Room Planner

Факултет за Информатички Науки и Компјутерско Инженерство

Ментор:

Проф Д-р Бобан Јоксимоски

Изработил:

Андреј Андоновски – 211090

Марино Јакимоски – 211017

Апстракт

Проектот Room Planner кој е изработен во OpenGL C++ е проект кој им овозможува на корисниците да креираат и дизајнираат различни типови на просотрии и амбиенти. Овај проект се користи за виртуелно планирање на простории во домови, канцеларии и многу други различни објекти.Корисниците бираат различни димензии, стилови и многу објекти за да го креираат и прегледаат својот простор пред да го изградат во реалност.Односно се користи како алатка за дизајнерите на ентериер, архитектите, давајќи им многу можности да експериментираат и да ги реализираат нивните визии во виртуелна област.

Клучни Зборови

Room planner, виртуелно дизајнирање, 3D приказ, осветлување, текстури, дизајн на интериор, креативни можности.

Интро

Во денешно време, потребата за прецизно и креативно планирање на интериорните простории постојано расте. Традиционалните методи за планирање не се доволно сигурни и не претставуваат доволно реалистични сценарија. Во овој контекст, 3D планирањето на соби е револуционарен напредок.Оваа напредна технологија овозможува создавање на тродимензионални репрезентации на собите и просториите, што овозможува потполно разгледување на различни дизајн концепти и распореди на мебел. Преку користење на моќни софтверски алатки како OpenGL, корисниците можат лесно да ги организираат објектите и да визуелизираат промени во дизајнот во реално време.Ова иновативно планирање на соби не само што засилува креативноста, туку и обезбедува практични предности. Дизајнерите можат многу поедноставно да соработуваат со своите клиентите. Домаќините имаат можност да експериментираат со различни распореди пред да извршат физички промени, додека архитектите можат да симулираат просторни распореди за оптимизација на функционалноста и естетиката.3D планирањето на соби им овозможува на корисниците да ги визуелизираат своите креативни идеи и да ги реализираат своите визии за совршените животни простории во виртуелна област.

Setup

1) **Подготовка на OpenGL околина**:
   Ги поставуваме потребните библиотеки за OpenGL. Ова вклучува инсталирање на потребни пакети за OpenGL и подесување на околината за програмирање во C++.

2) **Иницијализација на OpenGL прозорец со користење на GLFW**:

