Vulkan 3d Rendering,Drawing (Вазна)

203064 Давор Јолески Computer Graphics

* .first.cpp ->Целта на кодот е да вчита текстура и да ја постави како сет на дескриптори што може да се користи за рендерирање во прогрмата.

Кодот најпрво создава базен на дескриптори (globalPool) со одреден максимален број на множества на дескриптори и додава две големини на базен за униформни бафери и комбинирани семплери на слики. Потоа, ги вчитува објектите на програматаг користејќи го методот loadGameObjects.

Во методот на извршување, тој создава низа од униформни баферски објекти, ги мапира баферите за да биде домаќин на видлива меморија и ја вчитува текстурата од патеката на датотеката. Информациите за текстурата се зачувуваат во структура VkDescriptorImageInfo.

Следно, кодот создава распоред на глобално множество дескриптори со две врзувања, една за униформни бафери и една за комбиниран семплер на слики. Потоа создава низа од глобални множества дескриптори, ги запишува буфферот и информациите за сликата во множествата и ги гради множествата.

Конечно, кодот поставува едноставен систем за рендерирање и систем за точка светло и создава камера. Камерата е контролирана од контролорот за движење на тастатурата, а играта се води во циклус каде што ја ажурира камерата и ја прикажува сцената.Loading<Models &fromfilepath>вметнуваме модел за прикажување на програмта со координати и трнаслации

Vo frist.hpp gi imame inicializirano funkciiite koi trebat da gi implementirame vo first.cpp so Class::Object

* 2. keyboard\_movement\_controller.cpp

Овој код имплементира едноставен контролер за движење на тастатурата во C++. Класата KeyboardMovementController обезбедува метод moveInPlaneXZ кој ја менува позицијата и ориентацијата на објектот LveGameObject врз основа на корисничкиот влез добиен од прозорецот GLFW.

Методот ја користи функцијата glfwGetKey на GLFW за да ја провери состојбата на наведените копчиња и соодветно да ја ажурира ориентацијата и позицијата на LveGameObject. Ротацијата се ажурира врз основа на состојбата на копчињата „погледни лево/десно“ и „поглед нагоре/надолу“, а позицијата се ажурира врз основа на состојбата на копчињата „движи напред/назад“ и „помести лево/десно“.

if (glfwGetKey(window, keys.lookRight) == GLFW\_PRESS) rotate.y += 1.f;

if (glfwGetKey(window, keys.lookLeft) == GLFW\_PRESS) rotate.y -= 1.f;

if (glfwGetKey(window, keys.lookUp) == GLFW\_PRESS) rotate.x += 1.f;

if (glfwGetKey(window, keys.lookDown) == GLFW\_PRESS) rotate.x -= 1.f;

Движењето нагоре/надолу се контролира со копчињата „помести нагоре/надолу“. Ротациите се прицврстени за да се ограничи аголот на наклон на +/- 85 степени, а аголот на скршнување е нормализиран за да се завитка на 0 кога ќе надмине 2π.

1. Keyboard\_movemnt\_controlle.hpp
2. float moveSpeed{ 3.f };
3. float lookSpeed{ 1.5f };

имплементирани за бризна на движњењ и вртење во околина иницајлизирани копичинања на кои букви ке функицонира

3.Compile.bat

Горенаведениот код е серија датотека на Windows која компајлира две датотеки со шејдер на GLSL, „shader.vert“ и „shader.frag“, во формат SPIR-V користејќи го компајлерот glslc од Vulkan SDK. Компајлираните датотеки SPIR-V ќе бидат зачувани како „shader.vert.spv“ и „shader.frag.spv“. Командата „пауза“ на крајот од датотеката предизвикува сериската датотека да чека за внесување од корисникот пред да заврши, што ви овозможува да видите какви било пораки за грешка пред да се затвори командниот прозорец.

Light.bat го прави истото

4.Lve\_buffer.cpp

Овој код е имплементација на класа на обвивка за Vulkan бафер, наречена LveBuffer. Целта на оваа класа е да обезбеди апстрактен и удобен начин за управување со баферот и неговата меморија во апликацијата Vulkan.

