Általános információk

A diplomaterv szerkezete:

1. Diplomaterv feladatkiírás
2. Címoldal
3. Tartalomjegyzék
4. A diplomatervező nyilatkozata az önálló munkáról és az elektronikus adatok kezeléséről
5. Tartalmi összefoglaló magyarul és angolul
6. Bevezetés: a feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása
7. A feladatkiírás pontosítása és részletes elemzése
8. Előzmények (irodalomkutatás, hasonló alkotások), az ezekből levonható következtetések
9. A tervezés részletes leírása, a döntési lehetőségek értékelése és a választott megoldások indoklása
10. A megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek
11. Esetleges köszönetnyilvánítások
12. Részletesés pontos irodalomjegyzék
13. Függelék(ek)

Felhasználható a következő oldaltól kezdődő **Diplomaterv sablon** dokumentum tartalma. Ügyeljen a tanszék, a hallgató, a konzulens nevét és a beadás évét jelölő szövegdobozokra, mert azokra külön ki kell adni a frissítést. A mezők tartalma a sablonban a dokumentum adatlapja alapján automatikusan kerül kitöltésre (Fájl/Információ/Tulajdonságok/Speciális tulajdonságok).

A diplomaterv szabványos méretű A4-es lapokra kerüljön. Az oldalak tükörmargóval készüljenek (mindenhol 2.5cm, baloldalon 1cm-es kötéssel). Az alapértelmezett betűkészlet a 12 pontos Times New Roman, másfeles sorközzel.

Minden oldalon - az első négy szerkezeti elem kivételével - szerepelnie kell az oldalszámnak.

A fejezeteket decimális beosztással kell ellátni. Az ábrákat a megfelelő helyre be kell illeszteni, fejezetenként decimális számmal és kifejező címmel kell ellátni. A fejezeteket decimális aláosztással számozzuk, maximálisan 3 aláosztás mélységben (pl. 2.3.4.1.). Az ábrákat, táblázatokat és képleteket célszerű fejezetenként külön számozni (pl. 2.4. ábra, 4.2 táblázat vagy képletnél (3.2)). A fejezetcímeket igazítsuk balra, a normál szövegnél viszont használjunk sorkiegyenlítést. Az ábrákat, táblázatokat és a hozzájuk tartozó címet igazítsuk középre. A cím a jelölt rész alatt helyezkedjen el.

A képeket lehetőleg rajzoló programmal készítsék el, az egyenleteket egyenlet-szerkesztő segítségével írják le.

Az irodalomjegyzék szövegközi hivatkozása történhet a Harvard-rendszerben (a szerző és az évszám megadásával) vagy sorszámozva. A teljes lista névsor szerinti sorrendben a szöveg végén szerepeljen (sorszámozott irodalmi hivatkozások esetén hivatkozási sorrendben). A szakirodalmi források címeit azonban mindig az eredeti nyelven kell megadni, esetleg zárójelben a fordítással. A listában szereplő valamennyi publikációra hivatkozni kell a szövegben. Minden publikáció a szerzők után a következő adatok szerepelnek: folyóirat cikkeknél a pontos cím, a folyóirat címe, évfolyam, szám, oldalszám tól-ig. A folyóirat címeket csak akkor rövidítsük, ha azok nagyon közismertek vagy nagyon hosszúak. Internet hivatkozások megadásakor fontos, hogy az elérési út előtt megadjuk az oldal tulajdonosát és tartalmát (mivel a link egy idő után akár elérhetetlenné is válhat), valamint az elérés időpontját.

Fontos:

- a szakdolgozat készítő/diplomatervező nyilatkozata (a jelen sablonban szereplő szövegtartalommal) kötelező előírás Karunkon, ennek hiányában a szakdolgozat/diplomaterv nem bírálható és nem védhető!
- mind a dolgozat, mind a melléklet maximálisan 15 MB méretű lehet!

Jó munkát, sikeres szakdolgozat készítést ill. diplomatervezést kívánunk!

FELADATKIÍRÁS

A feladatkiírást a **tanszék saját előírása szerint** vagy a tanszéki adminisztrációban lehet átvenni, és a tanszéki pecséttel ellátott, a tanszékvezető által aláírt lapot kell belefűzni a leadott munkába, vagy a tanszékvezető által elektronikusan jóváhagyott feladatkiírást kell a Diplomaterv Portálról letölteni és a leadott munkába belefűzni (ezen oldal HELYETT, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell megismételni a feladatkiírást.

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

Borbás András

3D VILÁG TERVEZÉSÉT TÁMOGATÓ ALKALMAZÁS KÉSZÍTÉSE

KONZULENS

Fridvalszky András

BUDAPEST, 2022

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Borbás András**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2022. 05. 17.

...…….
Borbás András

Összefoglaló

Ide jön a ½-1 oldalas magyar nyelvű összefoglaló, melynek szövege a Diplomaterv Portálra külön is feltöltésre kerül.

Abstract

Ide jön a ½-1 oldalas angol nyelvű összefoglaló, amelynek szövege a Diplomaterv Portálra külön is feltöltésre kerül.

1 Bevezetés

A motivációmmal és egy rövid feladat ismertetővel szeretném kezdeni a szakdolgozatomat.

1.1 Előzmények

Már nagyon régóta tudom, hogy játékfejlesztéssel akarok majd foglalkozni, mindig is ez volt az álmom. Ezért is jelentkeztem az Irányítás és Informatika Tanszék szoftverfejlesztés specializációjára, mert itt tudtam, hogy 3D motorokkal és játékfejlesztéssel is lehet foglalkozni.

Míg ezen specializációra jártam, sikerült belátást nyernem a játékfejlesztés rejtelmeibe és végre élesebb környezetben használnom a két legismertebb videójáték fejlesztő környezetet és motort, az Unreal Engine 4-et és a Unity-t, a témalaboratórium tárgy keretein belül. Ezeket a szoftvereket én már régebb óta ismerem és használtam őket, de mindig csak hobbiból kísérletezgettem bennük, most viszont három játékot is kellett bennük készíteni. Ez alatt csak még jobban megerősödött bennem, hogy én 3D motorokkal és játékfejlesztéssel szeretnék foglalkozni.

