SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Tin Vujasinović

Vulkan

SEMINARSKI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Tin Vujasinović

Matični broj: 0016115323

Studij: Informacijsko i programsko inženjerstvo

Vulkan

SEMINARSKI RAD

Mentor:

Doc. dr. sc. Ivan Hip

Varaždin, siječanj 2021.

Sadržaj

1. Uvod 1
2. Razvojno okruženje 2
3. Mogućnosti Vulkan API-a 8
3.1. Postavljanje pravilnog programa 8
3.1.1. Funkcija createInstance 9
3.1.2. Funkcija setupDebugMessenger 9
3.1.3. Funkcija createSurface 10
3.1.4. Funkcija pickPhysicalDevice 10
3.1.5. Funkcija createLogicalDevice 10
3.1.6. Funkcija createSwapChain 10
3.1.7. Funkcija createImageViews 10
3.1.8. Funkcija createRenderPass 11
3.1.9. Funkcija createGraphicsPipeline 11
3.1.10. Funkcija createFramebuffers 11
3.1.11. Funkcija createCommandPool 11
3.1.12. Funkcija createVertexBuffer 11
3.1.13. Funkcija createIndexBuffer 11
3.1.14. Funkcija createCommandBuffers 12
3.2. Kreiranje i kompajliranje shader-a 12
4. Primjeri 14
5. Zaključak 15
6. Literatura 15

1. Uvod

Kao i drugi prethodni API-i, Vulkan je razvijen za upravljanje kao apstrakcija koja je dostupna na više platformi za upravljanje grafičkim procesnim jedinicama (eng. GPU). Kako hardware svakodnevno napreduje i razvija se, tako se razvijaju i grafičke procesne jedinice. Proizvođači su u novijem modernom hardware-u počeli razvojnim programerima otvarati sve više i više programibilnih mogućnosti. Stariji API-i teško su pratili ovaj razvoj i promjene kojima su iskoristili neke od novih mogućnosti bile su jedva primjetne. Neke od tih novih promjena dovele su do niza nekonzistentnosti između proizvođača, kao na primjer, sintaksa koda koji prihvaćaju shaderi. Vulkan se javlja kao moderne rješenje koje rješava veliku većinu ovih problema. Razvijen je iz nule za modernu arhitekturu grafičkih procesnih jedinica. Njegova prednost je što jako puno mogućnosti ostavlja razvojnim programerima, a za puno manje stvari se brinu driveri nego što je to bilo kod uporabe ranijih API-a. U ovome seminaru prikazati će se koji su sve koraci potrebni kako bi se započeo rad s Vulkan API-em, koje se razvojno okruženje može koristiti te kako ga treba podesiti. Na kraju, biti će prikazani neki primjeri koji su ekvivalentni primjerima koji su obrađeni u sklopu vježbe 6 iz kolegija Računalna grafika izvedenog na Fakultetu organizacije i informatike u akademskoj godini 2020./2021.

Vulkan API ima određeni niz akcija kojima se postavlja pravilno napisan program, koji se sastoje od sljedećih koraka:

- Postavljanje instance Vulkan API-a i odabir fizičkog uređaja
- Kreiranje logičkog uređaja i porodice redova za upravljanje
- Kreiranje prozora za prikaz i lanca za izmjenu slika
- Postavljanje prikaza slika i buffera za okvire
- Prolazi renderiranja
- Konfiguriranje grafičkog cjevovoda
- Postavljanje buffera za komande
- Postavljanje glavne petlje

Ovi koraci definiraju pravilno konfigurirani Vulkan API program, iako u primjerima koji slijede neće se koristiti neke od ovih funkcionalnosti, kao npr. prikaze slika jer neće se postavljati teksture na objekte iz slika. Ovo za početnike izgleda kao puno koraka, ali svaki od njih važan je kako bi razvojni programeri dobili maksimum iz grafičkih procesnih jedinica za koje razvijaju.