Кодот обезбедува разновидни методи за мапирање и запишување податоци во баферот, flushnuva го за да го направи видливо за уредот и откачете го. Конструкторот на класата зема параметри како што се големината на примероците на баферот, бројот на инстанци и својствата на употреба и меморија за баферот. Класата, исто така, обезбедува статичен метод, getAlignment(), за усогласување на големината на тампонот до одредено минимално порамнување, како што се бара од некои уреди.

Деструкторот на класата го чисти баферот и неговата меморија со повикување на vkDestroyBuffer и vkFreeMemory. Генерално, оваа класа обезбедува удобен и апстрактен начин за управување со Vulkan баферите и нивната меморија, поедноставувајќи го процесот на користење на бафери во Vulkan апликација.

.hpp se inicaializirani funkciite

5.LveCamera

->Овој код ја дефинира класата LveCamera, која претставува камера во 3D простор. Класата обезбедува методи за поставување на матрицата на проекцијата на камерата како правописна или перспективна проекција, како и методи за поставување на насоката на погледот и целта на камерата. Класата ја користи библиотеката GLM за векторски и матрични операции.

Методот setOrthographicProjection ја поставува матрицата на проекцијата како правописна проекција со дадените параметри. Ги поставува елементите на матрицата за проекција 4x4 според дадените вредности, кои ги дефинираат координатите на левата, десната, горната, долната, блиската и далечната рамнина на фрустумот на погледот.

Методот setPerspectiveProjection ја поставува матрицата на проекцијата како перспективна проекција со дадените параметри. Методот ги поставува елементите на матрицата да дефинираат перспективна проекција со дадениот агол на видното поле, аспект на односот на страните, блиска рамнина оддалеченост блиску и далечна рамнина далеку.

Методот setViewDirection ја поставува матрицата за приказ на камерата според дадената позиција, насока и вектори нагоре. Методот прво ги пресметува десните и горе векторите на камерата врз основа на дадената насока и векторите нагоре, а потоа соодветно ги поставува елементите на матрицата за гледање за да ја дефинира позицијата и ориентацијата на камерата во светскиот простор. Методот исто така ја пресметува и поставува матрицата за инверзен поглед.

Методот setViewTarget ги поставува матрицата за преглед и матрицата за инверзен поглед на камерата со повикување на методот setViewDirection со дадените вектори на позиција, цел и нагоре. Векторот на насоката се пресметува како разлика помеѓу векторите на целта и позицијата.

Методот setViewYXZ ги поставува матрицата за преглед и матрицата на инверзен поглед на камерата според дадената позиција и ротација во Ојлерови агли (редослед YXZ). Методот ги пресметува векторите на десната, нагоре и на насоката на камерата врз основа на дадените агли на ротација, а потоа соодветно ги поставува елементите на матрицата за гледање и матрицата на инверзен поглед за да ја дефинира позицијата и ориентацијата на камерата во светскиот простор.

6. lve\_descriptor.cpp

Овој код е наменет за распоред на сет на дескриптори и базен на дескриптори во приспособено графичко API изградено на врвот на Vulkan. Кодот дефинира две класи LveDescriptorSetLayout и LveDescriptorPool.

Класата LveDescriptorSetLayout е одговорна за креирање распоред на сет на дескриптори. Распоредот на сет на дескриптори ги опишува типовите на дескриптори кои можат да се врзат за цевката. Распоредот е креиран со користење на образец за градење каде што корисникот може да додаде врзување на изгледот. Секое врзување го одредува бројот на врзувањето, типот на дескрипторот, знаменцата на сцената и бројот на дескриптори.

Класата LveDescriptorPool е одговорна за креирање на базен на дескриптори. Базен со дескриптори е збирка на дескриптори. Описорите се уникатни референци за меморијата што може да се користат за складирање на информациите што ги бараат засенувачите, како што се текстурите и униформните бафери. Базенот е креиран со користење на шема за градење каде што корисникот може да го одреди бројот на дескриптори од секој тип, максималниот број на множества што може да се креираат од базенот и знаменцата на базенот.

Двете класи дефинираат конструктор што го поставува распоредот или базенот и деструктор што го уништува распоредот или базенот.

7.lve\_device.cpp

Овој код е дел од Vulkan моторот во C++ со помош на рамката LittleVulkanEngine. Кодот создава примерок на Vulkan и поставува месинџер за отстранување грешки за известување за грешки. Кодот потоа создава површина за рендерирање и избира физички уред за употреба во моторот. Кодот потоа создава логички уред и команден базен за распределба на командните бафери што се користат за рендерирање. Конечно, кодот има деструктор кој ги чисти создадените објекти во обратен редослед.