Így már az önálló laboratóriumomnak is ehhez kapcsolódó témát választottam, viszont mivel szerintem már egész jól ismertem a fent említett két játékmotort, és akkor érdekelt a Vulkan grafikus API és hogy hogy működnek ezek a játékmotorok ezért egy saját készítésű, Vulkan alapú játékmotort csináltam. Ez egész jól működött, viszont nehéz volt használni, csak kódban lehetett bármit is szerkeszteni.

1.2 Feladat bemutatása

Szakdolgozatomban a fent említett motoromat szeretném továbbfejleszteni úgy, hogy egy használható pályaszerkesztőt kapjunk, amiben ráadásul nagyobb pályákat is hatékonyan lehet készíteni azok feldarabolásával és utána dinamikus betöltésével. Ezt trigger volume-ok segítségével csinálnám, amik be- illetve kitöltik a hozzájuk megadott pályarészeket. A szerkesztést pedig egy egyszerű felhasználó felület segítené, ahol menteni, betölteni, és szerkeszteni lehetne a pálya és objektumok különböző tulajdonságait.

2 CatEngine

Erre a névre kereszteltem az objektum orientáltan újraírt, önálló laboratóriumon elkészített játékmotoromat.

2.1 Történet

A motort eredetileg a Vulkan Tutorial weboldal[1] alapján csináltam, viszont a vulkan.hpp c++-os binding-okkal[2], a biztonságosabb típuskezelés és modernebb használat érdekében.

Hozzáadtam az ImGui[3] könyvtárat is, aminek segítségével készítettem a kezdetleges felhasználói felületet.

Az önálló laboratórium és a szakdolgozat készítés között átírtam a motort objektum orientálttá, egy youtube tutorial[4] alapján.

2.2 Felhasznált szoftverek

Bemutatom az összes szoftvert, amiket felhasználtam a szerkesztőmotor fejlesztése során.

2.2.1 GitHub[5] és GitKraken[6]

Már nagyon régóta tudom hogy a legfontosabb dolog szoftverfejlesztés közben a gyakori mentés. Viszont nem sokra megyünk a mentésünkkel ha mondjuk korruptálódik az adat egy áramkimaradás miatt. Ezért én szinte minden projektemet eltárolom online is, a kód alapú projekteket általában GitHub-on. Így a különböző verziók is megmaradnak és ezekre bármikor vissza tudok állni, vagy összevetni, hogy mi változott.

Lokálisan pedig a GitKrakent használom, ami egy felhasználói felület a git hez, ami nagyon átláthatóan mutatja a módosításokat bármilyen szöveges fájlon vagy képen.

2.2.2 Visual Studio[7] és Resharper C++[8]

A fejlesztést még a 2019 es Visual Studio val kezdtem, de mikor megláttam, hogy a 2022 es Visual Studionak lesz beépített CMake projekt kezelése, egyből letöltöttem a előzetes verzióját. Ekkor már át volt írva a projekt Visual Studio Solution formából CMakeLists alapú projekté, mert sok különböző fejlesztői környezetet kipróbáltam, mint

például a CLion[10] vagy a Visual Studio Code[11], vagy a Visual Studio Visual Assist[9] plugin-nal, de mindig visszatértem a Visual Studio-hoz valamiért. Végül kipróbáltam a JetBrains ReSharper C++[8] plugin-ját ami hozzáadta a CLion összes jó tulajdonságát a Visual Studio-hoz, mint például a kódba ágyazott paraméter nevek és típusok, és a clang tidy ellenőrzések. Így erre esett a végleges választásom C++ forráskód szerkesztésre.

2.2.3 Visual Studio Code[11]

Webfejlesztő hátterem miatt nagyon jól ismerem és szeretem a VSCode-ot, mert gyors, testreszabható és könnyen el lehet benne érni hasznos parancsokat szövegesen. Ezért egyszerű választás volt a C++-on kívül (de néha arra is) minden más forrás és konfigurációs fájl, például a shader-ek kódjának, szerkesztésére ezt használni. Ha tehetném a C++ forráskódot is ebben szerkeszteném, de sajnos még nincs olyan funkcionalitási szinten, mint a Visual Studio a ReSharper-rel.

2.2.4 CMake[12]

Az önálló laboratóriumom projektje még Visual Studio Solution-ként volt elkészítve, de eközben is kísérleteztem más fejlesztő környezetekkel, viszont ezt nem minden támogatta. Ekkor találtam rá a CMake-re, ami egy cross platform eszköz projektek fordítására és csomagolására. Ebbe átírva a projektet, megtudtam azt nyitni bármilyen fejlesztő környezetben, bármilyen platformon, így például VSCode-ban is, és azoknak is működtek a kódolást segítő funkcióik.

2.3 Felhasznált technológiák

Bemutatom az összes könyvtárat, amiket felhasználtam a szerkesztőmotor fejlesztéséhez.

2.3.1 Vulkan[13]

A Vulkan egy sokkal hardver közelibb grafikus api, mint az OpenGL vagy a Direct3D 11, ami miatt sokkal jobb teljesítményt lehet vele elérni, mint az előbb említett, régebbi apikkal. Viszont a hardver közelisége miatt nehezebb is kezelni, nagyon sok kódot kell írni ahhoz, hogy eljussunk csak egy megjelenített háromszögig. Viszont előnye a Direct3D 12 vel szemben, hogy teljesen platform független, nem csak Windowson és

XBOX on működik, hanem Linuxon és mobiltelefonokon is, ráadásul ugyanazzal az api-val, bármilyen változtatás nélkül.

2.3.2 GLFW[15]

A GLFW egy olyan könyvtár, ami kizárólag platformfüggetlen ablakok létrehozására és különböző beviteli eszközök bemeneteinek feldolgozására képes. Viszont cserébe nagyon lightwieght, Vulkan mellé tökéletes használni mert ott úgyis nekünk kell implementálnunk mindent és így nincs semmi felesleges a programunkban.

