**Демонстрација на Physically Based Rendering (PBR) во OpenGL**

Целта на овој проект е да се демонстрира PBR во OpenGL користејќи GLSL. Одделните commits во git се претставени подолу:

1. Initial (basic) setup
2. Basic window setup - Render loop - Brown background
3. Vertex & Fragment shaders added (String source)
4. Added VAO & VBO | Draw ornage triangle for testing purposes
5. Add EBO | Draw orange square using EBO
6. Add Shader class
7. Texture setup for future use
8. Added camera class, model, view, and projection matricies
9. Basic lighting setup | Texture setup deleted
10. Ambient, diffuse, and specualr ligthing added
11. Lighting materials setup (Material and Light structs)
12. Framebuffer lesson, postprocessing effects, and kernels
13. Cubemap (skybox) test
14. PBR trial. Red ball in center with moving white light source. Ar + CTRL + SHIFT to move light source. M, N K, L to adjust metallic and r parameters
15. IBL Diffuse Irradiance trial (UNSUCCESSFUL)
16. Diffuse irradiance and specular IBL (SUCCESSFUL)
17. 2 'rooms' - One with PBR - Other with basic ligside room

Ќе ги објасниме сите оддлено.

**Initial (basic) setup**

Во овој дел ги направивме основните поставувања на соодетните библиотеки – имено glfw, glm, и stbi библиотеките. Исто така ги поставивме модовите на оперирање на opengl – верзија, core profile, итн.

**Basic window setup – Render loop – Brown background**

Креираме објект window во кој понатаму ќе ги извршуаме сите операции.  
Проверуваме дали успешно е креиран. Испишуваме соодветна порака при неуспешна операција. Проверуваме дали GLAD е функционална и ако не – испишуваме порака на екран. Креираме while loop каде ќе се извршуваат сите операции за исцртување на екран. Ќе се искочи од циклусот при копчето ESC. Креираме функција processInput која ќе процесира информации од корисникот од тастатура. Ја поикуваме при секој frame.   
Ја поставуваме позадината со кафена боја.

**Vertex & Fragment shaders added (string src)**

Додадовме изворен код за вертекс и фрагмент шејдерите (како string во главната програма). Првиот само ги прима координатите на точките за исцртување и ги поставува соодветно со gl\_Position. Вториот само ја поставува бојата за исцртување на одредена боја преку fragColor. Креираме низа од точки преку float array. Ова се точки на триаголник. Креираме vertex buffer object и го поврзуваме со нашите точки. Ги компајлираме вертекс и фрагмент шејдерите и проверуваме дали е успешна таа операција. Ако не, испишуваме соодветна порака на екран.

**Added VAO & VBO | Draw ornage triangle for testing purposes**

Креиран Vertex Array Object со кој ќе се зачуваат сите податоци потребни за цртање на триаголникот. Триаголникот има темиња со координати: (0.5, 0.0), (-0.5, 0.0), (0.0, 0.5) – 2D триаголник. Креиран е и шејдер кој се состои од претходно напишаните вертекс и фрагмент шејдери. Направена е проверка за дали успешно ќе се поврзат напишаните шејдери (linking check). Ако има проблем се испишува порака на екран. Поставено е glPolygonMode на FRONT\_AND\_BACK и FILL – од двете страни ќе се гледа нацртаната површина и ќе биде пополнета површината наместо само да гледаме исцртани линии.

**Add EBO | Draw orange square using EBO**

Додаден е Element Buffer Object заедно со низа од индекси за темињата. Со нив ќе може да исцртаме 2D правоаголник на екранот со само 4 специфицирани темиња. Во нормални услови не би можеле да го направиме ова – ќе ни бидат потребни вкупно 6 темиња, односно 3 за првиот триаголник и 3 за вториот. Вака само кажуваме кој темиња сакаме да ги групираме заедно и ја повикуваме ф-јата glDrawElements наместо glDrawArrays.