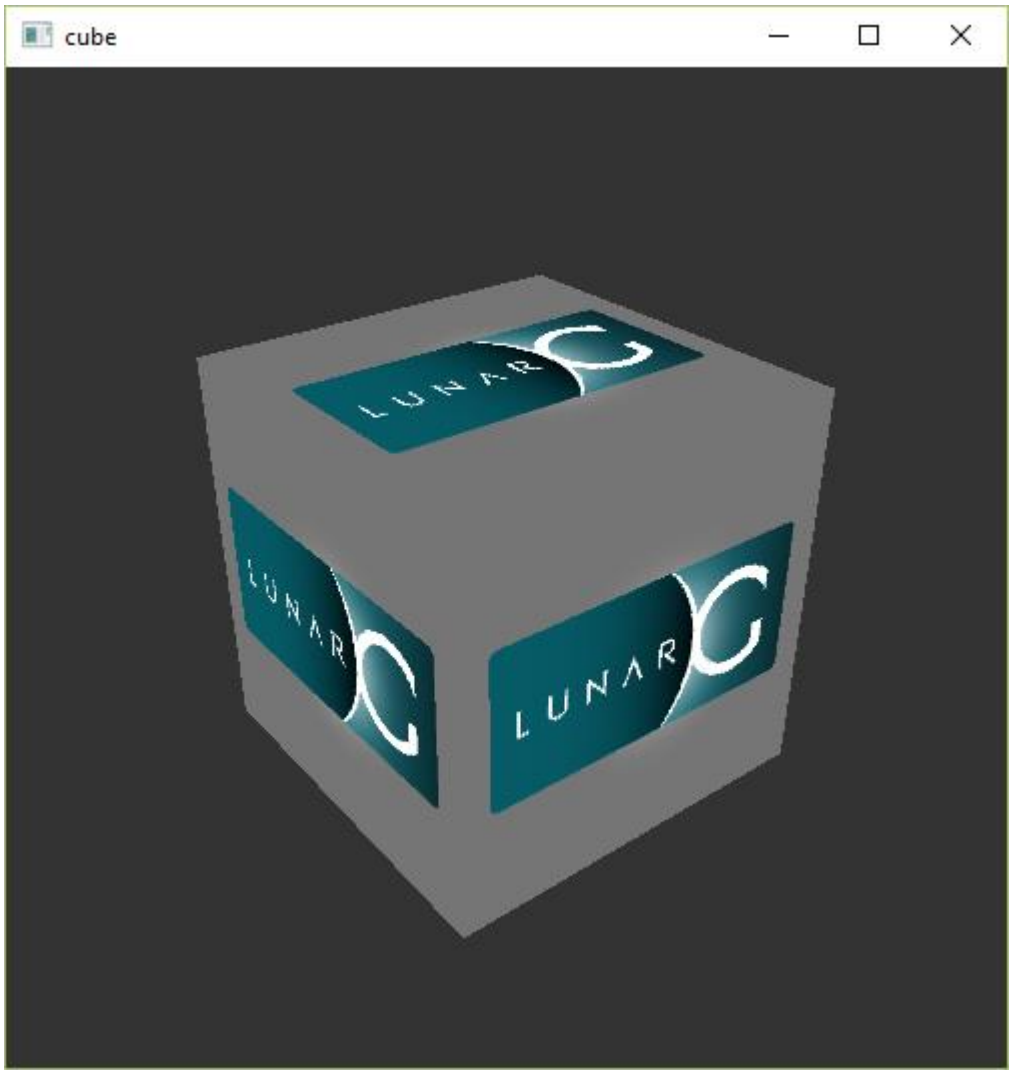
2. Razvojno okruženje

U ovome poglavlju prikazati će se kako postaviti razvojno okruženje za razvoj pomoću Vulkan API na operacijskog sustavu Windows. Kao integrirano razvojno okruženje koristiti će se Visual Studio 2019 uz pomoćni program za uređivanje shader datoteka Notepad++. Potreba za korištenjem Notepad++ programa javlja se zato jer shader programi nemaju svoj definirani nastavak i mogu se spremiti kako ih god korisnik poželi. Ovo je zanimljivo, ali razlog što naziv i nastavak shader programa nije bitan je taj što se shader programi prevode u SPIR-V bytecode koji je standardan za Vulkan API shader. To je primjer funkcionalnosti gdje Vulkan smanjuje razlike između proizvođača i postavlja svojevrsni standard na tržište. Nadalje, za razvoj za Vulkan API potrebno je preuzeti Vulkan SDK koji se može preuzeti na sljedećoj poveznici https://vulkan.lunarg.com/.



Slika 1. Preuzimanje SDK

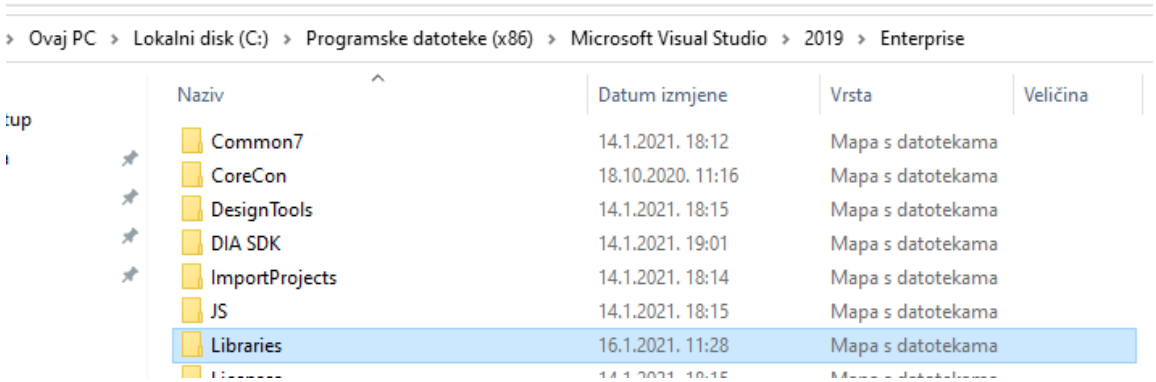
Potrebno je odabrati ikonu Windows operacijskog sustava i preuzeti i instalirati SDK na vlastito računalo. Kako bi provjerili da hardware podržava Vulkan API, možemo u mapi gdje je instalirani SDK otvoriti **Bin** mapu i u njoj pokrenuti **vkcube.exe** program. Ukoliko hardware podržava Vulkan API na ekranu će se prikazati sljedeća slika.