2.3.3 GLM[16]

A GLM vagy hosszan OpenGL Mathematics egy OpenGL shaderek írását, és vektorok és mátrixok használatát segítő könyvtár. Egyszerű és komplexebb matematikai műveleteket lehet vele végezni vektorokon és mátrixokon egyszerű függvényhívásokkal, így kevésbé kell a matematikára koncentrálni.

2.3.4 stb_image[17]

Az stb_image egy kicsi könyvtár, amivel különféle formátumú képeket tudunk betölteni, akár HDR támogatással.

2.3.5 Loguru[18]

A Loguru egy egyszerűen használható és gyors logolási könyvtár. Egy függvényhívással lehet printf formázással[19] konzolra és fájlba írni előre formázott szövegeket, amiken látszik a hívás helye és ideje mellett az is, hogy melyik szálról jött. Emellett könnyen testre szabható, hogy mit hova írjon ki.

2.3.6 tinyobjloader[20]

A tinyobjloader pontosan azt tudja amire a neve is utal, egy nagyon kicsi könyvtár, amivel Wavefront[21] formátumú 3D modelleket lehet betölteni egészen gyorsan és memória effektíven.

2.3.7 ImGui[3]

Az ImGui az egyik legelterjedtebb és legközkedveltebb C++ könyvtár grafikus felületek készítéséhez. Egyszerű használni, bármilyen renderer-rel működik, bármilyen platformon, és nincsenek külső függőségei. Azonnali módú megjelenítése miatt ideális

szerkesztők felhasználói felületének készítésére. Nagyon sok előre elkészített komponenst és példát tartalmaz, amik alapján könnyedén lehet egy egyszerűen használható felhasználói felületet összerakni vele.

2.3.8 ImGuizmo[22]

Az ImGuizmo egy az ImGui-ra épülő könyvtár, ami egy gizmot rajzol a képernyőre, amivel egy mátrixot tudunk manipulálni az egérrel. Segítségével könnyedén és a 3D-s szerkesztőkhöz hasonlóan, intuitívan tudjuk mozgatni, forgatni és skálázni az objektumokat.

2.3.9 JSON[23]

Nagyon sok json könyvtár van c++-hoz, azért esett erre a választásom mert írni és olvasni is tud json fájlokat és nagyon könnyen lehet használni hogyha tudjuk, hogy hogyan van strukturálva az adatunk.

2.4 Projekt felépítés

Ebben a részben a projekt felépítését szeretném ismertetni, hogy mi miért van ott, ahol.

2.4.1 Mappaszerkezet

- CatEngine – A szerkesztő forráskódja.
    - assets – A felhasznált modellek, textúrák és bináris kódra lefordított shaderek, a nekik megfelelő nevű mappában.
    - shaders – A shaderek forráskódja.
    - Cat – A szerkesztő alkalmazás teljes forráskódja.
- Libraries – Minden felhasznált könyvtár a saját mappájában és a hozzájuk tartozó CMakeLists fájl a fordításhoz.
- out – A célmappa, ahova a projekt fordul, innen lehet elindítani fordítás után.

2.4.2 Projekt fordítása

Mivel most már szinte mindegyik nagy új fejlesztő környezet támogatja a CMakeList es projekteket, ezért egyész könnyedén lehet ezekkel lefordítani a projektet. Klónozás után a Visual Studio 2022, a CLion és a Visual Studio Code is felismeri egyből a projektet és ha van megfelelő fordítónk, MSVC biztosan működik, és a Vulkan SDK is fel van telepítve akkor le is tudjuk egyből fordítani mert a projekt tartalmazza a függőségeit is lokálisan.

2.4.2.1 Egész alkalmazás projektfájl

Az elején beállításra kerülnek a platform és fordító függő változók. Ezután hozzá van adva minden forrás fájl, mint fordításra szánt fájl. Ezek után hozzáadásra kerül a Vulkan és minden más könyvtár a Libraries mappából.

```
find_package(Vulkan REQUIRED)
target_include_directories(${PROJECT_NAME} PUBLIC ${Vulkan_INCLUDE_DIRS})
target_link_libraries(${PROJECT_NAME} glm tinyobjloader stb_image loguru
imgui ImGuizmo json)
target_link_libraries(${PROJECT_NAME} Vulkan::Vulkan glfw)
```

Ezt követően a hozzáadom a Shaderek fordítására írt cmake függvényt. Ez név alapján betölt egy shadert, lefordítja SPIR-V[24] reprezentációra, majd áthelyezi az assets/shaders mappába.

```
function(add_shader TARGET SHADER)
    # Find glslc shader compiler.
    find_program(GLSLC glslc)

    # All shaders for a sample are found here.
    set(current-shader-path
${CMAKE_SOURCE_DIR}/${PROJECT_NAME}/shaders/${SHADER})

     # Output path is inside the assets folder.
    set(current-output-path
${CMAKE_SOURCE_DIR}/${PROJECT_NAME}/assets/shaders/${SHADER}.spv)

    # Add a custom command to compile GLSL to SPIR-V.
    get_filename_component(current-output-dir ${current-output-path}
DIRECTORY)
    file(MAKE_DIRECTORY ${current-output-dir})
    add_custom_command(
            OUTPUT ${current-output-path}
            COMMAND ${GLSLC} -o ${current-output-path} ${current-shader-
path}
            DEPENDS ${current-shader-path}
            IMPLICIT_DEPENDS CXX ${current-shader-path}
            VERBATIM)

    # Make sure our native build depends on this output.
```

```
    set_source_files_properties(${current-output-path} PROPERTIES
GENERATED TRUE)
    target_sources(${TARGET} PRIVATE ${current-output-path})
endfunction(add_shader)
```