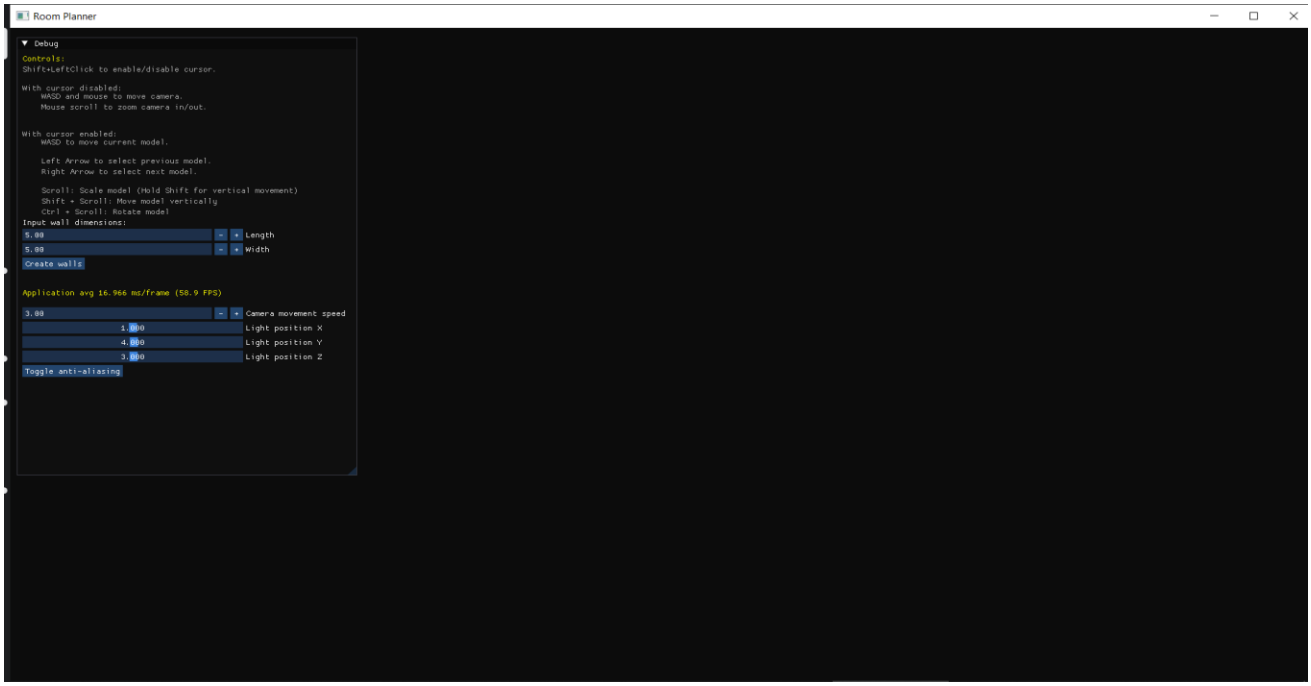
a. Пред да го создадеме прозорецот, треба да се иницијализира GLFW. Ова се прави користејќи ја функцијата glfwInit(). Ако иницијализацијата е успешна, GLFW е спремен да создава прозорци.

b. **Создавање на прозорец:**
   Кодот создава прозорец користејќи ја функцијата glfwCreateWindow(), која прима параметри за ширината, висината, насловот. Оваа функција враќа покажувач GLFWwindow* до создадениот прозорец.

c. **Конфигурација на контекстот и баферот:**
   По создавањето на прозорецот, кодот го поставува тековниот OpenGL контекст на новиот создаден прозорец користејќи ја функцијата glfwMakeContextCurrent(window). Исто така, се наведува функцијата за повик на големината на баферот користејќи ја glfwSetFramebufferSizeCallback(window, framebuffer_size_callback). Оваа функција ќе биде повикана секогаш кога големината на прозорецот ќе се промени, дозволувајќи соодветна промена на приказот.

d. **Конфигурација на OpenGL контекстот:**
   Кодот го поставува прозорецот и параметрите користејќи ги функциите glViewport() и glEnable().

e. **Main Loop:**
   На крајот, кодот влегува во главниот loop каде што проверува настани (на пример, настани за затворање на прозорецот, притискање на копчиња) користејќи ги функциите glfwWindowShouldClose(window) и glfwPollEvents().



3) **Дефинирање на собата и рендерирање собата**:
Собата ја дефинираме користејќи вертекси, агли и површини. Ова го правиме со наведување на координатите на аглите на собата и потоа ги поврзуваме за да се создадат површини.Имплементираме функции за рендерирање на собата. Ова вклучува подесување на вертексни бафери, објекти на вертексни низи и шејдери (вертексни и фрагментни шејдери) за правилно рендерирање.

   a. **Создавање на Зидови**:
   Зидовите се создаваат во ImGui прозорецот каде внесуваме рандом димензии за зидовите (должина и широчина). Кога ќе кликнеме на "Create walls", со функцијата **ImGui::Button** ги креираме зидовите со одбраните димензии.Врвовите на зидовите се дефинирани во векторот **wallVertices**, кој ги содржи координатите и нормалите за секој врв за различни зидови (преден, заден, лев, десен и под). Овие врвови потоа се зачувани во баферите (**VAO_walls**, **VBO_walls**) за да се користат за рендерирање.