**Add Shader Class**

За да биде почитлив кодот и за да може понатаму да го реискористиме кодот напишан за нашата шејдер програма, правиме посебна Shader класа. Оваа класа ќе содржи готов конструктор за иницијализација каде ќе се направи проверка на GLSL кодот и linking на самата програма, за уклучуање на програмата (use), и за поставување на различни униформи (uniforms).

**Textre setup for future use**

На екран се појаува правоаголник со текстура на едната страна на дрвено-метална кутија. Секој ќош е делумно обоен со различна боја. Ова е постигнато со помош на 2D текстура и библиотеката stb\_image. Поставено е да има min filter = linear-mipmap-linear и mag filter = linear. Ова е прави ако треба да се покаже текстурата во намалена / зголемена форма. Текстурата е всупшност .png слика. Се проверува успешноста на прикачувањето на текстурата на програмата и соодветно се испишува порака на екран ако е неуспешна операцијата.

**Added camera class, model, view, and projection matricies**

Повторно се појавува на екран истиот правоаголник, но сега можеме со помош на тастатурата и глувчето да се движиме низ просторот. Ова е овозможено со помош на новододадената класа Camera која содржи методи ProcessKeyboard – за промена на полобата на камерата, ProcessMouseMovement – за промена на ориентацијата на камерата, ProcessMouseScroll – за промена на zoom-от со помош на тркалцето од глувчето, покрај други споредни помошни методи (конструктор, GetViewMatrix, updateCameraVectors). Во main() додадени со нови променливи (lastX, lastY, deltaTime, lestFrame, итн.) кои ги користиме за пресметка на движењето на камерата. Додадени се и нови матрици (model, projection, view матрици). Model matrix се користи за одредување на положбата на предметите во просторот, view matrix за одредување на положбата на предметот во однос на камерата, а projection matrix се користи за одредуање на положбата на пикселот на екранот. Други промени: проширување на processInput, нови функции mouse\_callback и scroll\_callback.

**Basic lighting setup | Texture setup deleted**

На екран се појавува портокалова коцка и помала бела коцка која ќе служи како осветлување понатаму. За сега сите страни од коцките се подеднакво “осветлени” (нема дифузно и спекуларно осветлување). Сега користиме 2 шејдери – 1 за осетлените објекти, и уште 1 за самото осетлување (сијалица). За сега кај првиот шејдер само се пресметуваат координатите (вертекс шејдер) и се користи униформа objectColor за портокалова боја. Кај вториот шејдер е истото само што наместо имаме бела lightColor. При исцртување на 2та коцка се променува model матрицата – се прави транслација и скалирање на коцката. Се користи уште еден дополнителен VAO за цртање на осветлуањето.

**Ambient, diffuse, and specualr ligthing added**

На екран се појавува истата сцена разликата што сега имаме дифузно и спекуларно осетлуање. Тоа значи дека сега секоја различна страна е осветлена со различен интензитет во зависност од ориентацијата на страната во однос на сијалицата (дифузно), и сега во зависност од положбата на камерата, фрагментот, и осветлувањето, може да забележиме бела рефлекија (спекуларно). Додадени се нови вредности во низата verteces, поточно додадени се информации за нормалите на страните кои се користат во пресметките на дифузното и спекларното осветлување. Исто така имаме амбиентално осветлување. Во фрагмент шејдерот едноставно се собираат сите овие 3 компоненти од осветлувањето. Кај сијалицата едноставно ја исцртуваме целосно бела без диф. и спек. осветлување.

**Lighting materials setup (Material and Light structs)**

На екран се појавува истата сцена со само еден додаток – овозможено е промена на позицијата на сијалицата со помош на тастатурата. Во главниот фрагмент шејдер додадени се две структури (struct) – Material со елементи ambient, diffuse, specular, shininess (кои ќе се користат во пресметките за осветлувањето) и Light со елементи position, ambient, diffuse, и specular кои исто така ќе се користат во пресметките. Нивните вредности соодветно се поставуваат преку униформи.