Функцијата createInstance() создава инстанца Vulkan и ги поставува потребните наставки и слоеви за валидација доколку се овозможени. Функцијата pickPhysicalDevice() избира физички уред од списокот на достапни уреди и функцијата createLogicalDevice() создава логички уред од избраниот физички уред. Функцијата createCommandPool() создава команден базен за доделување на командни бафери. Деструкторот на класата LveDevice ги чисти создадените објекти во обратен редослед за да обезбеди правилно распоредување на ресурсите.

Кодот вклучува неколку функции за иницијализирање на уредот:

createSurface: создава површина на прозорец за уредот

isDeviceSuitable: проверува дали дадениот физички уред е соодветен за апликацијата со проверка на достапноста на фамилиите на редици, екстензии на уредот и потребните функции

setupDebugMessenger: поставува гласник за отстранување грешки за уредот за да помогне при отстранување грешки

checkValidationLayerSupport: проверува дали потребните слоеви за валидација се поддржани од уредот

getRequiredExtensions: враќа список на потребни наставки за уредот

hasGflwRequiredInstanceExtensions: проверува дали се достапни потребните екстензии за GLFW

checkDeviceExtensionSupport: проверува дали бараните наставки на уредот се поддржани од уредот.

Има и помошни функции како што се „populateDebugMessengerCreateInfo“ која иницијализира структура потребна за создавање гласник за отстранување грешки и „findQueueFamilies“ што ги наоѓа индексите на фамилиите на редици поддржани од уредот. редици, секоја со една или повеќе редици. Овие редици се користат за поднесување команди до графичкиот процесор.

Структурата ги содржи индексите на две специфични фамилии на редици: graphicsFamily и presentFamily. GraphicsFamily се користи за графички операции, додека presentFamily се користи за прикажување слики на екранот.

Структурата вклучува и две булови вредности, graphicsFamilyHasValue и presentFamilyHasValue, кои покажуваат дали се пронајдени соодветните фамилии на редици. Конечно, структурата вклучува функција „isComplete()“ која враќа true доколку се пронајдени и graphicsFamily и presentFamily.

Класата не може да се копира или да се поместува за да се спречи ненамерно копирање или преместување на примероците на класата. Конструкторот на класата зема референца на пример од LveWindow како аргумент, кој е потребен за креирање на површината.

Lve\_frameinfo->

Овој код дефинира именски простор податоци што се користат за прикажување на 3D графика во Vulkan околина.

Структурата PointLight содржи информации за една точка светло во сцената, вклучувајќи ја нејзината позиција и боја.

Структурата GlobalUbo содржи информации што се користат за целата сцена, вклучувајќи ги матриците за проекција, поглед и инверзен поглед, бојата на амбиенталната светлина и низа точки светла. „Ubo“ е кратенка за „Uniform Buffer Object“.

Структурата FrameInfo содржи информации за една рамка во процесот на рендерирање, вклучувајќи го индексот на рамката, времето на рамката, командниот бафер Vulkan, референцата за објект на камерата, сет дескриптор за глобални информации и мапа на објекти на играта.

lveGameObject.cpp->

Овој код ја дефинира имплементацијата за класите TransformComponent и LveGameObject во именскиот простор „lve“. Класата TransformComponent содржи матрица 4x4 за позицијата,

ротацијата и размерот на објектот во 3D простор. Методот mat4 ја враќа матричната претстава на трансформацијата на објектот, а методот normalMatrix ја враќа инверзната транспонирање на подматрицата на ротација во скала 3x3 на трансформацијата на објектот.

Класата LveGameObject претставува објект на играта во моторот. Класата содржи објект TransformComponent, боја и уникатен покажувач на објект PointLightComponent, доколку објектот на играта е точка светло. Статичкиот метод makePointLight создава примерок LveGameObject и ги поставува неговите својства да бидат точка светло со даден интензитет, радиус и боја.

Објектот на играта има неколку членови на податоци:

lveGameObject.хпп:->

id: единствен идентификатор за објектот.