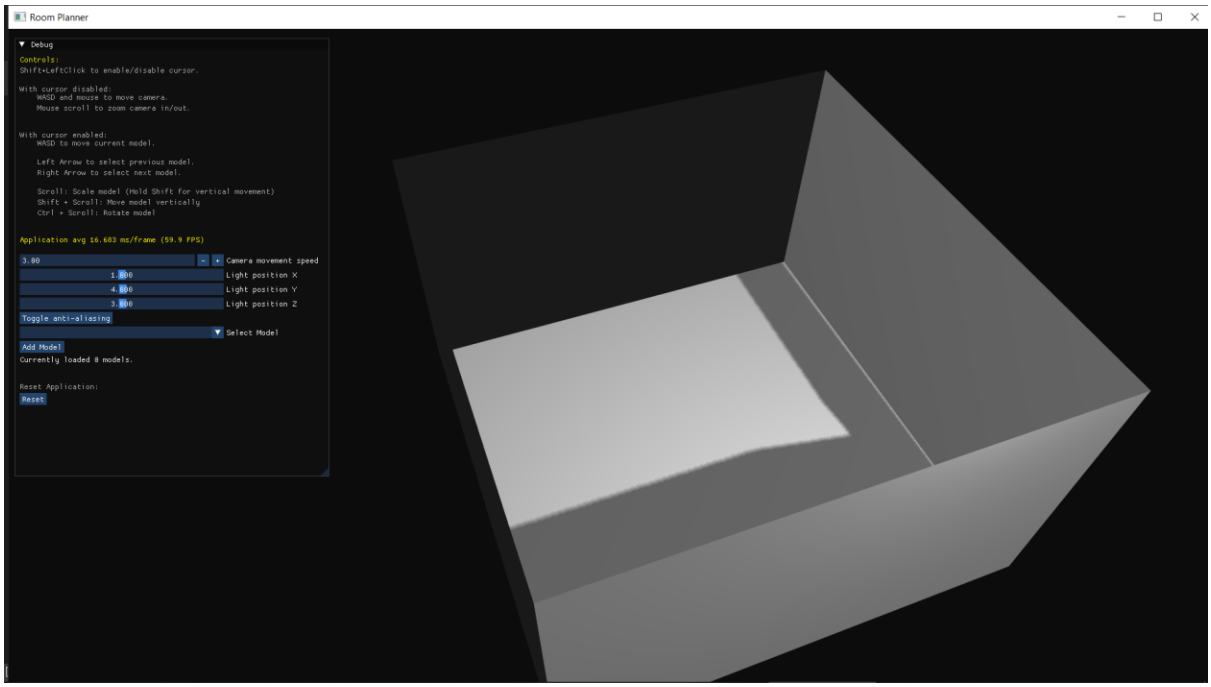
   b. **Рендерирање на Зидови**:
   Зидовите се рендерирани со помош на OpenGL во главниот рендеринг loop. Во продолжение е објаснето како се врши процесот:

a. **Рендерирање на Depth Map**: Пред да се рендерираат зидовите, се генерира Depth Map од перспектива на изворот на светлина со помош на посебен шејдер (**simpleDepthShader**). Овој чекор е клучен за мапирање на сенката за да се симулираат реалистични ефекти на осветлувањето.

   b. **Рендерирање со Нормален Шејдер**: По генерирањето на Depth Map, сцената се рендерира повторно со користење на **modelShader** и **wallShader**. Овие шејдери се грижат за осветлувањето, сенките и трансформацијата на перспективата на зидовите и другите модели во сцената.

c. **За зидовите**: Се користи **wallShader** со соодветни униформи поставени за матриците за проекција, погледот, моделите, позицијата на светлината, бои на светлината и други параметри потребни за осветлувањето.

d. **ImGui Рендерирање**: На крајот, ImGui рендерира своите компоненти, вклучувајќи ги било кои UI елементи како текст, копчиња, слајдери итн., над рендерираната сцена во OpenGL.



