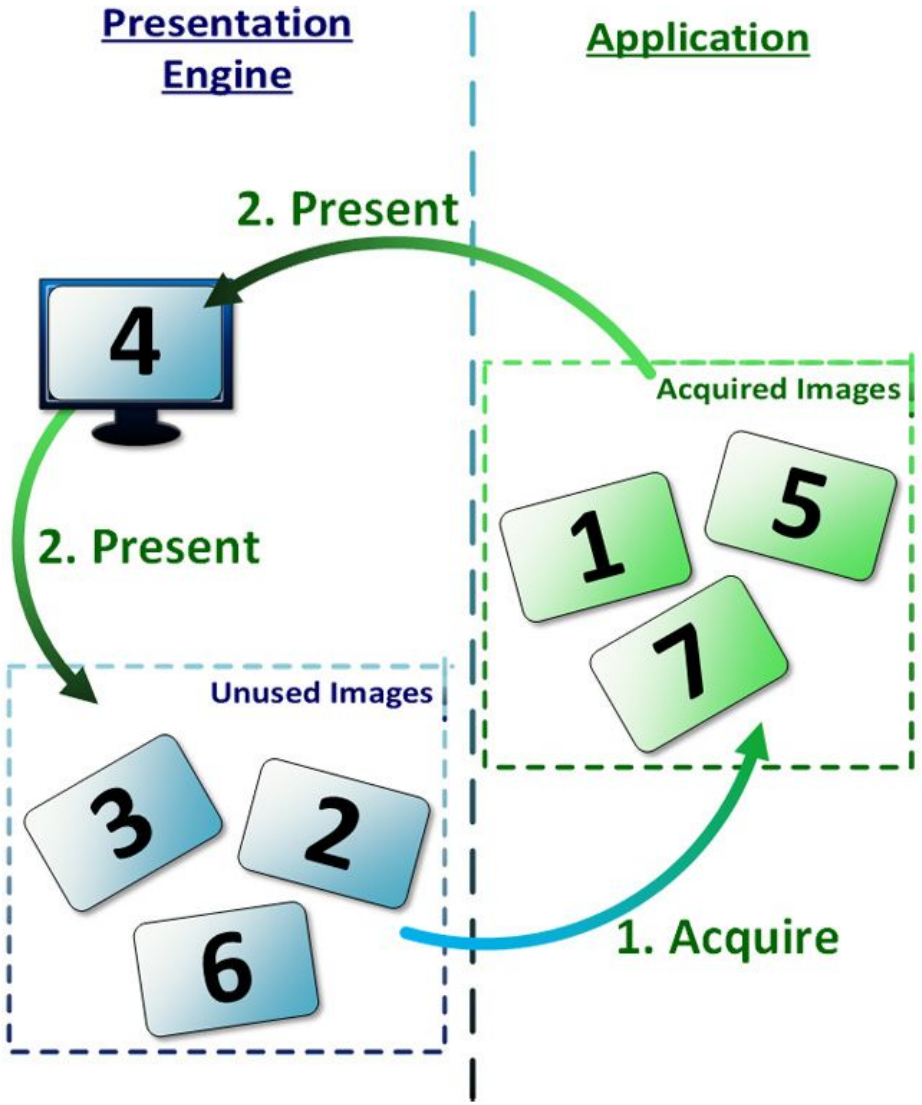


Sursa: https://www.khronos.org/assets/uploads/developers/library/2016-vulkan-devday-uk/8-swapchains-unch[illegible]