Ezek után betöltöm az összes shadert a mappából és mindegyikre meghívom a függvényt, majd az egész assets mappát átmásolom a kimeneti célmappába.

```
add_custom_command(TARGET ${PROJECT_NAME} POST_BUILD
        COMMAND ${CMAKE_COMMAND} -E copy_directory
        ${CMAKE_SOURCE_DIR}/${PROJECT_NAME}/assets
$<TARGET_FILE_DIR:${PROJECT_NAME}>/assets)

include(Shaders.cmake)

# Find all shaders.
file(GLOB vertex-shaders
${CMAKE_SOURCE_DIR}/${PROJECT_NAME}/shaders/*.vert)
file(GLOB fragment-shaders
${CMAKE_SOURCE_DIR}/${PROJECT_NAME}/shaders/*.frag)
file(GLOB compute-shaders
${CMAKE_SOURCE_DIR}/${PROJECT_NAME}/shaders/*.comp)

# Add them to the build.
foreach (vertex-shader ${vertex-shaders})
    get_filename_component(shader ${vertex-shader} NAME)
    add_shader(${PROJECT_NAME} ${shader})
endforeach (vertex-shader)

foreach (fragment-shader ${fragment-shaders})
    get_filename_component(shader ${fragment-shader} NAME)
    add_shader(${PROJECT_NAME} ${shader})
endforeach (fragment-shader)

foreach (compute-shader ${compute-shaders})
    get_filename_component(shader ${compute-shader} NAME)
    add_shader(${PROJECT_NAME} ${shader})
endforeach (compute-shader)

set(output-assets $<TARGET_FILE_DIR:${PROJECT_NAME}>/${TARGET}/assets)
```

2.4.2.2 Függőségi könyvtárak projektfájlja

Ebben a fájlban van minden könyvtár hozzáadva, hogy azokat hogyan kell fordítani.

```
find_package(Vulkan REQUIRED)

add_library(glm INTERFACE)
target_include_directories(glm INTERFACE glm)

add_library(stb_image INTERFACE)
target_include_directories(stb_image INTERFACE stb-master)

add_library(glfw INTERFACE)
target_include_directories(glfw INTERFACE glfw-3.3.4.bin.WIN64/include)
target_link_directories(glfw INTERFACE glfw-3.3.4.bin.WIN64/lib-vc2019)
```

```
target_link_libraries(glfw INTERFACE GLFW3)

add_library(loguru STATIC)
target_sources(loguru PRIVATE
    "loguru-2.1.0/loguru.hpp"
    "loguru-2.1.0/loguru.cpp"
    )
target_include_directories(loguru PUBLIC loguru-2.1.0)

add_library(tinyobjloader STATIC)
target_sources(tinyobjloader PRIVATE
    "tinyobjloader-master/tiny_obj_loader.h"
    "tinyobjloader-master/tiny_obj_loader.cc"
    )
target_include_directories(tinyobjloader PUBLIC tinyobjloader-master)

add_library(imgui STATIC)
target_include_directories(imgui PUBLIC
"${CMAKE_CURRENT_SOURCE_DIR}/imgui")

file(GLOB sources imgui/*.cpp)

message("${sources}")

target_sources(imgui PRIVATE
    "${sources}"

    "${CMAKE_CURRENT_SOURCE_DIR}/imgui/imgui_impl_vulkan.cpp"
    "${CMAKE_CURRENT_SOURCE_DIR}/imgui/imgui_impl_glfw.cpp"

    "${CMAKE_CURRENT_SOURCE_DIR}/imgui/imgui_internal.h"
    "${CMAKE_CURRENT_SOURCE_DIR}/imgui/imgui.h"
    "${CMAKE_CURRENT_SOURCE_DIR}/imgui/imgui.cpp"

    "${CMAKE_CURRENT_SOURCE_DIR}/imgui/imgui_demo.cpp"
    "${CMAKE_CURRENT_SOURCE_DIR}/imgui/imgui_draw.cpp"
    "${CMAKE_CURRENT_SOURCE_DIR}/imgui/imgui_widgets.cpp"
    )
target_link_libraries(imgui PUBLIC "glfw;Vulkan::Vulkan")

add_library(ImGuizmo STATIC)
target_sources(ImGuizmo PRIVATE
    "ImGuizmo-1.83/ImGuizmo.h"
    "ImGuizmo-1.83/ImGuizmo.cpp"
    )
target_include_directories(ImGuizmo PUBLIC "ImGuizmo-1.83")
target_link_libraries(ImGuizmo imgui)

add_library(json INTERFACE)
target_include_directories(json INTERFACE json)
```

Ezek mind működnek bármilyen konfiguráció nélkül, mert hozzá vannak adva a projekthez.

2.5 Alkalmazás felépítése

Ebben a fejezetben az alkalmazás kódjának a felépítését és működését mutatom be.

2.5.1 Kód stílus

2.5.1.1 Formázás

A kód formázására ClangFormat[26]-ot használok egy saját konfigurációval, mert minden fejlesztő környezet támogatja. A konfigurációm a minél jobb átláthatóságra van kiélezve, így például minden kapcsos zárójel a saját sorába kerül, kivéve, ha az egész tartalma kifér egy sorba, ami maximum 128 karakter lehet. Emellett minden zárójel szóközökkel van kibélelve.

```
---
Language: Cpp
BasedOnStyle: Chromium
Standard: c++20
AccessModifierOffset: -4
AlignAfterOpenBracket: DontAlign
AllowShortFunctionsOnASingleLine: InlineOnly
AllowShortIfStatementsOnASingleLine: true
AlwaysBreakAfterDefinitionReturnType: false
BraceWrapping:
  AfterCaseLabel: false
  AfterClass: true
  AfterControlStatement: true
  AfterEnum: true
  AfterFunction: true
  AfterNamespace: true
  AfterStruct: true
  AfterUnion: true
  AfterExternBlock: false
  BeforeCatch: false
  BeforeElse: true
  BeforeLambdaBody: true
  SplitEmptyFunction: true
  SplitEmptyRecord: true
  SplitEmptyNamespace: true
BreakBeforeBraces: Custom
BreakBeforeBinaryOperators: NonAssignment
ColumnLimit: 128
EmptyLineBeforeAccessModifier: LogicalBlock
IndentCaseLabels: true
IndentWidth: 4
MaxEmptyLinesToKeep: 2
SortIncludes: false
TabWidth: 4
UseTab: Always
DerivePointerAlignment: false
PointerAlignment: Left
# ReferenceAlignment: Left
```

```
# SpaceAfterCStyleCast: true
SpaceBeforeParens: ControlStatements
SpacesInAngles: true
SpacesInConditionalStatement: true
SpacesInParentheses: true
SpacesBeforeTrailingComments: 1
SpacesInLineCommentPrefix:
  Minimum: 1
  Maximum: 1
```