Slika 2. Testiranje Vulkan API-a

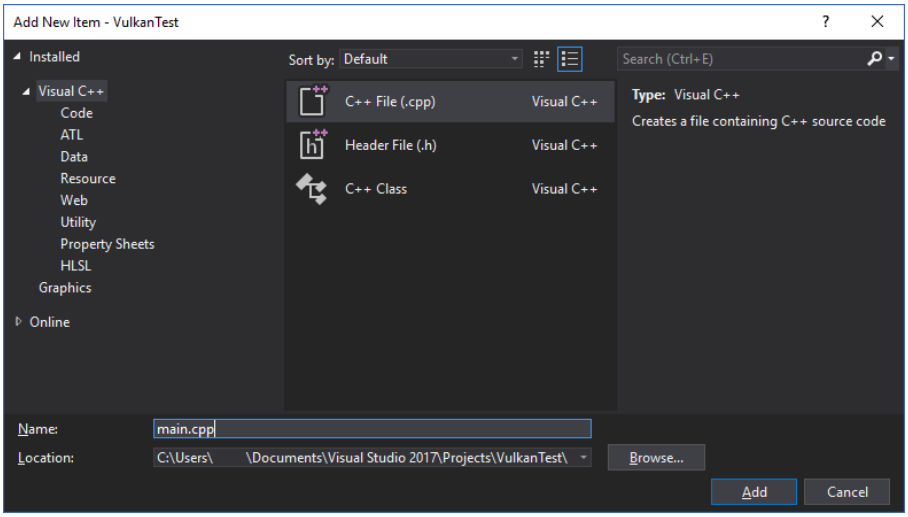
Ukoliko se pojavio ovakav prozor, može se nastaviti na daljnje korake. Kako je Vulkan API razvijen da bude neovisan o uređaju na kojem će se koristiti, nema vlastite metode za prikaz grafike koja se razvija pomoću njega. Iz tog razloga potrebno je preuzeti **GLFW** biblioteku koja može otvoriti prozor u kojoj će se prikazivati grafika. Ova biblioteka može se preuzeti na poveznici https://www.glfw.org/. Kako bi mogli koristiti biblioteku,

potrebno ju je smjestiti u mapu koja će nam biti lako dostupna. Preporuča se kreiranje mape *Libraries* unutar mape gdje je instaliran Visual Studio.



Slika 3. *Libraries* mapa

Vulkan također ne sadrži biblioteku za operacije linearne algebre pa je iz tog razloga potrebno preuzeti **GLM** biblioteku sa sljedeće poveznice https://github.com/g-truc/glm i također mapu smjestiti u mapu *Libraries.* Sada kada su sve potrebne datoteke za rad preuzete, potrebno je postaviti samo razvojno okruženje da uključi kod razvoje te datoteke. Potrebno je otvoriti Visual Studio i odabrati kreiranje novog projekta i odabrati *Windows Desktop Wizard*. Kod odabira vrste aplikacije, odabrati *Console Application*. Kako bi se mogao pisati kod, potrebno je desnim klikom kliknuti na projekt i dodati novu datoteku, a kao tip odabrati *C++ file* koji je potrebno nazvati main.cpp.