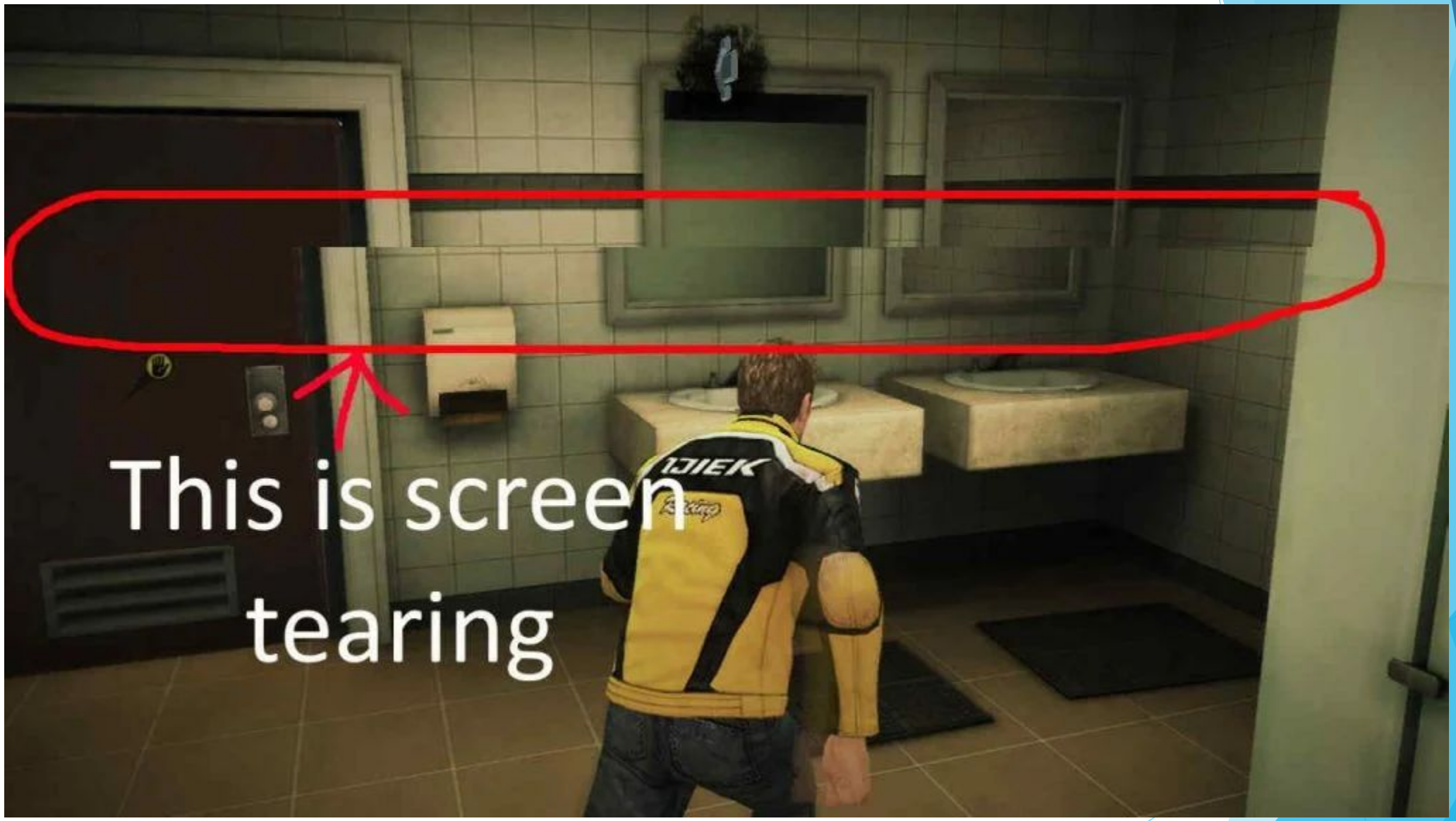
Moduri de prezentare

- La un moment dat pe ecran este prezentată o imagine din SwapChain
- Desenarea pe ecran folosește “V-Blank” sau “V-Sync”
- Tipuri de moduri de prezentare
  a. VK_PRESENT_MODE_IMMEDIATE_KHR
  b. **VK_PRESENT_MODE_MAILBOX_KHR**
  c. **VK_PRESENT_MODE_FIFO_KHR**
  d. VK_PRESENT_MODE_FIFO_RELAXED_KHR