**Framebuffer lesson, postprocessing effects, and kernels**

На екран се појаува една рамнина на која има две дрени кутии. Клучно во оваа сцена е што е искористен postprocessing effeect – сцената делува замаглена. Ова е постигнато со framebuffer и kernel. Во суштина откако ќе се рендерира сцената, ги презапишуваме пикселите во посебен бафер (framebuffer). Оттука имаме слобода што сакаме да правиме со овој бафер што во суштина е 2D слика. Во овој случај искористуваме kernel што може да го замислиме како 3x3 матрица каде средишниот елемент е всушност пикелот што треба да го исцртаме, а останатите елементи се околните пиксели. На овој начин, при цртање на секој поединечен пиксел ние ги зимаме предвид околните пиксели во пресметката на крајната боја на конкретниот пиксел што се исцртува. Ако го смениме kernel-от може да добиеме најразлични ефекти.

**Cubemap (skybox) test**

На екран се појавува една дрвена кутија која се чини лебди во простор каде има големо езеро одоздола, планини одоколу и ведро небо одозгора. Ова е всушност реалистична илузија која е постигната со cubemap. Тоа што навистина се случува е следното: прво се исцртува коца со димензии 1x1x1 која има фиксна позиција релативна на камерата. Тоа се прави така што имаме 6 текстури (слики) кои се поврзуваат со одредена samplerCube униформа во нашиот skybox фрагмент шејдер. Видливиот дел од коцката се црта на framebuffer со што во ефект добиваме позадина на сцената. Оттука само ја исцртуваме нашата кутија врз сцената. За ова да го постигнеме мора да специфираме glDepthfunc(GL\_LEQUAL). Така се добива ефект дека нашиот предмет се наоѓа во накаков простор, наместо во некој вакум како што беше до сега. Исто така додадени се помошни функции loadTexture и loadCubemap со кои може на едноставен начин да ги лоадираме соодветните текстури / фајлови.

**Дополнителен коментар – HDR, Gamma correction, & Attenuation**

Измеѓу претходниот и следниот commit направени се тестирања поврзани со HDR, Gamma correction, attenation. Овие commits се изоставени поради тоа што не се директно поврзани со крајната сцена што сакаме да ја прикажеме на екран. Направени се само за тестирање на овие функционалности. Следи краток опис на што било постигнато во овие необјевени commits. Прво беше направана сцена каде имавме повеќе извори на светлина и направени пресметки за High Dynamic Range lighting. Ова значи дека наспроти нормален случај каде осветлувањето на даден пиксел може да има вредност од 0.0f до 1.0f, тука имаме мапирање на пожирок спектар на можни вредности на осетлување – на пример од 0.0f до 300.0f. Ова е постигнато со некоја мапирачка функција. Потоа во фрагмент шејдерот направено е да се земе предвид далечината на осетлениот фрагмент од изворот на светлина. Искористена е инверзна функција. Ова значи дека фрагменти кои се оддалечени од изворот на светлина ќе бидат потамни од поблиски фрагменти. На крај воведовме gamma correction. Поради фактот дека има разлика меѓу физичка јачина на светлина (бројот на фотони присутен) и како ние ја перцепираме јачината на светлината (нелинеарно), голем дел од мониторите прават промени при претстаување на пикселите. Ова во нашиот случај не ни е пожелно, па сакаме да го неутрализираме овој ефект. Ова е постигнато со коренуање на крајната вредност на пикселот во фрагмент шејдерот.