Slika 4. Kreiranje main.cpp

U novokreiranu datoteku potrebno je dodati sljedeći kod koji će javljati greške što je zasad uredu jer još uvijek Visual Studio nije podešen da uključuje prethodno preuzete datoteke. Kod koji je potrebno upisati je:

```
#define GLFW_INCLUDE_VULKAN
#include <GLFW/glfw3.h>

#define GLM_FORCE_RADIANS
#define GLM_FORCE_DEPTH_ZERO_TO_ONE
#include <glm/vec4.hpp>
#include <glm/mat4x4.hpp>

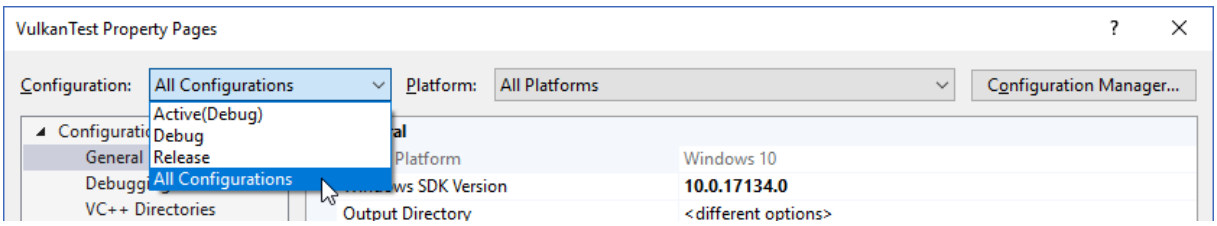
#include <iostream>


int main() {
    glfwInit();
    glfwWindowHint(GLFW_CLIENT_API, GLFW_NO_API);
    GLFWwindow* window = glfwCreateWindow(800, 600, "Vulkan window",
nullptr, nullptr);
    uint32_t extensionCount = 0;
    vkEnumerateInstanceExtensionProperties(nullptr, &extensionCount,
nullptr);
    std::cout << extensionCount << " extensions supported\n";

    glm::mat4 matrix;
    glm::vec4 vec;
    auto test = matrix * vec;

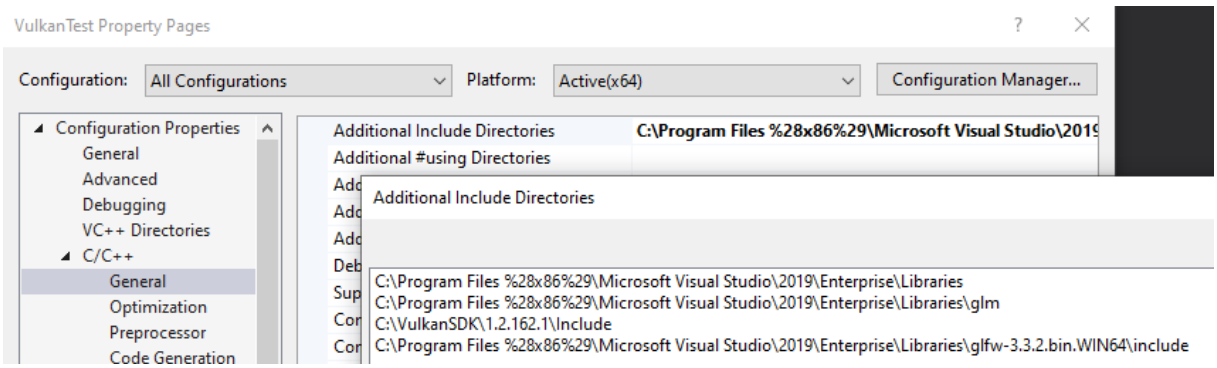

    while(!glfwWindowShouldClose(window)) {
        glfwPollEvents();
    }
    glfwDestroyWindow(window);
    glfwTerminate();
    return 0;
}
```

Kako bi se riješile greške koje vraća ovaj kod, potrebno je najprije otvoriti postavke projekta i pod *Configuration* odabrati *All configurations.*



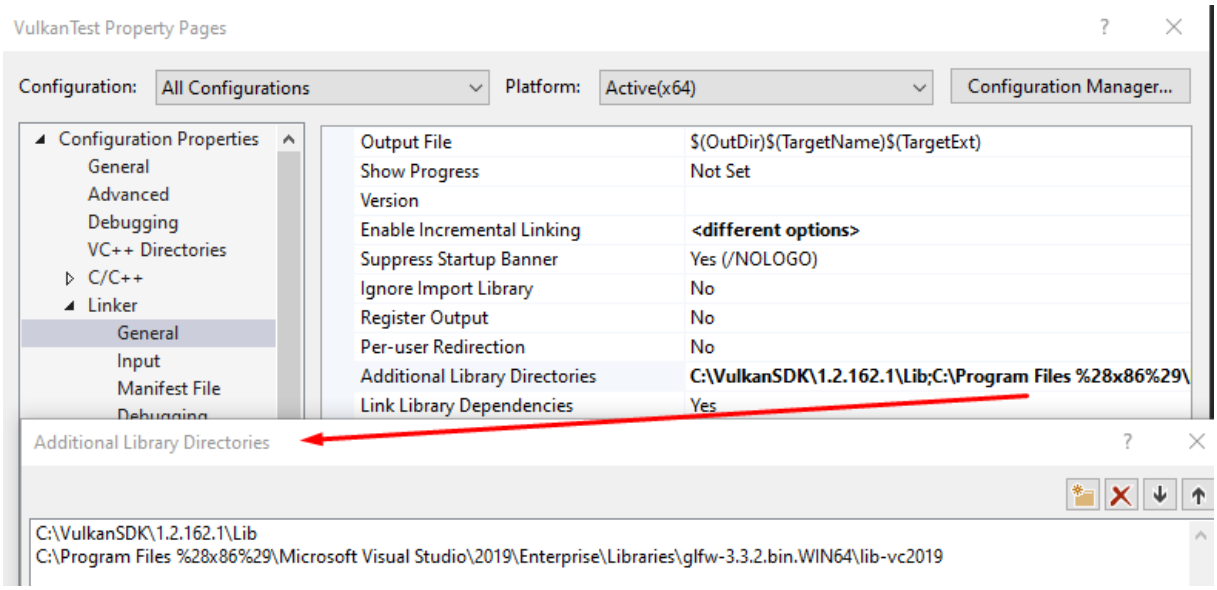
Slika 5. Postavljanje Visual Studio razvojnog okruženja

Nakon što je odabrano da ove postavke utječu na sve konfiguracije, potrebno je postaviti putanje do prethodno preuzetih biblioteka **GLFW** i **GLM**. Na sljedećoj slici prikazane su putanje do tih biblioteka, dodane pod stavkom **C/C++**, **General** te odabirom polja **Additional Include Directories.**