2.5.1.2 Névadás

Megpróbáltam minden változónak és függvénynek olyan nevet adni, amiről lehet tudni, hogy az mire való, és nem használni rövidítéseket, mert az automatikus kód formázás miatt, ha valami túl hosszú is lenne, akkor csak szépen új sorba törik.

Az általános szabály az, hogy minden függvény és változó lowerCamelCase[27], az osztályok és globális függvények UpperCamelCase, a globális konstansok pedig ALL_UPPER stílust használnak.

Emellett használom a magyar jelölést[28], minden változó megkapja a típusát a neve elé egy karakterként, a tagváltozók pedig egy „m_” előjelet is kapnak.

2.5.2 Fő részek felépítése

Minden általam írt, az alkalmazáshoz tartozó kód a cat névtér alatt van és emellett minden kívülre látható osztály kapott egy Cat előjelet.

2.5.2.1 Globals.hpp

A Globals.hpp tartalmazza a Vulkan és a GLWF include-jait, pár kényelmi definíciót, hogy c++ 20-as designated initializers[25]-t lehessen használni Vulkan-nal, és néhány statikus globális változót.

2.5.2.2 CatApp

Ez az osztály maga az editor alkalmazás, ez tartalmaz minden rendereléshez szükséges változót, mint például az ablak, a videókártya vagy a renderer, emellett a kirajzolandó objektumok is itt vannak és a fő képkocka frissítő és kirajzoló loop is itt található. Ez inicializálja is az egész alkalmazást.

2.5.2.3 CatFrameInfo

Itt találhatóak a rendereléshez szükséges változók, amik bármikor megváltozhatnak. Itt van definiálva az UBO[29] is, ami a kamerát és a fényeket tartalmazza.

```
static constexpr auto MAX_LIGHTS = 16;

struct PointLight
{
     glm::vec4 position{}; // ignore w
     glm::vec4 color{};     // w is intensity
};

struct GlobalUbo
{
     glm::mat4 projection{ 1.f };
     glm::mat4 view{ 1.f };
     glm::mat4 inverseView{ 1.f };
     glm::vec4 ambientLightColor{ .8f, .8f, 1.f, .086f }; // w is
intensity
     PointLight pointLights[MAX_LIGHTS];
     int numLights;
};

using CatFrameInfo = struct CatFrameInfo_t
{
     short m_nFrameIndex = 0;
     uint64_t m_nFrameNumber = 0;
     double m_dFrameTime = 0.0;
     vk::CommandBuffer m_pCommandBuffer;
     CatCamera& m_rCamera;
     CatObject& m_rCameraObject;
     vk::DescriptorSet m_pGlobalDescriptorSet;
     GlobalUbo& m_rUBO;
     CatObject::Map& m_mObjects;
     CatObject::id_t m_selectedItemId;
…
```

2.5.2.4 CatWindow

Ez az osztály kezeli a GLFW-s ablak létrehozását és ennek az átméretezésének a követését.

2.5.2.5 CatImgui

Ez az osztály kezeli az ImGui inicializálását, az ImGui ablakok felépítését és ezek kirajzolását.

2.5.2.6 CatSimpleRenderSystem, CatPointLightRenderSystem és CatWireframeRenderSystem

Ezek az osztályok kezelik a nekik megfelelő objektumok frissítését és videókártyára másolását, hogy azokat ki lehessen rajzolni. Nevüknek megfelelően a Simple felel bármilyen nem különleges, modellel rendelkező objektumért, a PointLight a fények, a Wireframe pedig a betöltést előidéző formák kirajzolásáért felel. Emellett mindegyik elkészíti a pipeline-t, ami a shaderek viselkedését szabályozza és az infókat, amik feltöltésre kerülnek a kirajzoláshoz.

Példa pipeline készítésre:

```
void CatPointLightRenderSystem::createPipeline( vk::RenderPass pRenderPass
)
{
     assert( !!m_pPipelineLayout && "Cannot create pipeline before
pipeline layout" );

     PipelineConfigInfo pipelineConfig{};
     CatPipeline::defaultPipelineConfigInfo( pipelineConfig );
     CatPipeline::enableAlphaBlending( pipelineConfig );
     CatPipeline::disableBackFaceCulling( pipelineConfig );
     pipelineConfig.m_aAttributeDescriptions.clear();
     pipelineConfig.m_aBindingDescriptions.clear();
     pipelineConfig.m_pRenderPass = pRenderPass;
     pipelineConfig.m_pPipelineLayout = m_pPipelineLayout;
     m_pPipeline = std::make_unique< CatPipeline >(
          m_rDevice, "assets/shaders/point_light.vert.spv",
"assets/shaders/point_light.frag.spv", pipelineConfig );
}
```