боја: бојата на предметот.

transform: структура TransformComponent која ја опишува положбата, скалата и ориентацијата на објектот во 3D простор.

модел: споделен покажувач на пример LveModel што ја претставува 3Д геометријата на објектот.

pointLight: уникатен покажувач кон примерок на PointLightComponent кој опишува точкаст извор на светлина поврзан со објектот.

Lve\_model.cpp

Овој код дефинира класа LveModel во именскиот простор lve. Класата LveModel се користи за претставување на 3D модел во моторот. Моделот е конструиран од темиња и индекси, кои може да се вчитаат од датотека користејќи ја функцијата loadModel во класата Builder.

Класата LveModel има функции за креирање на теме и индексни бафери, цртање на моделот и врзување на баферот на темето и индексниот бафер со команден тампон. Исто така, има објект lveDevice што е референца за објект LveDevice што претставува Vulkan логички уред.

Lve\_model.hpp

Овој код дефинира класа C++ со име „LveModel“. Класата е дел од именскиот простор „lve“. Класата претставува 3D модел што може да се нацрта на уред Vulkan.

Класата има две главни компоненти: темиња и индекси. Темињата се аголните точки на 3D моделот, а индексите се користат за да се опишат односите помеѓу темињата за да се формираат триаголници и други форми. Темињата на моделот се чуваат во структура „Вртекс“, која содржи информации за положбата, бојата, нормалната и УВ координатата на секое теме.

Класата „LveModel“ има два конструктори: еден што зема објект „LveDevice“ и објект „Builder“, а друг што зема објект „LveDevice“ и низа за патека на датотека. Објектот „Builder“ се користи за складирање на темињата и индексите на моделот, додека низата патека на датотеката се користи за вчитување на модел од датотека.

Класата "LveModel" има два главни методи: "врзува" и "црта". Методот „врзува“ се користи за поврзување на темето и индексните бафери на моделот со командниот бафер на Vulkan, додека методот „цртање“ се користи за издавање повик за извлекување на командниот бафер на Vulkan.

Класата има и статичен метод наречен „createModelFromFile“, кој создава примерок од класата „LveModel“ од патека на датотека. Овој метод е практична метода која ги комбинира функционалноста на објектот „Builder“ и конструкторот „LveModel“.

Lve\_pipline.cpp

Овој код е C++ имплементација на графички цевковод за употреба со графичкиот API на Vulkan. Гасоводот е креиран со користење на два шејдер модули (теме и фрагмент засенчувачи) и конфигурација на цевковод обезбедена во форма на структура PipelineConfigInfo.

Класата LvePipeline содржи конструктор кој зема референца за објект LveDevice (претставува физички уред во Vulkan API), патеките на датотеките за шејдерите на темето и фрагментите и информациите за конфигурацијата на гасоводот. Класата, исто така, содржи деструктор кој го уништува гасоводот, модулите за засенчување и другите ресурси што ги користи гасоводот.

Процесот на креирање на цевководот започнува со читање на содржината на датотеките со теме и фрагменти во меморијата користејќи го методот readFile. Потоа, методот createShaderModule се користи за креирање на вистинските модули за засенчување од кодот. Цевководот се креира со специфицирање на шејдерите, влезот на темето, приказот, растеризацијата и другите состојби во структурата VkGraphicsPipelineCreateInfo, која се пренесува на функцијата vkCreateGraphicsPipeline.

Класата, исто така, содржи имплементација на комунален метод readFile кој зема патека на датотека и ја чита содржината на датотеката во std::vector<char>.

Lve\_pipileine.hpp

„LvePipeline“. Целта на оваа класа е да создаде и управува со графички гасовод Vulkan. Графичкиот цевковод е суштинска компонента во Vulkan што го дефинира процесот на прикажување слика. Нафтоводот се состои од неколку фази, како што се внесување на теме, обработка на теме, обработка на фрагменти и мешање, а дефиницијата на цевководот одредува како овие фази имаат интеракција едни со други.

Класата зема како влез референца за пример на „LveDevice“, кој претставува Vulkan логички уред, патеките на датотеките за темето и засенчувачите на фрагментите и структурата „PipelineConfigInfo“ која содржи информации за конфигурацијата на гасоводот. Цевководот е креиран во конструкторот на класата, а цевководот може да се врзе за команден тампон користејќи го методот "bind".