VK_PRESENT_MODE_IMMEDIATE_KHR

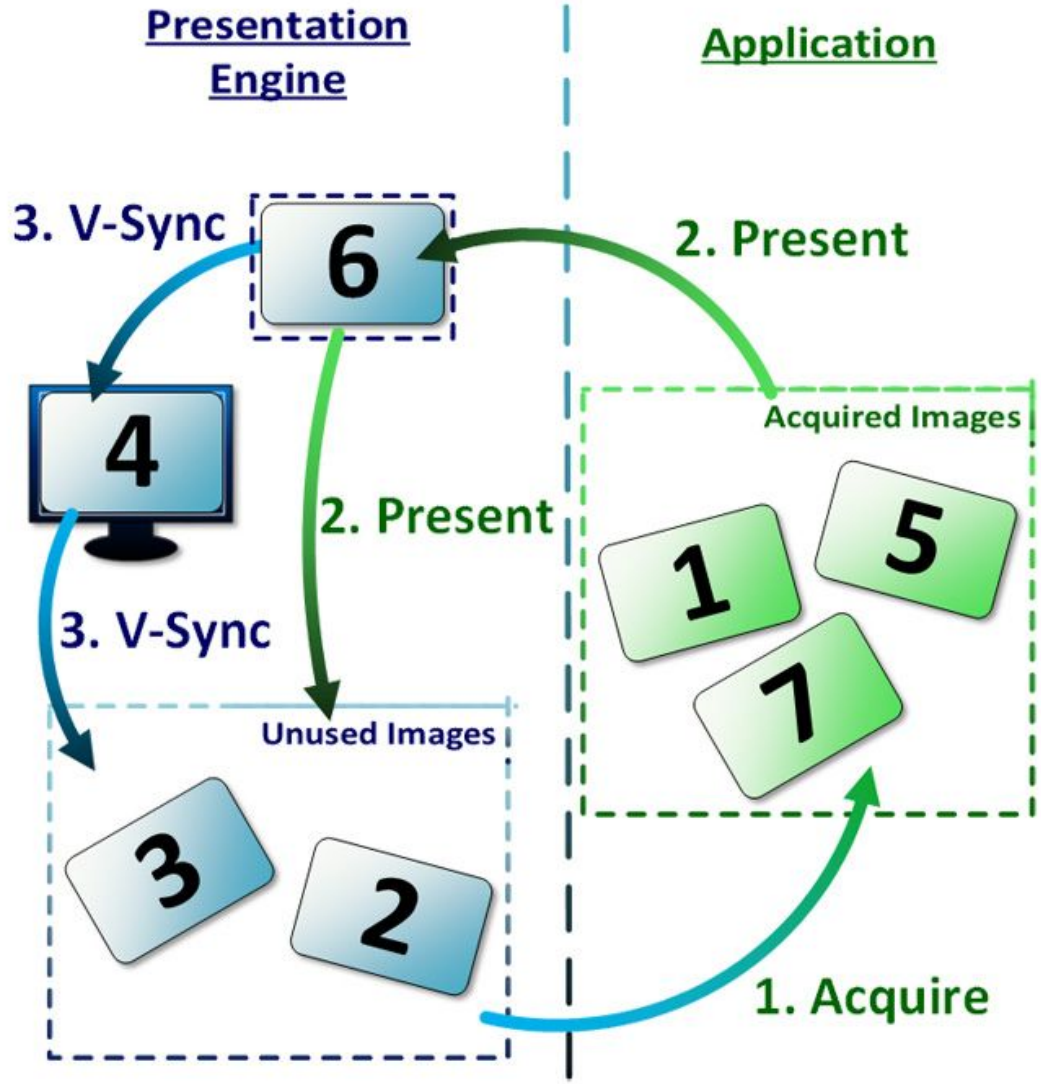


Sursa: https://www.intel.com/content/dam/develop/external/us/en/images/api-vulkan-part-2-graph



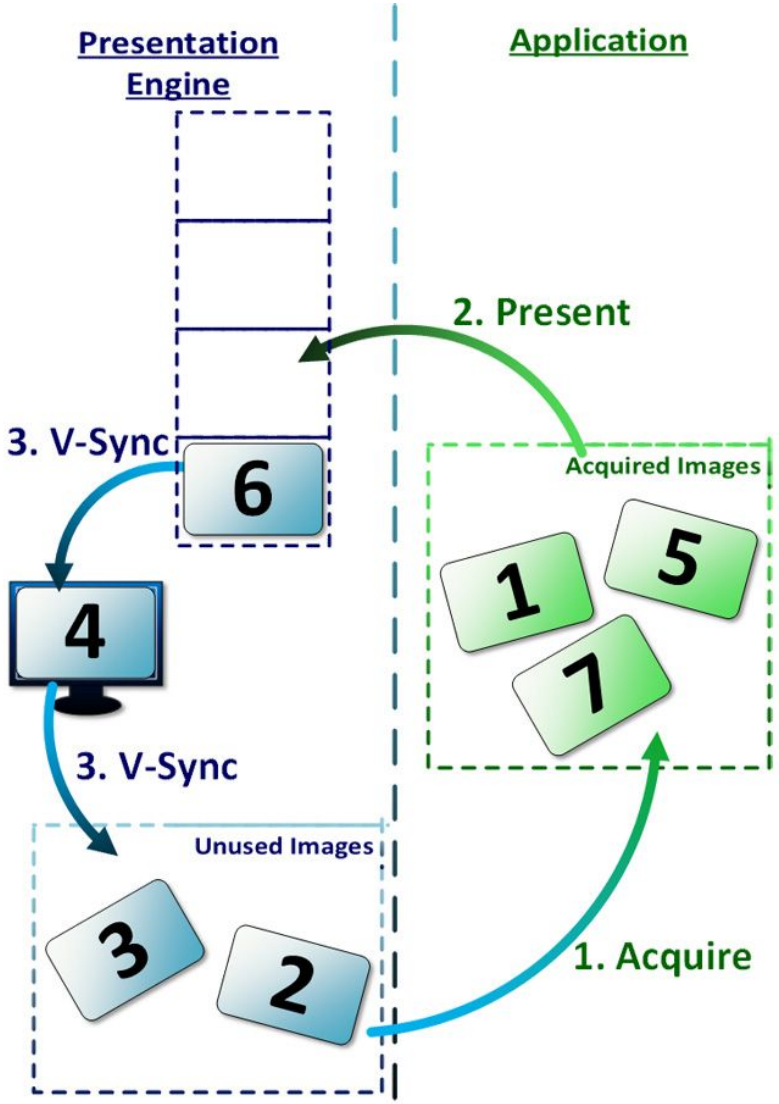
Sursa: https://nerdburglars.net/what-is-screen-tearing/

VK_PRESENT_MODE_MAILBOX_KHR



Sursa: https://www.intel.com/content/dam/develop/external/us/en/images/api-vulkan-part[illegible]623623.jpg

VK_PRESENT_MODE_FIFO_KHR



Sursa: https://www.intel.com/content/dam/develop/external/us/en/images/api-vulkan-par[illegible]623623.jpg