Példa kirajzolásra:

```
void CatWireframeRenderSystem::renderObjects( const CatFrameInfo&
frameInfo )
{
     m_pPipeline->bind( frameInfo.m_pCommandBuffer );

     frameInfo.m_pCommandBuffer.bindDescriptorSets(
          vk::PipelineBindPoint::eGraphics, m_pPipelineLayout, 0, 1,
&frameInfo.m_pGlobalDescriptorSet, 0, nullptr );

     for ( auto& [key, obj] : frameInfo.m_mObjects )
     {
          if ( obj->m_pModel == nullptr ) continue;
          if ( auto volume = dynamic_cast< CatVolume* >( obj.get() ) )
          {
               CatPushConstantData push{};
               push.m_mxModel = volume->m_transform.mat4();
               push.m_mxNormal = volume->m_transform.normalMatrix();

               frameInfo.m_pCommandBuffer.pushConstants(
m_pPipelineLayout,
```

```
                    vk::ShaderStageFlagBits::eVertex |
vk::ShaderStageFlagBits::eFragment, 0, sizeof( CatPushConstantData ),
                    &push );
               volume->m_pModel->bind( frameInfo.m_pCommandBuffer );
               volume->m_pModel->draw( frameInfo.m_pCommandBuffer );
          }
     }
}
```

2.5.2.7 CatCamera

A kamera nézet és projekciós mátrixainak az előállítását végzi, glm segítségével, amik alapján a modellek transzformálva vannak.

```
void CatCamera::setViewYXZ( glm::vec3 vPosition, glm::vec3 vRotation )
{
     m_mxView = glm::lookAtRH( vPosition, vPosition + vRotation,
glm::vec3{ 0.f, 1.f, 0.f } );

     m_mxInverseViewMatrix = glm::inverse( m_mxView );
}
```

2.5.2.8 CatInput

A kamera mozgatását és forgatását végzi, gombok megnyomására és az egér mozgatására jobb egér gomb lenyomása közben.

Gombok:

```
struct KeyMappings
{
     int look = GLFW_MOUSE_BUTTON_RIGHT;
     int moveLeft = GLFW_KEY_A;
     int moveRight = GLFW_KEY_D;
     int moveForward = GLFW_KEY_W;
     int moveBackward = GLFW_KEY_S;
     int moveUp = GLFW_KEY_E;
     int moveDown = GLFW_KEY_Q;
     int speed = GLFW_KEY_LEFT_SHIFT;
     int lookLeft = GLFW_KEY_LEFT;
     int lookRight = GLFW_KEY_RIGHT;
     int lookUp = GLFW_KEY_UP;
     int lookDown = GLFW_KEY_DOWN;
};
```

Mozgás: először, ha lenyomtuk a kamera forgás gombot, akkor kikapcsoljuk a kurzort és megnézzük mennyit mozgott az egér és arra forgatjuk a kamerát, aztán a mozgás irány gomboknak megfelelően mozgatjuk a kamerát.

```
void CatInput::moveInPlaneXZ( GLFWwindow* window, float dt, CatObject&
gameObject )
{
     if ( glfwGetMouseButton( window, m_eKeys.look ) == GLFW_RELEASE )
     {
          m_bFirstMouse = true;
```

```
            glfwSetInputMode( window, GLFW_CURSOR, GLFW_CURSOR_NORMAL );
        }

        if ( glfwGetMouseButton( window, m_eKeys.look ) == GLFW_PRESS )
        {
            glfwSetInputMode( window, GLFW_CURSOR, GLFW_CURSOR_DISABLED );
            double* pXPos = new double;
            double* pYPos = new double;
            glfwGetCursorPos( window, pXPos, pYPos );
            if ( pXPos && pYPos )
            {
                if ( m_bFirstMouse )
                {
                    m_vMouseLastPos.x = *pXPos;
                    m_vMouseLastPos.y = *pYPos;
                    m_bFirstMouse = false;
                }

                float fXOffset = *pXPos - m_vMouseLastPos.x;
                float fYOffset = m_vMouseLastPos.y - *pYPos; // reversed
since y-coordinates go from bottom to top

                m_vMouseLastPos.x = *pXPos;
                m_vMouseLastPos.y = *pYPos;

                m_fYaw += fXOffset * m_fMouseSensitivity;
                m_fPitch += fYOffset * m_fMouseSensitivity;
            }
            delete pXPos;
            delete pYPos;
        }
        m_fPitch = glm::clamp( m_fPitch, -1.5f, 1.5f );
        m_fYaw = glm::mod( m_fYaw, glm::two_pi< float >() );

        glm::vec3 vFront;
        vFront.x = cos( m_fYaw ) * cos( m_fPitch );
        vFront.y = sin( m_fPitch );
        vFront.z = sin( m_fYaw ) * cos( m_fPitch );
        // normalize the vectors, because their length gets closer to 0 the
more you look up or down which results in slower
        // movement.
        vFront = glm::normalize( vFront );
        glm::vec3 vRight = glm::normalize( glm::cross( vFront, glm::vec3{
0.f, 1.f, 0.f } ) );
        glm::vec3 vUp = glm::normalize( glm::cross( vRight, vFront ) );

        gameObject.m_transform.rotation = vFront;

        glm::vec3 moveDir{ 0.f };
        if ( glfwGetKey( window, m_eKeys.moveForward ) == GLFW_PRESS )
moveDir += vFront;
        if ( glfwGetKey( window, m_eKeys.moveBackward ) == GLFW_PRESS )
moveDir -= vFront;
        if ( glfwGetKey( window, m_eKeys.moveRight ) == GLFW_PRESS ) moveDir
+= vRight;
        if ( glfwGetKey( window, m_eKeys.moveLeft ) == GLFW_PRESS ) moveDir -
= vRight;
        if ( glfwGetKey( window, m_eKeys.moveUp ) == GLFW_PRESS ) moveDir +=
vUp;
```

```
     if ( glfwGetKey( window, m_eKeys.moveDown ) == GLFW_PRESS ) moveDir -
= vUp;

     if ( glfwGetKey( window, m_eKeys.speed ) == GLFW_PRESS )
m_fMovementSpeed = 6.9f;
     if ( glfwGetKey( window, m_eKeys.speed ) == GLFW_RELEASE )
m_fMovementSpeed = 3.0f;

     if ( glm::dot( moveDir, moveDir ) > std::numeric_limits< float
>::epsilon() )
     {
          gameObject.m_transform.translation += m_fMovementSpeed * dt *
glm::normalize( moveDir );
     }
}
```

2.5.2.9 CatModel

Egy 3D-s modell betöltésért, tárolásáért, memóriába feltöltéséért és kirajzolásáért felelős.