Методите „defaultPipelineConfigInfo“ и „enableAlphaBlending“ се користат за поставување на стандардните конфигурации за гасоводот. Методот „defaultPipelineConfigInfo“ поставува стандардни вредности за различните состојби во цевководот, додека методот „enableAlphaBlending“ овозможува мешање на алфа.

Конечно, методот „readFile“ е помошна функција која чита датотека и ја враќа нејзината содржина како вектор од знаци. Методите „createGraphicsPipeline“, „createShaderModule“ се користат за креирање на графичкиот цевковод и модулите за шејдер, соодветно.

Lve\_render.cpp->

Ова е имплементација на класата LveRenderer која се користи за управување со операциите за рендерирање во апликацијата Vulkan. Класата е зависна од пример LveWindow и пример LveDevice за иницијализирање на синџирот на размена и ресурсите на уредот.

Класата е одговорна за управување со синџирот на размена и командните бафери што се користат при рендерирање. Има функции за повторно создавање на синџирот за размена кога се менува големината на прозорецот и доделување на командни бафери. Функциите BeginFrame и EndFrame се користат за започнување и завршување на рамка, соодветно. Функцијата beginSwapChainRenderPass се користи за започнување на пропусница за рендерирање за сликите на синџирот за замена. Функцијата getCurrentCommandBuffer го враќа командниот бафер за тековната рамка.

Класата е дизајнирана да се справи со сложеноста на рендерирањето на Vulkan, што го олеснува користењето за развивачот.

Lve\_redner.hpp->

Ова е имплементација на класата LveRenderer која се користи за управување со операциите за рендерирање во апликацијата Vulkan. Класата е зависна од пример LveWindow и пример LveDevice за иницијализирање на синџирот на размена и ресурсите на уредот.

Класата е одговорна за управување со синџирот на размена и командните бафери што се користат при рендерирање. Има функции за повторно создавање на синџирот за размена кога се менува големината на прозорецот и доделување на командни бафери. Функциите BeginFrame и EndFrame се користат за започнување и завршување на рамка, соодветно. Функцијата beginSwapChainRenderPass се користи за започнување на пропусница за рендерирање за сликите на синџирот за замена. Функцијата getCurrentCommandBuffer го враќа командниот бафер за тековната рамка.

Класата е дизајнирана да се справи со сложеноста на рендерирањето на Vulkan, што го олеснува користењето за развивачот.

Lve\_Texture.cpp->

Тој е одговорен за вчитување на датотека со слика и создавање на потребните Vulkan објекти за да се користи како текстура.

Конструкторот зема пример од класата LveDevice, која го претставува логичкиот уред Vulkan и патеката на датотеката на датотеката со слики што треба да се вчита. Датотеката со слика е вчитана со помош на библиотеката stb\_image и нејзините податоци се складираат во тампон за фаза. Потоа, тампонот за стадиум се користи за копирање на податоците од сликата на објект на слика Vulkan, кој потоа се префрла на соодветниот распоред за земање примероци во шејдери. Mipmaps се генерираат за сликата и се создава објект за семплер за примерок од текстурата во шејдери. Конечно, се креира приказ на слика за да се обезбеди приказ на текстурата за графичкиот цевковод.

Lve\_texture.hpp->

Текстура“. Класата се користи за ракување со текстури во апликација базирана на Vulkan. Класата има неколку функции од јавен член и неколку функции на приватни членови и променливи членови.

Класата има конструктор кој зема пример од класата „LveDevice“ и низа од патеката на датотеката како аргументи. Конструкторот е одговорен за вчитување на текстурата од наведената патека на датотеката.

Класата има деструктор за да ги исчисти сите ресурси што биле создадени за време на животниот век на објектот.

Класата има неколку функции на јавниот член за да се добие семплерот, приказот на сликата и распоредот на сликата на текстурата.

Класата има неколку функции на приватни членови, вклучително и „transitionImageLayout“, кој се користи за транзиција на распоредот на сликата од еден формат во друг, и „generateMipmaps“, кој се користи за генерирање mipmaps за текстурата.

Класата, исто така, има неколку приватни променливи членови, вклучувајќи ја ширината и висината на текстурата, бројот на нивоа на mip, уредот со кој е поврзана текстурата, меморијата на сликата и сликата, приказот на сликата и семплерот, форматот на сликата и распоредот на сликата.