Imagini și ImageViews

- SwapChain are un set de un set de imagini asociate VkImage
- Pentru a putea desena în aceste imagini avem nevoie de acces la aceste imagini
- Imaginile nu se pot folosi direct pentru că sunt imagini brute
- ImageView este o interfață care descrie cum să citești/desenezi/prezinți o imagine

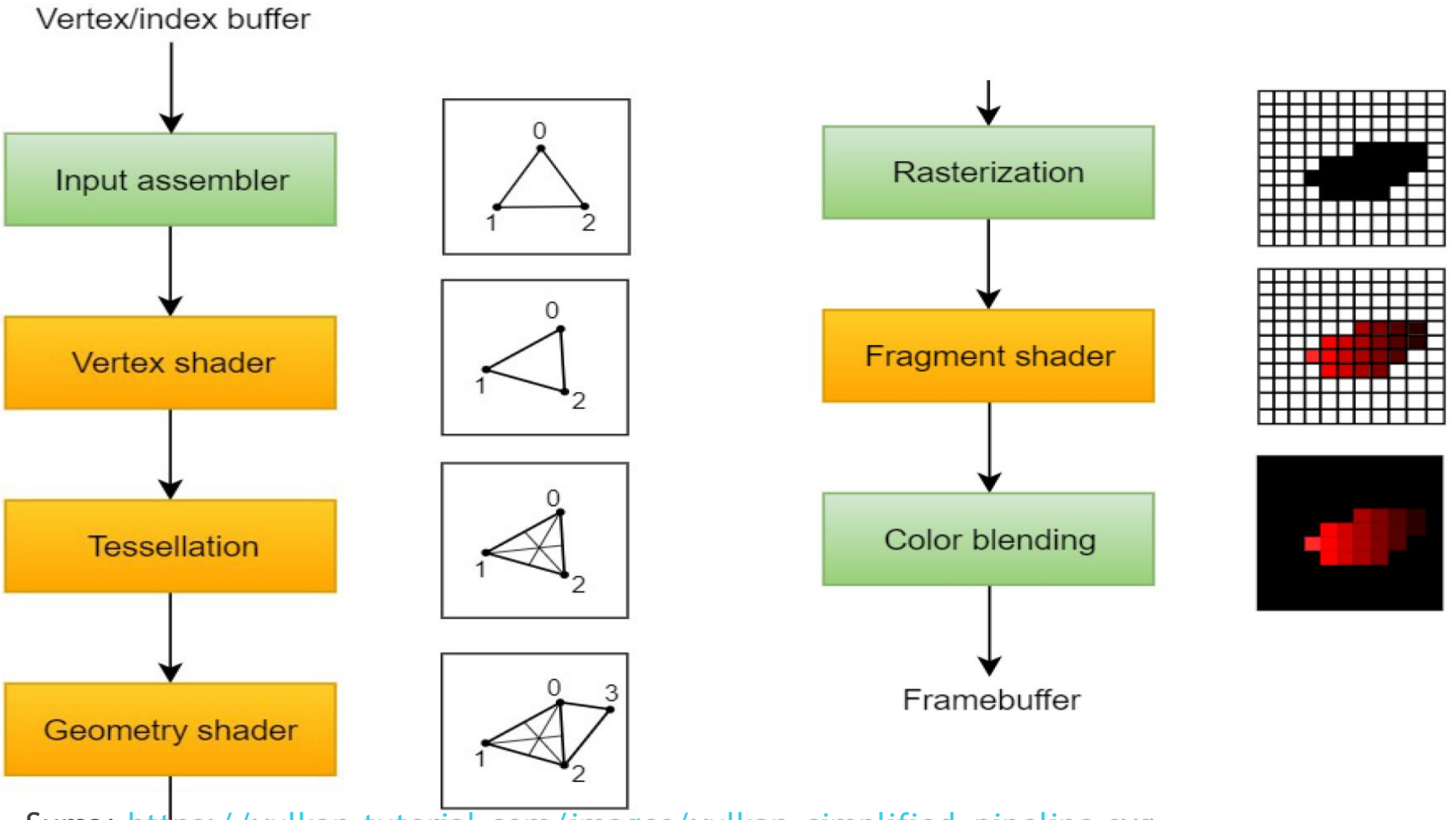
Crearea de ImageView

```
std::vector<VkImage> swapChainImages;
std::vector<VkImageView> swapChainImageViews;
for (size_t i = 0; i < swapChainImages.size(); i++) {
    VkImageViewCreateInfo createInfo{};
    createInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
    createInfo.image = swapChainImages[i];
    createInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
    if (vkCreateImageView(device, &createInfo, nullptr,
&swapChainImageViews[i]) != VK_SUCCESS) {
        throw std::runtime_error("failed to create image views!");}
}
```

Vulkan pipeline

- Pipeline-ul este în mare la fel ca în OpenGL
- Fiecare etapă se configurează individual
- Fiecare pipeline se conectează la un “Render pass” care randează în framebuffer