Egy vertex-ről tárolt adatok:

```
     struct Vertex
     {
          glm::vec3 m_vPosition{};
          glm::vec3 m_vColor{};
          glm::vec3 m_vNormal{};
          glm::vec2 m_vUV{};

          static std::vector< vk::VertexInputBindingDescription >
getBindingDescriptions();
          static std::vector< vk::VertexInputAttributeDescription >
getAttributeDescriptions();

          bool operator==( const Vertex& other ) const
          {
               return m_vPosition == other.m_vPosition && m_vColor ==
other.m_vColor && m_vNormal == other.m_vNormal
                         && m_vUV == other.m_vUV;
          }
     };
```

Modellek betöltése:

```
void CatModel::Builder::loadModel( const std::string& filepath )
{
     tinyobj::attrib_t attrib;
     std::vector< tinyobj::shape_t > shapes;
     std::vector< tinyobj::material_t > materials;
     std::string warn, err;

     if ( !tinyobj::LoadObj( &attrib, &shapes, &materials, &warn, &err,
filepath.c_str() ) )
     {
          throw std::runtime_error( warn + err );
     }
```

```
        m_aVertices.clear();
        m_aIndices.clear();

        std::unordered_map< Vertex, uint32_t > uniqueVertices{};
        for ( const auto& shape : shapes )
        {
             for ( const auto& index : shape.mesh.indices )
             {
                  Vertex vertex{};

                  if ( index.vertex_index >= 0 )
                  {
                       vertex.m_vPosition = {
                            attrib.vertices[3 * index.vertex_index + 0],
                            attrib.vertices[3 * index.vertex_index + 1],
                            attrib.vertices[3 * index.vertex_index + 2],
                       };

                       vertex.m_vColor = {
                            attrib.colors[3 * index.vertex_index + 0],
                            attrib.colors[3 * index.vertex_index + 1],
                            attrib.colors[3 * index.vertex_index + 2],
                       };
                  }

                  if ( index.normal_index >= 0 )
                  {
                       vertex.m_vNormal = {
                            attrib.normals[3 * index.normal_index + 0],
                            attrib.normals[3 * index.normal_index + 1],
                            attrib.normals[3 * index.normal_index + 2],
                       };
                  }

                  if ( index.texcoord_index >= 0 )
                  {
                       vertex.m_vUV = {
                            attrib.texcoords[2 * index.texcoord_index + 0],
                            attrib.texcoords[2 * index.texcoord_index + 1],
                       };
                  }

                  if ( !uniqueVertices.contains( vertex ) )
                  {
                       uniqueVertices[vertex] = static_cast< uint32_t >(
m_aVertices.size() );
                       m_aVertices.push_back( vertex );
                  }
                  m_aIndices.push_back( uniqueVertices[vertex] );
             }
        }
}
```

2.5.2.10 CatObject, CatLight és CatVolume

A CatObjectből származik le a másik kettő, és csak abban különböznek, hogy a fényeknek nincs, a betöltős formáknak pedig mindig egy kocka a modellje, és így

polimorfizmus alapján tudom tesztelni, hogy egy objektum fény vagy forma e és az alapján kirajzolni őket.

Minden objektumnak a transzformját ez az osztály tárolja:

```
struct TransformComponent
{
	glm::vec3 translation{};
	glm::vec3 rotation{};
	glm::vec3 scale{ 1.f, 1.f, 1.f };

	// Matrix corrsponds to Translate * Ry * Rx * Rz * Scale
	// Rotations correspond to Tait-bryan angles of Y(1), X(2), Z(3)
	// https://en.wikipedia.org/wiki/Euler_angles#Rotation_matrix
	[[nodiscard]] glm::mat4 mat4() const;

	[[nodiscard]] glm::mat3 normalMatrix() const;
};
```

És a mátrix, amivel a videókártya tud számolni így kerül kiszámolásra:

```
glm::mat4 TransformComponent::mat4() const
{
	auto m = glm::mat4( 1.0f );

	const auto rotationRad = glm::radians( rotation );

	m = glm::translate( m, translation );
	m = glm::rotate( m, rotationRad.y, { 0.f, 1.f, 0.f } );
	m = glm::rotate( m, rotationRad.x, { 1.f, 0.f, 0.f } );
	m = glm::rotate( m, rotationRad.z, { 0.f, 0.f, 1.f } );
	m = glm::scale( m, scale );

	return m;
}
```

2.5.2.11 CatDevice

2.5.2.12 CatRenderer

2.5.2.13 CatSwapChain

2.5.2.14 CatPipeline

2.5.2.15 CatDescriptors

2.5.2.16 CatBuffer

2.5.3 Az alkalmazás működése

2.5.3.1 Belépési pont

A program belépési pontja a main.cpp main függvénye, amiben inicializálódik a loguru és a két kimeneti log fájl, az egyikbe minden belekerül az összes futásból, a másikba a jelenlegi futás csak olvasható logjai, és létrejön egy alkalmazás példány, amit egyből el is indítunk, ami már csak akkor tér vissza, ha kilépünk az alkalmazásból, így ezután a destruktort is meghívjuk és kilépünk.

```
int main( int argc, char** argv )
{
	loguru::init( argc, argv );
	// Put every log message in "everything.log":
	loguru::add_file( "everything.log", loguru::Append,
loguru::Verbosity_MAX );
	// Only log INFO, WARNING, ERROR and FATAL to "latest_readable.log":
	loguru::add_file( "latest_readable.log", loguru::Truncate,
loguru::Verbosity_INFO );

	// Only show most relevant things on stderr:
	loguru::g_stderr_verbosity = 1;


	try
	{
		cat::CreateEditorInstance();

		// Main Loop
		cat::GetEditorInstance()->run();


		cat::DestroyGameInstance();

	} catch ( const std::exception& e )
	{
		ABORT_F( e.what() );
		std::cerr << e.what() << '\n';
		return EXIT_FAILURE;
	}

	return EXIT_SUCCESS;
}
```

3 Howto

3.1.1 Címsorok

A fejezetcímek esetén a **Címsor 1-4** (Heading 1-4) stílusokat használjuk.