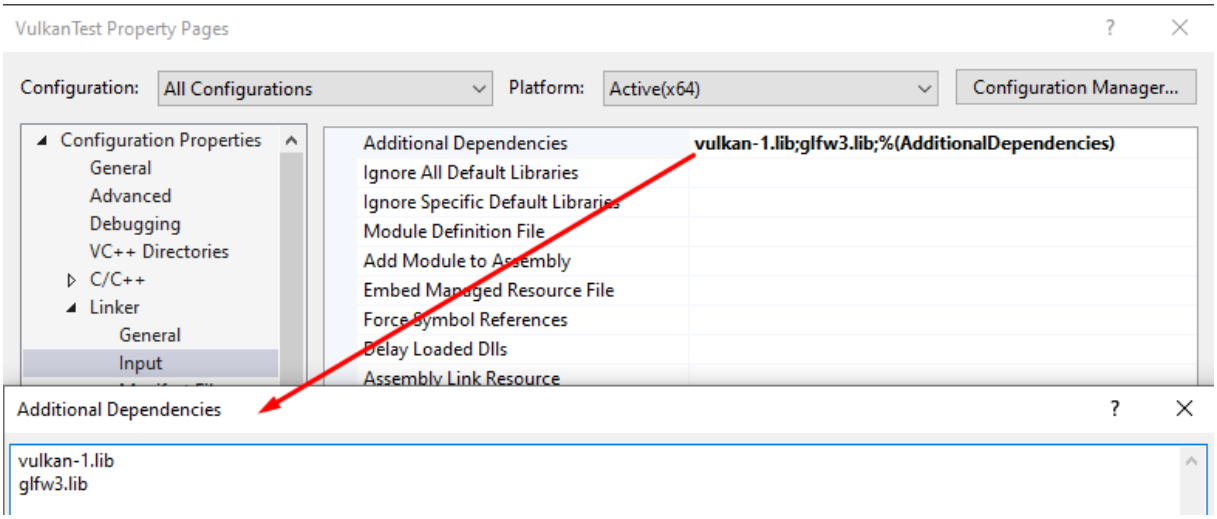
Slika 6. Additional Include Directories

Sličan postupak potrebno je provesti i pod stavkom **Linker**. Na polju **Additional Library Directories** potrebno je odabrati **Edit** i dodati putanje do mapa koje su prikazane na sljedećoj slici.



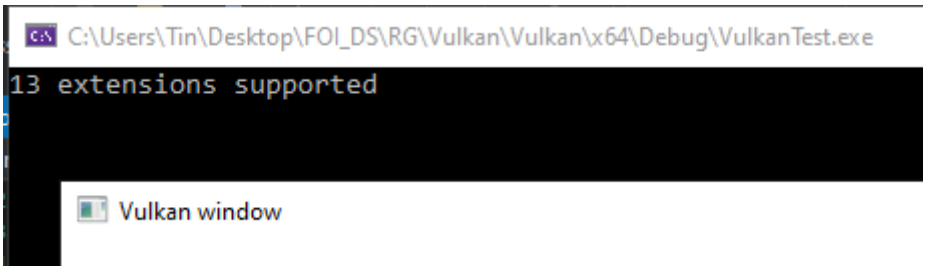
Slika 7. Additional Library Directories

Također je potrebno *.lib* datoteke dodati kao dodatne ovisnosti našeg programa bez kojeg on ne može raditi. Postupak je prikazan na slici 8.



Slika 8. Dodatne ovisnosti

Nakon što se odrade svi ovi koraci, potrebno je još odabrati C++ 17 standard kao korišteni te ukoliko su svi koraci odrađeni pravilno, program više ne bi trebao prikazivati nikakve greške te se može kompajlirati i pokrenuti. Ovaj program služi kao test te ukoliko se pokreće i prikazuje neki broj dodataka koji su dostupni, sve je spremno za razvoj uz Vulkan API.



Slika 9. Pokretanje testnog programa

Na prethodnoj slici može se vidjeti da su se otvorila dva prozora, jedan je konzola programa u kojoj se prikazuje da je dostupno 13 dodataka, a drugi prozor je **GLFW** prozor koji služi za prikaz grafike.

3. Mogućnosti Vulkan API-a

Kada su sve postavke pravilno posložene, može se krenuti s razvojem grafike pomoću Vulkan API-a. Za razvoj se koristi programski jezik C++. U programima napisanim u C++ jeziku uglavnom postoji funkcija *main* koja služi kao glavna funkcija za pokretanje programa. Na početku programa treba naredbom *include* uključiti biblioteku **vulkan.h** te se tada mogu početi pozivati Vulkan API funkcije. Za kreiranje Vulkan API objekata koriste se funkcije oblika *vkCreateXYZ* ili za alokaciju *vkAllocateXYZ* gdje *XYZ* predstavljaju naziv objekta koji se kreira. Kod razvoja važno je obratiti pozornost na upravljanje memorijom pa je tako potrebno sve što je u nekom trenutku kreiralo ili alociralo, uništiti ili osloboditi memoriju. Funkcije koje se pozivaju za ove dvije funkcionalnosti su oblika *vkDestroyXYZ te vkFreeXYZ.*

Postavljanje pravilnog programa

Kao što je ranije spomenuto, za kreiranje pravilno strukturiranog Vulkan API programa potrebno je provesti određene korake. Ti koraci mogu se izolirati u posebne funkcije kako bi se točno prikazao redoslijed. Primjer takvog redoslijeda prikazan je sljedećim kodom:

```
void initVulkan() {
    createInstance();
    setupDebugMessenger();
```

```
        createSurface();
        pickPhysicalDevice();
        createLogicalDevice();
        createSwapChain();
        createImageViews();
        createRenderPass();
        createGraphicsPipeline();
        createFramebuffers();
        createCommandPool();
        createVertexBuffer();
        createIndexBuffer();
        createCommandBuffers();
    }
```

Sada će svaka od ovih funkcija biti ukratko opisana kako bi se moglo lakše shvatiti njena zadaća i svrha u cjelokupnom funkcioniranju programa.