Pipeline grafic Vulkan



Sursa: https://vulkan-tutorial.com/images/vulkan_simplified_pipeline.svg

Crearea unui pipeline grafic

- **Vertex Input**: definește aspectul și formatul datelor de intrare vertex
- **Input assembly**: definește cum se asamblează vârfurile primitivelor (triunghiuri, linii, puncte)
- **Viewport & scissor**: maparea pe suprafața imaginii și se decuparea

- **Dynamic states**: conducta e statică, dar sunt setări care se pot modifica în timpul rulării și pot fi specificate la creare

- **Rasterizer**: transformarea primitivelor în fragmente
- **Multisampling**: tip de antialiasing folosind pentru eliminarea a aliasingului
- **Blending**: amestecarea fragmentelor
- **Depth stencil**: adâncimea + tăierea și scrierea șablonului

Multisampling

Without multisampling

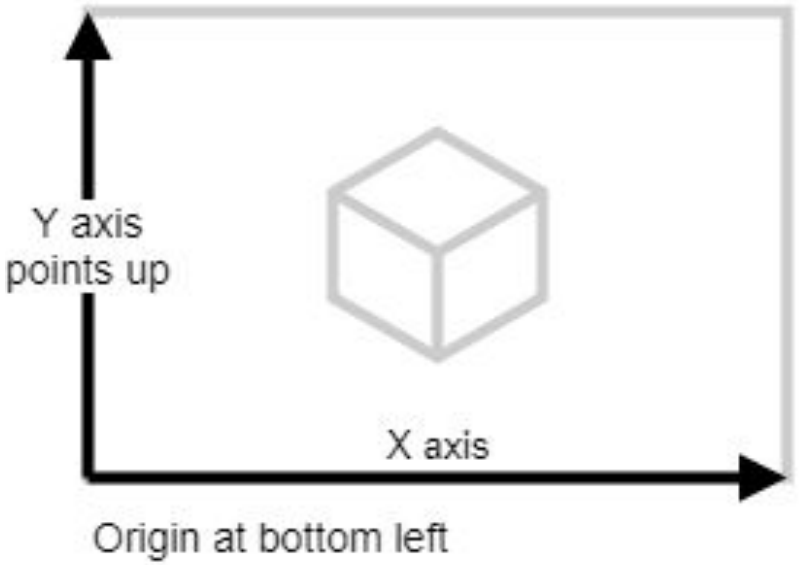
With multisampling (MSAAx8)

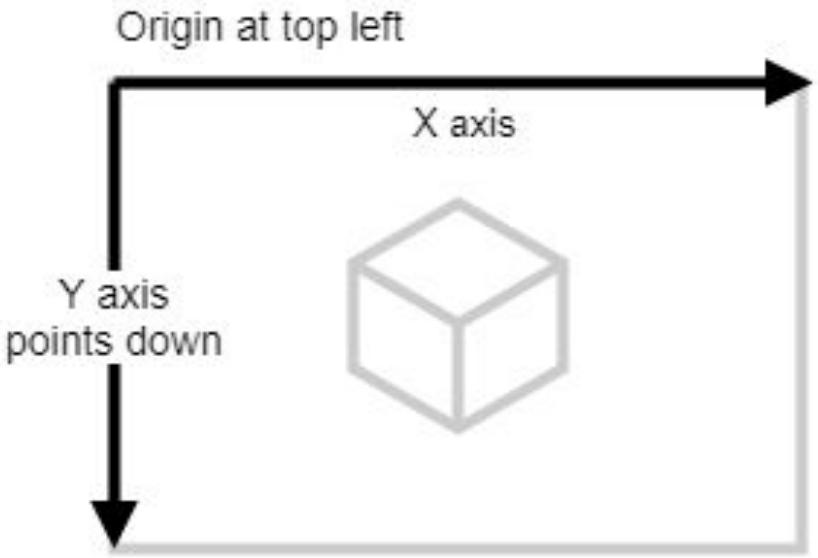
Sursa: https://vulkan-tutorial.com/images/multisampling_comparison2.png

Shadere și SPIR-V

- În API-urile grafice anterioare shaderele se încarcă în limbaj înalt
  - GLSL (OpenGL Shading Language) este folosit în OpenGL
  - HLSL (High Level Shading Language) este folosit în DirectX
- În Vulkan codul shader este precompilat în byte-code SPIR-V
- Byte cod = GLSL compilat cu glslangValidator.exe sau cu glslc.exe
- Codul SPIR-V se execută pe GPU
  - Vulkan - shader grafic sau shader calcul
  - OpenCL - shader calcul