4) **Имплементирање на контроли на камерата и движење на модели**:

Имплементираме контроли на камерата за дозвола на корисникот да навигира околу собата. Ова вклучува подесување на модел на камерата, дефинирање на нејзината позиција и ориентација, и имплементација на функции за контролирање на нејзиното движење (на пример, движење напред, назад, ротирање).

Escape key: Доколку е притиснато ESC копчето, функцијата поставува флаг за затворање на прозорецот на GLFW.

Обработка на Режим на Камера (!imguiMode): Кога не е во ImGui режим (camera mode), функцијата проверува за специфични притискања на копчиња за движење на камерата користејќи ја функцијата camera.ProcessKeyboard.

W: Помести камера напред.

S: Помести камера назад.

A: Помести камера лево.

D: Помести камера десно.

Обработка на Режим на ImGui (imguiMode): Кога е во ImGui режим, функцијата се справува со транслацијата на моделот наместо со движење на камерата.

W: Транслирај го тековниот модел наназад по Z-оската.

S: Транслирај го тековниот модел напред по Z-оската.

A: Транслирај го тековниот модел лево по X-оската.

D: Транслирај го тековниот модел десно по X-оската.

Во суштина, зависно од тоа дали апликацијата е во режим на камера или во режим на ImGui, функцијата processInput се справува со различни видови на влез. Во режим на камера, таа контролира движење на камерата, додека во режим на ImGui, се справува со транслацијата на тековниот модел.

5) **Текстурно мапирање**:

Текстури на зидовите, подот и таванот на собата за да им дадете реалистичен изглед. Ова вклучува вчитување на текстурни слики и мапирање на нив врз геометријата на собата.(ова не стигнавме да го имплементираме)

6) **Корисничка интеракција**:

Имплементирајте корисничка интеракција за дозвола на корисникот да додава мебел и објекти во собата. Ова може да вклучува mouse input за избирање и поставување на објекти, како и тастатурен влез за прилагодување на својствата како големина и ротација.

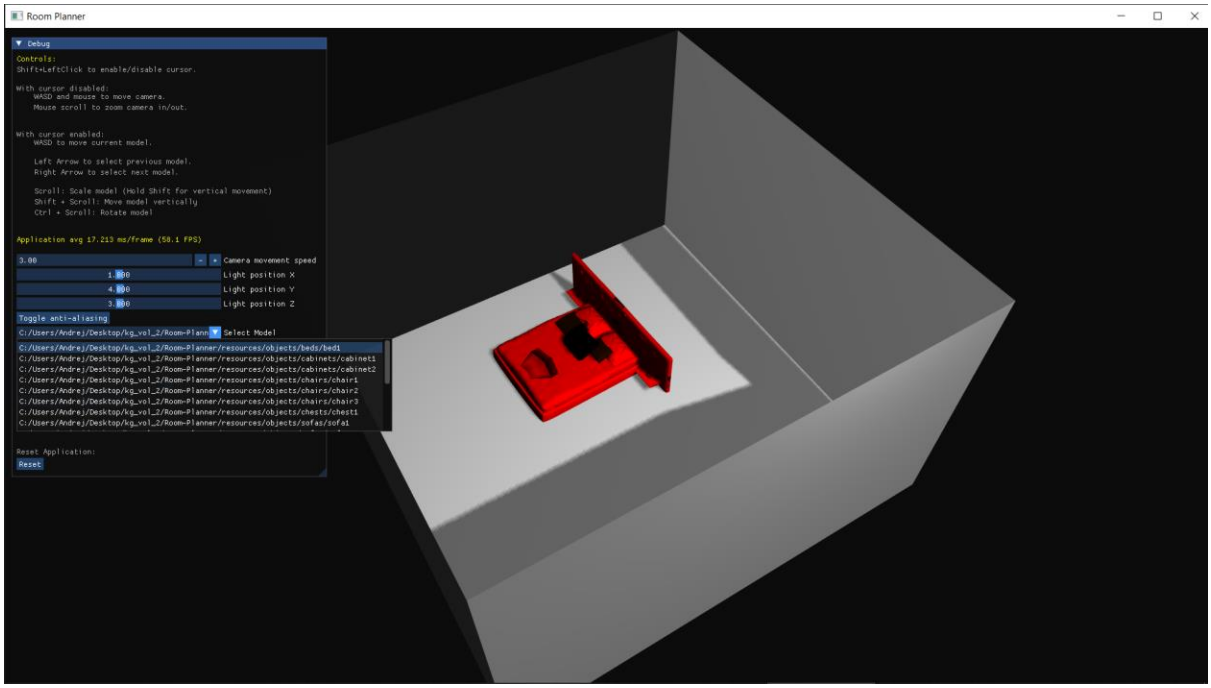
Како се пополнува drop down листата:

Овај дел започнува со условна проверка (**if (walls_created)**) за да се одреди дали треба да се изврши кодот што следи. Во рамките на овај услов, се иницијализира комбинирана кутија (combo box) користејќи го **BeginCombo** методот на ImGui, овозможувајќи на корисникот да избере модел од листата.

Кодот итерира низ колекција на достапни модели (**availableModels**) и ги пополнува кутиите за комбинирање со нивните имиња. Секое име на моделот се прикажува како избирачки предмет во кутијата за комбинирање. Кога корисникот избира модел, неговото име се зачувува во променливата **currentItem**.

Откако кутијата за комбинирање е пополнета, кодот ја зема листата на ".obj" датотеки од специфицирана директорија користејќи го **getFilesInDirectory** методот. Овие датотеки претставуваат достапни 3D модели. За секоја датотека се создава и иницијализира **ModelData** објект со својства како патеката на датотеката и други параметри.

Објектите **ModelData** се додаваат во **availableModels** векторот, правејќи ги достапни за подоцна користење во програмата



7) **Поставување на објекти**:

Дефинираме геометрија на мебел и објекти кои може да се постават во собата. Ова вклучува креирање на модели или внесување на 3D модели од надворешни датотеки (на пример, OBJ, FBX) и нивно рендерирање во сцената.

Рендерирање на моделите избрани во drop down листата:

Во rendering loop, итерираме низ векторот models за да се рендерира секој модел. За секој модел, ги извлекуваме неговите параметри за трансформација (транслирање, ротација, скела) и ги применуваме на неговата модел матрица. Потоа ја пренесуваме оваа модел матрица на шејдерскиот програм (modelShader) користејќи setMat4("model", modelMatrix).Ја повикуваме методата Draw на моделот за да се рендерира користејќи го model shader

Процесот вклучува избирање на модел од drop down листа, додавање на тој модел во сцената, и потоа исцртување на моделот заедно со сите други модели што веќе се вчитани во сцената, користејќи соодветни матрици на трансформација.

8) **Детекција на судирање**:

Имплементираме детекција на судирање за да спречиме објекти да се пресекуваат меѓусебно или со геометријата на собата. Ова гарантира дека објектите поставени во собата се однесуваат реалистично и не се затнуваат низ зидови или други објекти.

Функцијата **calculateAABB** е дизајнирана за да одреди AABB кој строго ги опкружува колекциите meshes претставени како множества на врвови во тродимензионалниот простор. AABB служи како поедноставена репрезентација на просторното проширение на meshes, што помага во откривањето на колизии, ограничувањето на видливоста и алгоритмите за просторно партиционирање.