Lve\_window->

Овој код дефинира класа LveWindow што создава прозорец GLFW со наведената ширина, висина и име. Класата го иницијализира прозорецот повикувајќи се на initWindow и создава површина на прозорецот за наведениот примерок на Vulkan. Знамето framebufferResized е поставено на true секогаш кога ќе се промени големината на прозорецот. Прозорецот има функција за повратен повик framebufferResizeCallback која се активира кога ќе се промени големината на прозорецот. Функцијата го поставува знамето framebufferResized и ги ажурира ширината и висината на прозорецот.

Main.cpp->

Ова е главна функција за C++ програма. Програмата создава примерок од класата FirstApp и го повикува нејзиниот метод на извршување. Доколку се фрли исклучок за време на извршувањето на програмата, тој ќе биде фатен и пораката за грешка ќе се отпечати на стандардниот проток на грешка. Конечно, функцијата враќа EXIT\_SUCCESS или EXIT\_FAILURE во зависност од тоа дали програмата работи успешно или не.

RAINBOW.cpp->cista zadaca test proba od studentot

Point\_light.vertx->

Ова е шејдер на теме што ги пресметува позициите на 6 темиња на квад на екран-простор, врз основа на дадените поместувања.

Шејдерот ги зема влезните податоци од униформните баферски објекти (UBO) и константите на туркање. „GlobalUbo“ UBO обезбедува податоци за информации за преглед, проекција и точка светло, додека константата „Push“ притискање обезбедува податоци за положбата, бојата и радиусот на објектот. Полињата „позиција“ и „боја“ во константата на притискање ги опишуваат атрибутите на објектот, а полето „радиус“ се користи за пресметување на големината на објектот.

Во главната функција, поместувањата на 6-те темиња се поставени врз основа на низата „OFFSETS“. Насоките на камерата десно и горе се пресметуваат со помош на матрицата „view“ од UBO. Потоа, светската позиција на секое теме се пресметува со додавање на радиусот на објектот, десната насока на камерата и насоката на камерата нагоре на положбата на објектот. Конечно, позицијата на екранот на секое теме се пресметува со множење на матриците на проекцијата и погледот со светската позиција. Резултатот е зачуван во променливата "gl\_Position", која е позицијата на темето во просторот за клип.

Излезната променлива „fragOffset“ го складира поместувањето на секое теме за употреба во шејдерот на фрагменти.

.point\_lightf.frag

Ова е засенчувач на фрагменти што ја зазема положбата и бојата на изворот на светлина, претставена со униформата „притискање“ и ја пресметува конечната боја на секој пиксел во фрагментот што се прикажува.

Позицијата на изворот на светлина се одредува врз основа на неговиот радиус и поместувањето на фрагментот од центарот на изворот на светлина. Бојата на изворот на светлина се користи за одредување на бојата на фрагментот. Осветленоста на фрагментот се одредува според растојанието на фрагментот од центарот на изворот на светлина. Светлината се намалува како што се зголемува растојанието, користејќи ја функцијата косинус. Бојата на фрагментот е резултат на мешање на бојата на изворот на светлина со неутрална боја (0,5, 0,5, 0,5), а интензитетот на бојата се одредува и со косинус функција. Ако растојанието на фрагментот од центарот е поголемо или еднакво на 1,0, фрагментот се отфрла.

Униформата GlobalUbo содржи матрица за проекција, матрица за гледање, матрица за обратен поглед, боја на амбиентално светло и точкести извори на светлина. Структурата PointLight претставува точкаст извор на светлина со позиција и боја. Бројот на извори на светлина е зачуван во полето "numLights".

Shader.vertx

Ова е теме шејдер во јазикот за засенчување SPIR-V. Тој е одговорен за трансформирање на податоците од темето и нивно пренесување во следната фаза во графичкиот цевковод.

Податоците за темето се пренесуваат во шејдерот како влезни променливи, како што се позицијата, бојата, нормалната и UV. Овие променливи се доделуваат на нивните локации со директивата layout(location = ...).

Шејдерот потоа користи униформен тампон објект (UBO) за пристап до глобалните податоци кои се константни низ сите темиња, како што се матриците за проекција, поглед и инверзен поглед, како и информации за точките светла во сцената.