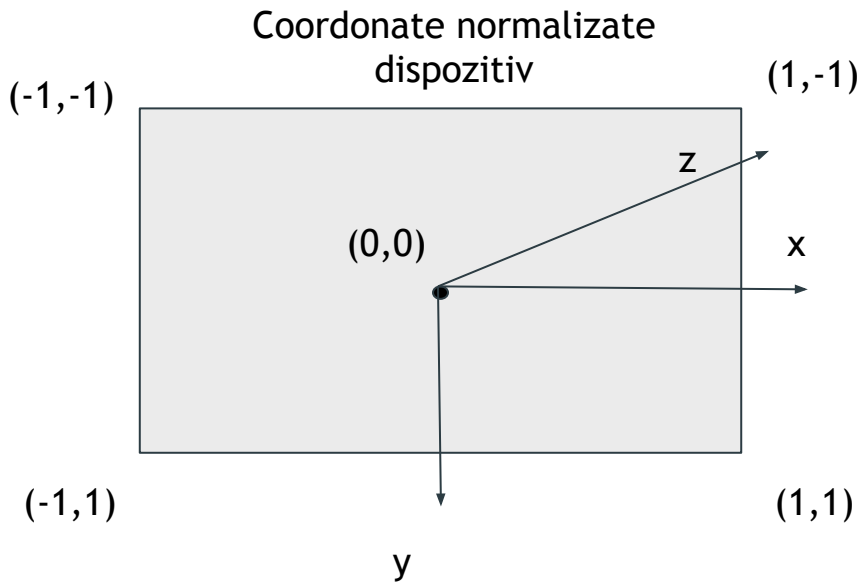
Diferența dintre sistemele de coordonate

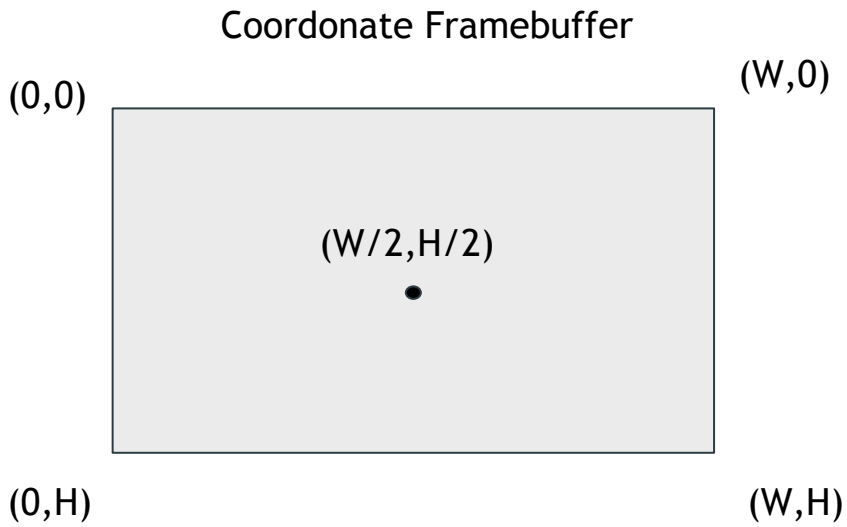




Sursa:
https://www.saschawillems.de/blog/2019/03/29/flipping-the-vulkan-viewport/

Sistemele de coordonate Vulkan





Detalii: https://matthewwellings.com/blog/the-new-vulkan-coordinate-system/

Vertex shader

```
#version 450

layout(location = 0) out vec3 fragColor;

vec2 positions[3] = vec2[](
    vec2(0.0, -0.5),
    vec2(0.5, 0.5),
    vec2(-0.5, 0.5));

vec3 colors[3] = vec3[](
    vec3(1.0, 0.0, 0.0),
    vec3(0.0, 1.0, 0.0),
    vec3(0.0, 0.0, 1.0));
```

```
void main() {
    gl_Position =

vec4(positions[gl_VertexIndex], 0.0, 1.0);
    fragColor = colors[gl_VertexIndex];
}
```