**PBR trial**

На екран се појавува црвено топче заедно со помало бело топче кое ни служи како сијалица. Црвенето топче е осветлено со цел да постигнеме PBR (Physically based rendering). Тоа значи дека сакаме да симулираме реалистично осветлвање. За тоа да се постигне треба неколку работи да запазиме: да биде основано на карактеристиките на осветлената површина (мазен материјал да сјае повеќе од рапав материјал на пример), да се земе предвид конзеварција на енергијата на осветлувањето, и да биде базирано на BRDF (biderectional reflective distribution function). Сцената може да ја манипулираме со помош на тастатурата – со W, A, S, D се движиме низ просторот, со стрелките, CTRL и SHIFT ја менуваме положбата на камерата, а со K, L, M, N ги променуваме карактеристиките на предметот (колку е метална / диелектрична и колку е рапава / мазна површината). Додадовме и нова функција renderSphere() со која може на лесен и едноставен начин да исцртаме топка. Главните пресметки во овој commit се случуваат во фрагмент шејдерот. Користиме повеќе униформи: albedo (што во овој случај е всушност бојата на предметот, а во други случаи може да се разбере како главната текстура на предметот), metallic (одредува колку е метална површината на предметот, односно колку рефлектира дифузна светлина), roughness (колку е рапава површината, односно колку ја расфрла светлината на сите страни и колку има геометриска обструкција / сенчење), ao (колку предметот е осетлив на амбиенталната светлина). Дополнително во самиот фрагмент шејдер користиме готови методи: DistributionGGX, GeometrySchlickGGX, GeometrySmith, fresnelSchlick кои се користат за пресметки на работи како fresnel effect (поголема рефлексија при тапи агли на инциденција), влијанието на рапавоста на површината, и сл.

**Diffuse irradiance and speclar IBL**

На екран се појаува потворно црвено топче и бело светло над камерата, но овој пат дополнително имаме поставено и cubemap така што личи како да сме надвор до некој канал каде што сонцето свети и има ведро небо. Тоа што е битно за овој commit е тоа што сега дополнително може да забележиме рефлексија на нашето црвено топче од светлото, *но и од самата околина*. На овој начин топчето личи како да е метално. Успеавме да направиме image based lighting (IBL). За околината искористивме .hdr фајл наместо обичен cubemap составен од повеќе .jpeg / .png фалјови. Тоа е направено за да можема полесно да имаме HDR осветлување. Самото рефлектирање на околината е постигнато на следниот начин: бидејќи не би било изводливо да процесираме за секој поединечен фрагмент рефлектираната светлина in real time, ќе искористиме претпроцесирање. Тоа значи дека од .hdr мапата ние ќе изведеме готови environment мапи за дифузното и спекуларното осветлување посебно. Ќе ги зачуваме овие мапи и на крај, при исцртување на конкретниот фрагмент, само ќе провериме која е соодветната боја што треба да ја прикажеме на екран. За да се постигне овој ефект потребни се доста комплексни пресметки (како на пример monte-carlo integration, importance sampling и сл.). Исто така додадени се помошни функции renderCube() и renderQuad() кои ги користиме при исцртување на cubemap-от и при коирстење на framebuffer-от.

**2 “rooms” - One with PBR - Other with basic lighting inside room**

Сега на екран се појавува истата сцена со тоа што е додадена функционалност за движење на на светлото со помош на тастатурата. Исто така ако се кликне копчето F на тастатурата целосно се променува сцената – се наоѓа камерата во темно сива соба каде наместо црвена топка, сега гледаме црвена коцка која постојано ротира. Светлото што го имавме е пристно и во оваа сцена. Клучната разлика е што сега не користиме PBR осветлување, туку обично ambient + diffuse + specular осветлување. оваа промена на сцената е постигната со тоа што кога ќе се кликне F се променува boolean променлива switchScene. Имаме сега додадено нов (обичен) шејдер каде се применети истите техники од *Ambient, diffuse, and specualr ligthing added* commit-от. Кога ќе се промени boolean променливата, во while циклусот имаме поставено if проверка и соодветно ја цртаме соодветната сцена со соодветните шејдери.