3.1.1. Funkcija createInstance

Ova funkcija služi za inicijalizaciju i kreiranje instance programa te se pomoću nje zadaju informaciju o programu. Informacije koje se mogu dodati su npr. ime aplikacije, verzija aplikacije, verzija API-a i slično. Instanca koja se kreira pomoću ove funkcije je veza između Vulkan API-a i aplikacije korisnika te ujedno prosljeđuje neke informacije upravljačkim programima grafičke procesne jedinice.

3.1.2. Funkcija setupDebugMessenger

Funkcija *setupDebugMessenger* koristi se kako bi vratila greške koje ispisuju slojevi validacije koje koristi Vulkan API. Ovaj API razvijen je tako da vrlo malo toga označava kao grešku sam po sebi ali je moguće uključiti dodatne slojeve validacije koji će provjeravati određene stvari. Moguće je dodati slojeve za:

- Provjeru parametara funkcija i usporedbu sa standardnim unosima
- Praćenje kreiranje i brisanja objekata za izbjegavanje curenja memorije
- Logiranje poziva i parametara funkcija itd.

3.1.3. Funkcija createSurface

Kako bi se grafika koja se može razviti Vulkan API-em prikazala na zaslonu računala, potrebno je kreirati površinu za prikaz. Tu se poziva prethodno spomenuta biblioteka **GLFW** kojoj se prosljeđuju informacije o aplikaciji.

3.1.4. Funkcija pickPhysicalDevice

Funkcija *pickPhysicalDevice* služi za odabir fizičkog grafičkog procesora koji će se koristiti za prikaz grafike. U većini računala postoji više grafičkih procesnih jedinica, a ovom funkcijom dohvaća se lista svih njih, provjerava se koji podržavaju Vulkan API te se testira i svakoj se dodjeljuje određeni rezultat. Prema tom rezultatu odabire se najbolja.

3.1.5. Funkcija createLogicalDevice

Pomoću ove funkcije za odabranu fizičku grafičku procesnu jedinicu kreira se logička koja će služiti kao sučelje. Ovisno o potrebama, moguće je kreirati više logičkih uređaja za istu fizičku grafičku jedinicu.

3.1.6. Funkcija createSwapChain

Vulkan API ne podržava koncept zadanog buffer-a u kojem se čuvaju i renderiraju slike koje sljedeće treba prikazati. Umjesto toga, koristi se *swapChain*, koji je zapravo red slika koje su renderirane i čekaju da ih se prikaže na ekranu.

3.1.7. Funkcija createImageViews

Ova funkcija kreira pogleda prema slikama koje su učitane u program, opisuje kako se slikama pristupa, koje dijelove slike se koristi te na koji način.

3.1.8. Funkcija createRenderPass

Funkcija *createRenderPass* poziva se za kreiranje prolaza prikaza koji se mogu sastojati od više pod-prolaza koji su slijedne operacije renderiranja koje ovise o stanjima buffera. Grupiranjem pod-prolaza u prolaze moguće je uštedjeti memoriju i ubrzati rad.

3.1.9. Funkcija createGraphicsPipeline

Pomoću ove funkcije u program se učitavaju unaprijed kompajlirani shaderi. Unutar ove funkcije poziva se pomoćna funkcija koja dohvaća i postavlja sve potrebne informacije o shaderima.

3.1.10. Funkcija createFramebuffers

Kako bi se slike koje su renderirane i čekaju na prikaz u *swapChain*-u mogle prikazati na ekranu, mora se prva slika koja je spremna za prikaze učitati u *frameBuffer*. Ova funkcija kreira i postavlja *frameBuffer*.

3.1.11. Funkcija createCommandPool

Naredbe se kod razvoja uz pomoć Vulkan API-a ne pozivaju direktno već se moraju naredbe učitati u *commandBuffer*. Ova funkcija kreira *commandPool* koji upravlja memorijom koju koriste *commandBuffer*-i i koju oni alociraju.

3.1.12. Funkcija createVertexBuffer

Kako bi se grafika prikazala potrebno je unijeti vrhove u program koje želimo iscrtati. Kako bi programer mogao prenijeti te vrhove iz programa u shader, da oni ne moraju biti upisani direktno u shaderu, koristi se *vertexBuffer*.