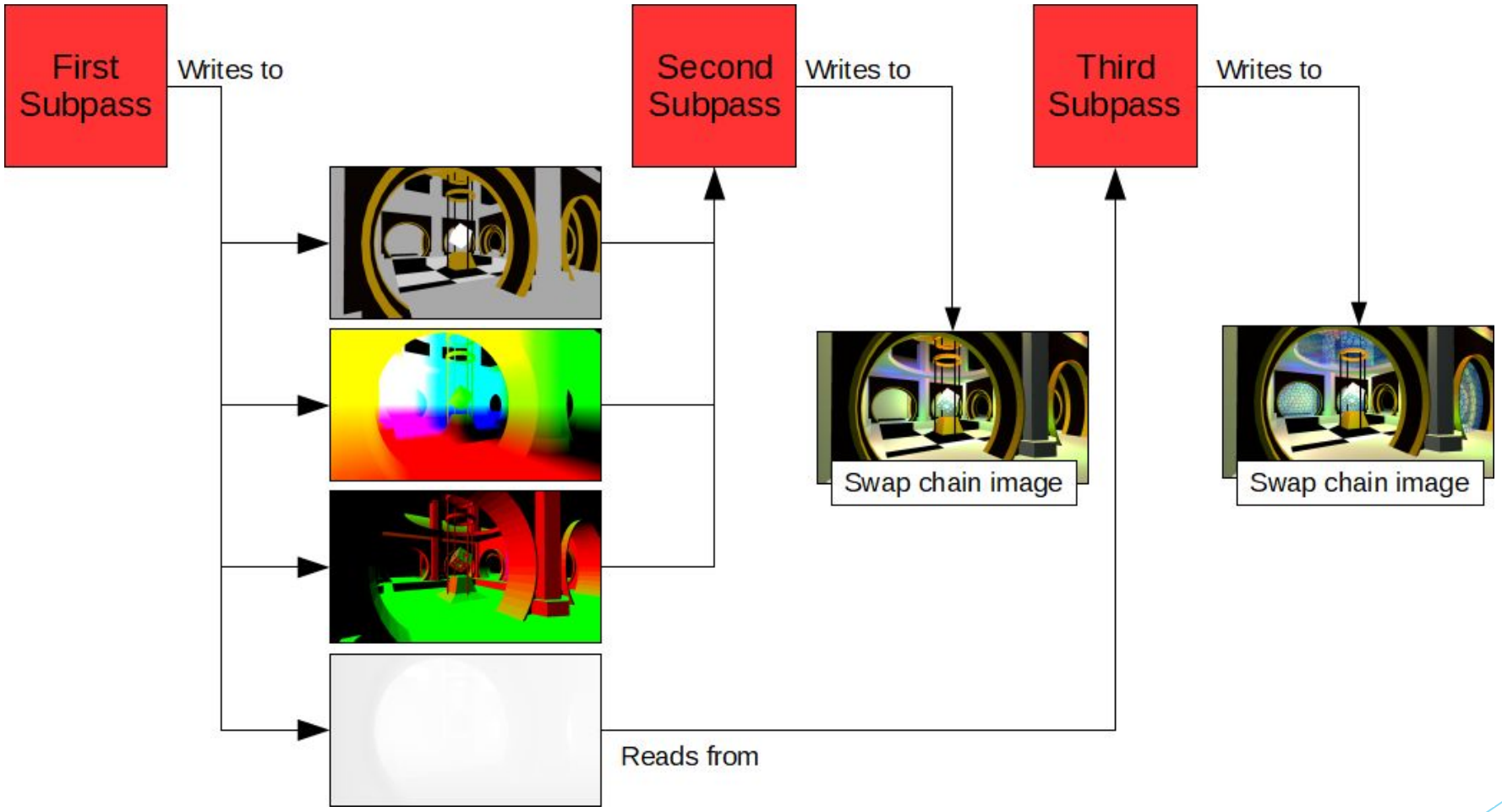
Fragment shader

```
#version 450

layout(location = 0) in vec3 fragColor;
layout(location = 0) out vec4 outColor;

void main() {
    outColor = vec4(fragColor, 1.0);
}
```

Render pass



Sursa: https://www.saschawillems.de/blog/2018/07/19/vulkan-input-attachments-and-sub-passes/