3.1.2 Képek

A képhez használjuk a **Kép** stílust.

Képaláírást a képen jobb gombbal kattintva a *Képaláírás beszúrása…* opcióval adhatunk hozzá, így az automatikusan **Képaláírás** (Caption) stílusú lesz.

3.1. ábra: Példa képaláírásra

3.1.3 Kódrészletek

Kódrészletek beillesztése esetén használjuk a **Kód** stílust.

```
using System;
namespace MyApp
{
      class Program
      {
            static void Main( string[] args )
            {
                  Console.WriteLine( "Szia Világ!" );
            }
      }
}
```

3.1.4 Irodalomjegyzék

Az Irodalomjegyzékben szereplő hivatkozásokat **Irodalomjegyzék sor** stílussal formázzuk, a címüket pedig **Irodalomjegyzék forrás** stílussal emeljük ki.

A szövegbe a hivatkozásokat**Hiba! A hivatkozási forrás nem található.** a *Kereszthivatkozás beszúrása* (Insert cross-reference) funkcióval helyezzük el (példa egy így beszúrt hivatkozásra: **Hiba! A hivatkozási forrás nem található.**), így azok a utomatikusan frissülnek a hivatkozások átrendezésekor.

4 Utolsó simítások

Miután elkészültünk a dokumentációval, ne felejtsük el a következő lépéseket:

- *Kereszthivatkozások frissítése:* miután kijelöltük a teljes szöveget (Ctrl+A), nyomjuk meg az F9 billentyűt, és a Word frissíti az összes kereszthivatkozást. Ilyenkor ellenőrizzük, hogy nem jelent-e meg valahol a "Hiba! A könyvjelző nem létezik." szöveg.
- *Dokumentum tulajdonságok megadása:* a dokumentumhoz tartozó meta adatok kitöltése (szerző, cím, kulcsszavak stb.). Erre való a Dokumentum tulajdonságai panel, mely a Fájl / Információ / Tulajdonságok / Dokumentumpanel megjelenítése úton érhető el.
- *Kinézet ellenőrzése PDF-ben:* a legjobb teszt a végén, ha PDF-et készítünk a dokumentumból, és azt leellenőrizzük.

Irodalomjegyzék

[1] Vulkan Tutorial: https://vulkan-tutorial.com/, (2021. március)

[2] Vulkan-HPP: *C++ Bindings for Vulkan, GitHub:* https://github.com/KhronosGroup/Vulkan-Hpp, (2021. március)

[3] Dear ImGui: *Omar Cornut, Github,* https://github.com/ocornut/imgui, (2022. május)

[4] Vulkan Game Engine Tutorial sorozat: *Brendan Galea, Youtube:* https://www.youtube.com/playlist?list=PL8327DO66nu9qYVKLDmdLW_84-yE4auCR, (2021. augusztus)

[5] GitHub: https://github.com/

[6] GitKraken: https://www.gitkraken.com/

[7] Visual Studio 2022: *Microsoft,* https://visualstudio.microsoft.com/vs/

[8] Resharper C++: *JetBrains,* https://www.jetbrains.com/resharper-cpp/

[9] Visual Assist: *WholeTomato,* https://www.wholetomato.com/features

[10] CLiom: *JetBrains,* https://www.jetbrains.com/clion/

[11] Visual Studio Code: https://code.visualstudio.com/

[12] CMake: https://cmake.org/

[13] Vulkan SDK: *LunarG,* https://www.lunarg.com/vulkan-sdk/, (1.3.211.0, 2022. április)

[14] Vulkan: *Wikipédia,* https://en.wikipedia.org/wiki/Vulkan, (2022. május)

[15] GLFW: https://www.glfw.org/, (3.3.4, 2021. október)

[16] GLM: *Github,* https://github.com/g-truc/glm, (0.9.9.8, 2020. április)

[17] stb_image: *stb, nothings, Github,* https://github.com/nothings/stb (2.27, 2021. október)

[18] Loguru: *Loguru c++, emilk, Github,* https://github.com/emilk/loguru (2.1.0, 2019. szeptember)

[19] Printf formázás: *Wikipédia,* https://en.wikipedia.org/wiki/Printf_format_string#:~:text=%22printf%22%20is%20the%20name%20of,parsing). (2022. május)

[20] tinyobjloader: *Github,* https://github.com/tinyobjloader/tinyobjloader (2.0-rc1, 2019. július)

[21] WaveFront obj: *Wikipédia*, https://en.wikipedia.org/wiki/Wavefront_.obj_file (2022. május)

[22] ImGuizmo: *CedricGuillemet, Github*, https://github.com/CedricGuillemet/ImGuizmo (1.83, 2021. július)

[23] JSON: *nlohmann, Github*, https://github.com/nlohmann/json (3.10.5, 2022. január)

[24] SPIR-V: *Khronos Group*, https://www.khronos.org/spir/, (2022. május)

[25] Designated initializers: *cppreference*, https://en.cppreference.com/w/cpp/language/aggregate_initialization, (2022. május)

[26] ClangFormat: *LLVM Clang*, https://releases.llvm.org/14.0.0/tools/clang/docs/ClangFormat.html, (14.0.0, 2022. március 25.)

[27] Camel case: *Wikipédia*, https://en.wikipedia.org/wiki/Camel_case, (2022. május)

[28] Hungarian notation: *Wikipédia*, https://en.wikipedia.org/wiki/Hungarian_notation, https://hu.wikipedia.org/wiki/Magyar_jel%C3%B6l%C3%A9s, (2022. május)

[29] Uniform Buffer Object: *Khronos Wiki*, https://www.khronos.org/opengl/wiki/Uniform_Buffer_Object, (2022. május)

[30]

Függelék