3.1.13. Funkcija createIndexBuffer

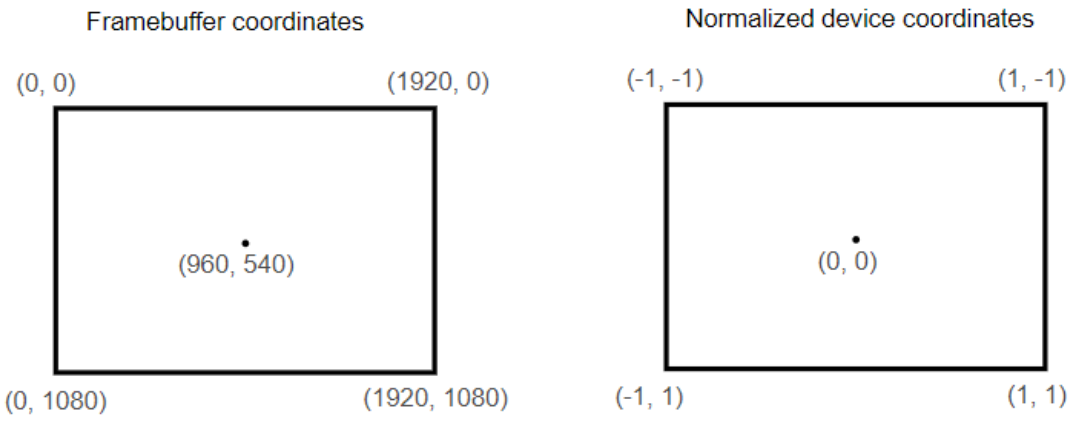
Funkcija *createIndexBuffer* služi sličnoj svrsi kao i funkcija *createVertexBuffer*, razlika je u tome što se u *indexBuffer*-u prenosi redoslijed indeksa vrhova prema kojem će se iscrtavati.

3.1.14. Funkcija createCommandBuffers

U ovoj funkciji kreiraju se i alocira se memorija za *commandBuffer*-e. Bira se *commandPool* te broj *commandBuffer*-a koji se trebaju alocirati.

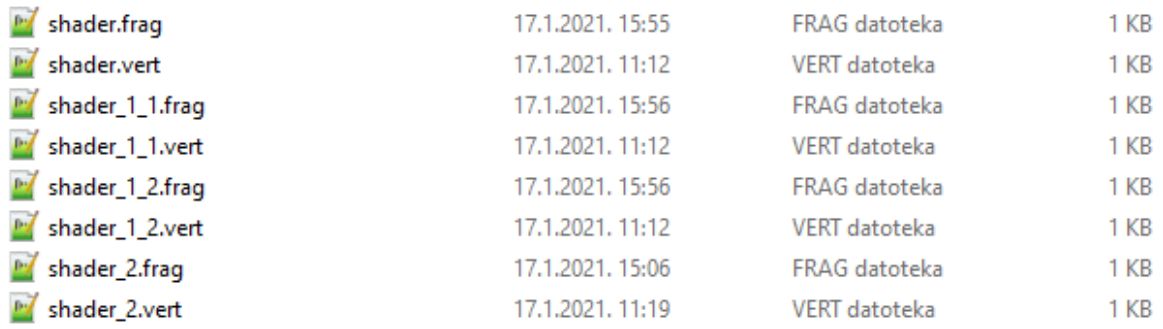
Kreiranje i kompajliranje shader-a

Za razliku od prethodnika, Vulkan API zahtjeva da shaderi budu prevedeni u posebni oblik *bytecode*-a koji on može razumjeti. Ovaj format *bytecode*-a nazvan je **SPIR-V** te je on dizajniran za korištenje s OpenGL-om te Vulkan-om. Kod preuzimanja Vulkan SDK, ujedno je preuzet i kompajler za SPIR-V. On pretvara GLSL kod koji se koristi za programiranje shadera u *bytecode*. Važno je napomenuti da Vulkan koristi normalizirane koordinate slično kao i OpenGL, ali predznak Y koordinate je ovdje okrenut.



Slika 10. Koordinate

Kao što je ranije spomenuto, shaderi se mogu pisati u obične tekstualne datoteke s bilo kojim datotečnim nastavkom. Na slici se može vidjeti više shadera koji su postavljeni da se otvaraju u programu Notepad++.



Slika 11. Shaderi

Za kompajliranje ovi shadera koristiti će se **glslc.exe** koji je razvio Google i koristi sintaksu sličnu kao i standardni GCC kompajler. Kako bi se kompajliranje shadera olakšalo, najbolje je kreirati praznu *.bat* datoteku koja će služiti kao program za kompajliranje. Na sljedećoj slici će se prikazati naredbe u *compile.bat* datoteci koja je korištena kod izrade primjera iz poglavlja 5.