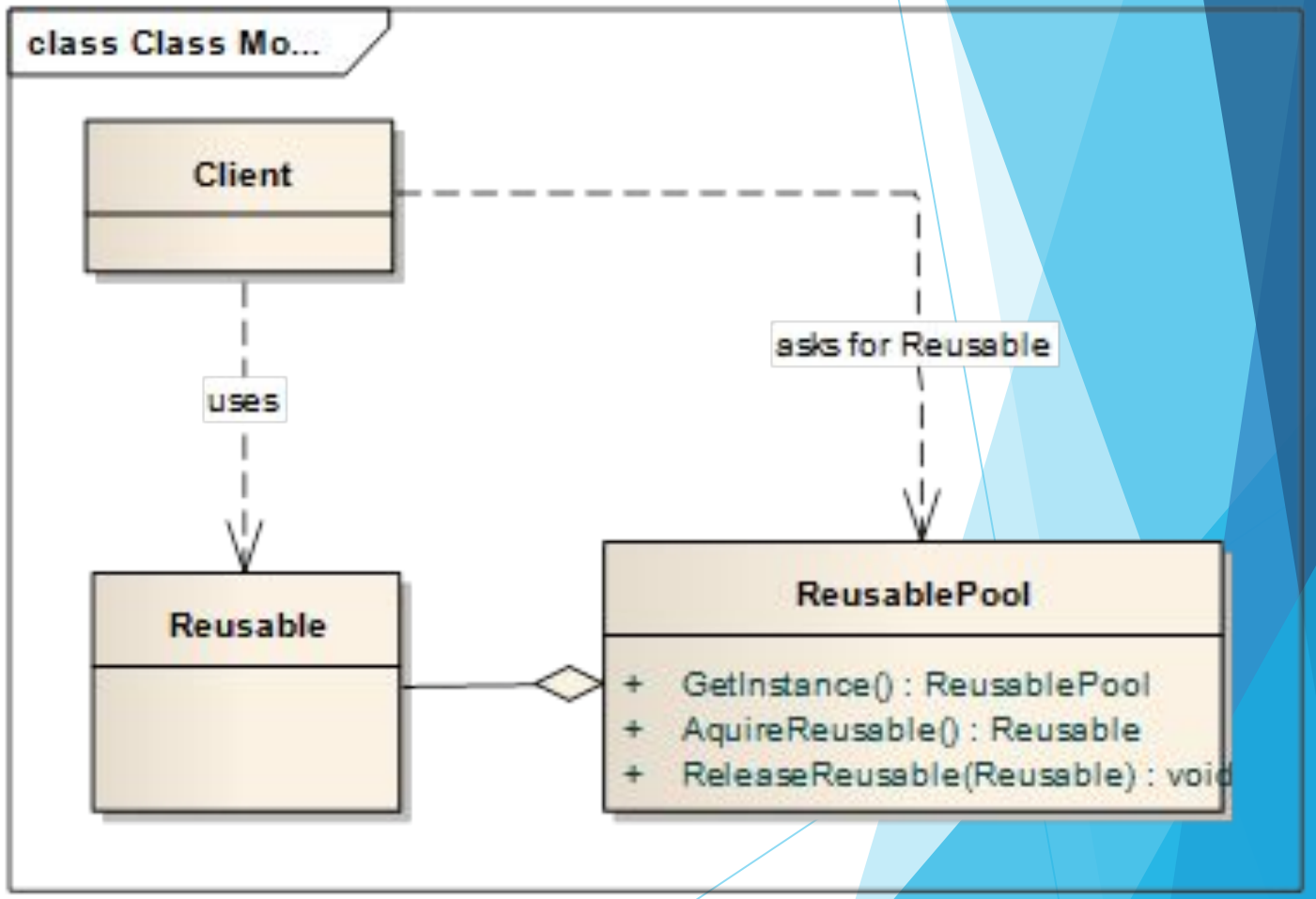
Crearea Framebuffers

- Un framebuffer poate avea atașată una sau mai multe imagini
- Un RenderPass scrie fragmentele rezultate dintr-un Pipeline în imaginile atașate unui Framebuffer
- Pentru fiecare imagine din SwapChain se crează câte un Framebuffer

```
swapChainFramebuffers.resize(swapChainImageViews.size());
for (size_t i = 0; i < swapChainImageViews.size(); i++) {
    VkImageView attachments[] = {
        swapChainImageViews[i]
    };
    VkFramebufferCreateInfo framebufferInfo{};
    framebufferInfo.sType = VK_STRUCTURE_TYPE_FRAMEBUFFER_CREATE_INFO;
    framebufferInfo.renderPass = renderPass;
    framebufferInfo.attachmentCount = 1;
    framebufferInfo.pAttachments = attachments;
    framebufferInfo.width = swapChainExtent.width;
    framebufferInfo.height = swapChainExtent.height;
    framebufferInfo.layers = 1;
    if (vkCreateFramebuffer(device, &framebufferInfo, nullptr,
&swapChainFramebuffers[i]) != VK_SUCCESS) {
        throw std::runtime_error("failed to create framebuffer!");   } }
```

Command pool

- Un command pool creează un spațiu de memorie pentru comenzi
- Aceste blocuri de memorie au aceeași dimensiune fără a avea detalii
- Aceste blocuri de memorie se folosesc apoi pentru Command Buffer
- Object Pool design pattern -> optimizarea utilizării memoriei
- Gestionează memoria folosită pentru buffere de comenzi



Structura unui Command buffer

- Forma unei comenzi:
  a. Start un Render Pass
  b. Bind la un Pipeline
  c. Bind Vertex/Index bufere
  d. Desenare

vkBeginCommandBuffer
  vkCmdBeginRenderPass
    vkCmdBindPipeline
    vkCmdBindVertexBuffers
    vkCmdBindIndexBuffer
    vkCmdDrawIndexed
  vkCmdEndRenderPass
vkEndCommandBuffer

Demo

Desenarea primului triunghi în Vulkan

Demo

Desenarea unei clepsidre în Vulkan

Vertex buffer

Index buffer

Temă

Bufer variabile uniforme în Vulkan

Să se modifice codul astfel încât dreptunghiul să se miște pe traiectoria unui cerc

Ce ați învățat astăzi?

- Să creați aplicații grafice 3D folosind Vulkan API și C++
- Să configurați Vulkan pentru a lucra pe un GPU
- Să creați elemente esențiale Vulkan, cum ar fi
  - Swapchain
  - Pipeline
  - Buffere de comenzi
  - Buffere de variabile uniforme

Resurse Vulkan

1. Specificații Vulkan
   a. Vulkan 1.0 Quick Reference
   b. Vulkan 1.1 Quick Reference
   c. Vulkan Specification
   d. Khronos Vulkan Registry
2. Vulkan SDK
   a. LunarG Vulkan SDK
3. Vulkan Cookbook
4. Articole
   a. Vulkan key resources
   b. Vulkan in 30 minutes
5. Tutoriale
   a. Vulkan tutorials
   b. API without Secrets: Introduction to Vulkan
6. Github
   a. Modern 3D computer graphics with Vulkan in C++ repository

Feedback

http://bit.ly/peakit004-feedback

Completați acum

Durează 2-3 minute

Feedback anonim - pentru formator și AgileHub