Преглед на имплементацијата:

1. **Иницијализација:** Функцијата иницијализира два вектори, **minCorner** и **maxCorner**, за да ги претстави најмалиот и најголемиот агол на AABB. Овие вектори првично се поставени на максималните и минималните можни вредности на плуваечки точки.
2. **Поминување на meshes:** Функцијата поминува низ секоја мрежа во влезниот вектор **meshes**.
3. **Поминување на врвовите:** За секој mesh, функцијата поминува низ нејзините врвови.
4. **Ажурирање на аголите:** За секој врв, функцијата ги ажурира векторите **minCorner** и **maxCorner** врз основа на позицијата на врвот. Осигурани се дека AABB ги заобиколува сите врвови на мрежите со прилагодување на неговите аголи за да се прилагодат минималните и максималните координати по секоја оска.
5. **Изградба на AABB:** Штом сите врвови на сите мрежи се процесираат, функцијата го изградува и враќа AABB користејќи ги пресметаните **minCorner** и **maxCorner**

(ИМПЛЕМЕНТИРАНО АМА НЕ Е ФУНКЦИОНАЛНО)

9) **Осветлување**:

Имплементираме осветлување за да се осветли собата и објектите реалистично. Ова вклучува подесување на извори на светлина (на пример, точкови светла, директивни светла) и шејдери за да се пресметуваат ефектите на осветлување како дифузни и спекуларни рефлексии.

1. **Извори на светлина**:
   - Имаме дефинирано два извори на светлина: **lightPos1** и **lightPos2**. Овие се променливи од тип **glm::vec3** кои ги претставуваат позициите на изворите на светлина во 3D просторот.
   - За секој извор на светлина е дефинирана соодветна боја со **lightColor1** и **lightColor2**. Овие се променливи од тип **glm::vec3** кои ги претставуваат RGB компонентите на светлината.
2. **Својства на осветлувањето**:
   - Секој извор на светлина има различни својства кои влијаат на тоа како ќе осветлува објекти во сцената. Овие својства вклучуваат:
     - **Амбиентна сила (ambientStrength1, ambientStrength2):** Определува силата на амбиентната светлина, која е светлина која се распрскува и одразува рамномерно во сите насоки од објектите.
     - **Спекуларна сила (specularStrength1, specularStrength2):** Определува силата на спекуларните светлосни моменти, кои се светлински точки што се појавуваат на светлите површини кога се осветлени од извор на светлина.
     - **Блескавост (shininess1, shininess2):** Параметар кој влијае на големината и интензитетот на спекуларните моменти. Поголемите вредности за

блескавост резултираат со помали и поинтензивни моменти.

3. **Пренесување на својствата на светлината на шејдерите**:
    - Својствата на секој извор на светлина се пренесуваат на шејдерите одговорни за рендерирањето на сцената. Ова се прави со поставување на униформни променливи во шејдерите со вредностите на својствата на светлината.
    - Соодветните униформни променливи во шејдерите вклучуваат:
        - **lightPos1**, **lightPos2**: Позиции на изворите на светлина.
        - **lightColor1**, **lightColor2**: Бои на изворите на светлина.
        - **ambientStrength1**, **ambientStrength2**: Амбиентна сила на изворите на светлина.
        - **specularStrength1**, **specularStrength2**: Спекуларна сила на изворите на светлина.
        - **shininess1**, **shininess2**: Блескавост на изворите на светлина.
4. **Имплементација на шејдерите**:
    - Шејдерите (**modelShader** и **wallShader**) ги користат пренесените својства на светлината за пресметување на светлосните ефекти како дифузно осветлување, спекуларни моменти и сенки за објекти во сцената.

10) Mouse и scroll настани:

- **mouse_callback функцијата:**
  - Параметрите **window**, **xposIn** и **yposIn** претставуваат компоненти на овој метод. **window** е покажувач кон прозорецот на GLFW каде што се прима влезот од глувчето, додека **xposIn** и **yposIn** претставуваат моменталната позиција на курсорот во тој прозорец.

  - За да се олесни подоцнашната обработка и интеракција, функцијата ги конвертира овие вредности од двојна точност во вредности со единечна точност со помош на **static_cast<float>()**, додека резултатот се **xpos** и **ypos** како float променливи.