compile.bat

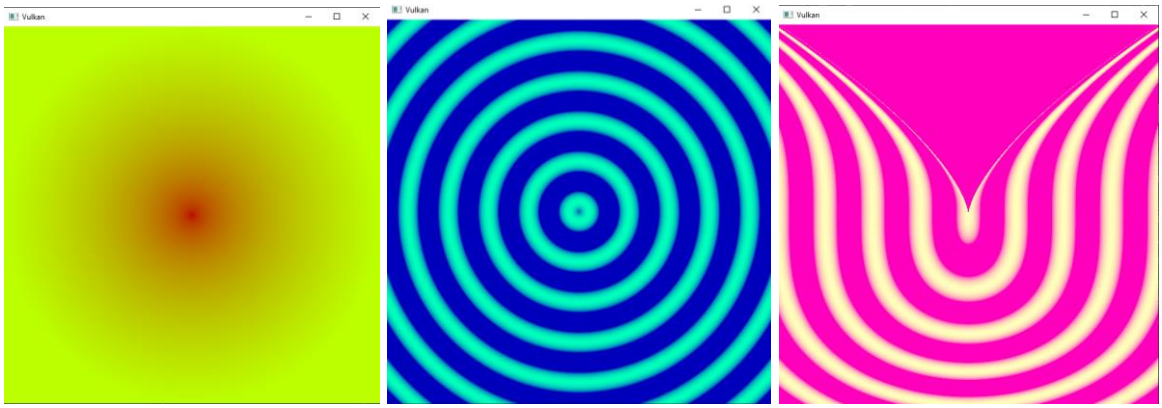
```
1  C:/VulkanSDK/1.2.162.1/Bin32/glslc.exe shader.vert -o shaders/vert.spv
2  C:/VulkanSDK/1.2.162.1/Bin32/glslc.exe shader.frag -o shaders/frag.spv
3
4  C:/VulkanSDK/1.2.162.1/Bin32/glslc.exe shader_1_1.vert -o shaders/vert_1_1.spv
5  C:/VulkanSDK/1.2.162.1/Bin32/glslc.exe shader_1_1.frag -o shaders/frag_1_1.spv
6
7  C:/VulkanSDK/1.2.162.1/Bin32/glslc.exe shader_1_2.vert -o shaders/vert_1_2.spv
8  C:/VulkanSDK/1.2.162.1/Bin32/glslc.exe shader_1_2.frag -o shaders/frag_1_2.spv
9
10 C:/VulkanSDK/1.2.162.1/Bin32/glslc.exe shader_2.vert -o shaders/vert_2.spv
11 C:/VulkanSDK/1.2.162.1/Bin32/glslc.exe shader_2.frag -o shaders/frag_2.spv
12
13 pause
```

Slika 12. *compile.bat* naredbe

Prethodne naredbe pozivaju *glslc.exe* koji se nalazi unutar mape gdje je instaliran Vulkan SDK, te nakon toga slijedi naziv datoteke u kojoj se nalazi kod shadera, tj. GLSL kod, a na kraju naziv datoteke u koju će se kompajlirani *bytecode* spremiti. Kako bi kompajlirani shaderi bili izolirani, spremaju se u mapu *shaders*.

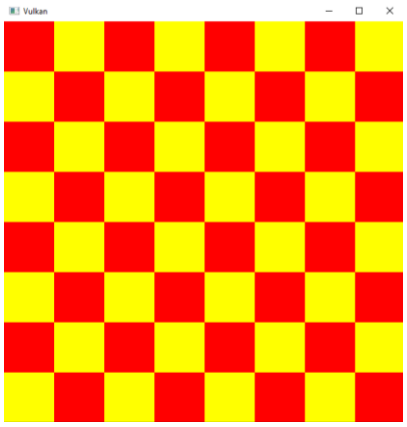
4. Primjeri

U ovom primjeru biti će prikazani primjeri koji su izrađeni uz pomoć Vulkan API-a. Izrađena su četiri primjera, od čega su prva tri primjera jednostavniji, prikazuju kako se može boja mijenjati prema udaljenosti od centra slike čime se dobivaju zanimljivi oblici, a posljednji primjer je prvi zadatak koji je bio zadan za zadaću, a to je izrada crveno žute šahovnice.



Slika 13. Primjeri 1-3

U ovim primjerima šalju se iste koordinate vrhova u fragment shader, ali se načini primjene boje na piksel razlikuju i na taj način moguće je dobiti različite oblike. Sljedeći primjer prikazuje šahovnicu sastavljenu od žutih i crvenih polja. Ovakav prikaz postignut je crtanjem crvenih kvadratića koji se svaki sastoje od dva trokuta, dok je pozadina žuta.



Slika 14. Šahovnica

5. Zaključak

Vulkan je novi API koji pruža veću razinu apstrakcije za moderne grafičke procesne jedinice. Novo sučelje omogućava programerima da bolje opišu koja je svrha njihovih programa te dovodi do boljih performansi dok smanjuje neočekivane konflikte i rezultate sa upravljačkim programima. Najveća prednost Vulkan API-a pred svojim prethodnicima je ta što je potpuno neovisan o platformi za koju se razvija, pa tako omogućava razvoj za Windows, Linux i npr. Android u isto vrijeme.

6. Literatura

[1] „Vulkan tutorial“, Dostupno: https://vulkan-tutorial.com [pristupano 16.1.2021.]

[2] „Vulkan C++ examples and demos“, Dostupno: https://github.com/SaschaWillems/Vulkan [pristupano 17.1.2021.]

[3] „Vulkan guide“, Dostupno: https://www.glfw.org/docs/latest/vulkan_guide.html [pristupano: 17.1.2021.]