Дополнително, шејдерот користи константа на притискање за да пристапи до податоци кои се единствени за секое теме, како што се моделот и нормалните матрици.

Потоа, шејдерот ја пресметува конечната позиција на темето во светскиот простор со негово трансформирање користејќи ги матриците на моделот и погледот. Исто така, го пресметува нормалниот вектор во светскиот простор и го пренесува, заедно со информациите за позицијата и бојата, во следната фаза како излезни променливи.

Конечно, УВ координатите се пренесуваат директно во следната фаза како излез, за да се користат за мапирање на текстурата.

Shader.frag

Овој шејдер имплементира основен модел на засенчување Phong за пресметување на дифузно и спекуларно осветлување за мрежа со текстура.

Влезот во овој шејдер е резултат на темето шејдер, кој вклучува:

fragColor: бојата на темето на решетката

fragPosWorld: позицијата на темето во светскиот простор

fragNormalWorld: нормалниот вектор на темето во светскиот простор

fragUV: текстура координати за темето

Униформниот тампон ubo ги содржи глобалните податоци за осветлување и камера, вклучувајќи:

проекција: матрицата на проекцијата

поглед: матрицата за преглед

invView: инверзна на матрицата за преглед

ambientLightColor: бојата и интензитетот на амбиенталната светлина

pointLights: низа точки светла, при што секое светло има позиција и боја/интензитет

numLights: бројот на светла во сцената

Сликата на семплерот се користи за примерок на текстурата на мрежата.

Константното притискање содржи две матрици:

modelMatrix: модел матрица на решетката

normalMatrix: нормалната матрица на решетката

Во главната функција се пресметува дифузното и спекуларното осветлување за фрагментот. Дифузното осветлување е збир на амбиенталната светлина и придонесите од секое од точките светла, каде што интензитетот на светлината се множи со косинус на аголот помеѓу насоката на светлината и нормалната површина. Спекуларното осветлување се пресметува со помош на моделот Blinn-Phong, каде што интензитетот на нагласувањето е пропорционален на косинус подигнат до моќта на аголот помеѓу векторот на полуаголник и нормалната површина. Конечната боја на фрагментот е збирот на дифузното и спекуларното осветлување, помножено со бојата на текстурата. Излезната боја е зачувана во OutColor.

Lve\_Utilis.hpp

Овој код дефинира функционален шаблон hashCombine во именскиот простор lve. Функцијата зема почетна вредност на семето, семе и променлив број на аргументи, v и одмор.

Функцијата ја користи хаш-функцијата на стандардната библиотека C++ за да го хашира првиот аргумент, v, и го комбинира со семената вредност користејќи операции на битови. Вредноста на семето се ажурира по секој повик до hashCombine. Ова овозможува функцијата да се повикува рекурзивно со преостанатите аргументи во остатокот од пакетот, овозможувајќи ѝ да ги комбинира хашовите од повеќе вредности во една вредност на семето.

Забележете дека оваа имплементација на hashCombine се заснова на функцијата boost::hash\_combine, која е широко користена во C++ за комбинирање хашови со повеќе вредности.

Resource.h

Овој код е датотека за дефиниција на ресурси за проект на Microsoft Visual C++. Целта на оваа датотека е да ги одреди стандардните вредности за различни ресурси што може да се користат во проектот, како што се слики, звуци или други видови медиуми.

Макрото \_APS\_NEXT\_RESOURCE\_VALUE ја дефинира следната достапна вредност што ќе се користи за идентификатор на ресурси.

Макрото \_APS\_NEXT\_COMMAND\_VALUE ја дефинира следната достапна вредност што ќе се користи за идентификатор на наредби, што се користи за поврзување на ставки од менито, копчиња или други елементи на корисничкиот интерфејс за одредена команда што треба да се изврши кога ќе се избере ставката.

Макрото \_APS\_NEXT\_CONTROL\_VALUE ја дефинира следната достапна вредност што ќе се користи за контролен идентификатор, кој се користи за идентификување на одредена контрола во корисничкиот интерфејс, како што е копче, поле за текст или поле за листа.

Макрото \_APS\_NEXT\_SYMED\_VALUE ја дефинира следната достапна вредност што треба да се користи за идентификатор на симбол, кој се користи за да се идентификува одреден симбол или ресурс во проектот.