  - Условната наредба **if (firstMouse)** служи како чувар, обезбедувајќи дека специфични дејства се преземаат само при примање на првиот влез од глувчето. Кога ќе се активира, овој блок код ја доделува моменталната позиција на курсорот на **lastX** и **lastY**, што означува почеток на следењето на движењето на глувчето. Потоа, булеанот **firstMouse** се префрла во false, указувајќи дека првиот влез од глувлето е обработен.

  - Последната фаза вклучува обработка на пресметаните одстапувања на мишот за да се извршат промени, обично поврзани со ориентацијата на камерата. Функцијата го повикува **camera.ProcessMouseMovement()** и ги предава **xoffset** и **yoffset** како аргументи. Овој метод ги модификува ориентацијата на камерата врз основа на детектираното движење на мишот, што овозможува интерактивно навигирање или манипулација на рендерираната сцена.

- **mouse_btn_callback функцијата:**
    - Што се однесува до параметрите на оваа функција, првиот параметар (GLFWwindow* window) е покажувач кон прозорецот на кој се случила акцијата со маусот. Вториот параметар (int button) е цел број кој претставува кое копче на маусот било притиснато. Третиот параметар (int action) претставува вид на акција што се случила со копчето на маусот, дали било притиснато, пуштено или повторено. Последниот параметар (int mods) претставува битно поле кое ги претставува модификаторните копчиња како Shift, Control, Alt итн.

    - Самата функција има условна наредба која проверува дали се исполнети одредени услови пред да изврши одредени акции. На пример, ако левото копче на маусот е притиснато и додека тоа се случило, копчето Shift било исто така притиснато, тогаш се повикува функцијата changeImguiMode().

- **scroll_callback функцијата:**
    - Првиот параметар (GLFWwindow* window) е покажувач кон прозорецот на кој се случила акцијата на скролирање. Другите два параметри (double xoffset и double yoffset) се вредности кои ги претставуваат померувањата на скролот по x-оската и y-оската, соодветно. Позитивните вредности на yoffset обично указуваат нагоре scroll, додека негативните вредности покажуваат надолу scroll.

    - Оваа функција прво проверува дали е активиран режимот на ImGui (imguiMode) и дали постои валиден модел (currentModel.valid). Ако и двата услов се исполнети, функцијата одлучува како да реагира на скролот во зависност од копчињата притиснати за време на настанот на скролирање:

- Ако Shift копчето (GLFW_KEY_LEFT_SHIFT или GLFW_KEY_RIGHT_SHIFT) е притиснато, скролирањето се користи за вертикално транслирање на тековниот модел.
- Ако Control копчето (GLFW_KEY_LEFT_CONTROL или GLFW_KEY_RIGHT_CONTROL) е притиснато, скролирањето се користи за ротација на тековниот модел.
- Ако ниту Shift ниту Control копчињата не се притиснати, скролирањето се користи за скалирање на тековниот модел.

- Ако режимот на ImGui не е активен или нема валиден модел, скролирањето се толкува како зумирање на камерата, и се повикува функцијата ProcessMouseScroll на камерата со yoffset како фактор за зумирање.

11) Reset функција:

Прво, функцијата ги ресетира позицијата и ориентацијата на камерата. Ова се извршува со инстанцирање на нов објект од класата Camera, со поставување на почетни вредности за позиција и ориентација. Камерата се поставува да гледа кон почетната позиција (0.0f, 7.0f, 5.0f)

Потоа, се ресетираат податоците за моделите. Ова се прави со празнење на листата на модели (models), поставување на тековниот модел (currentModel) и индексот на тековниот модел (currentModelIndex) на почетни вредности, како и ресетирање на бројачот за нови модели (newModels).

Додека, сите поставки за создавање на зидови (walls_created) се ресетираат на почетната вредност за дозвола на создавање на зидови.

Исто така, се ресетира и статусот на режимот на ImGui (imguiMode), кој се користи за контрола на интеракцијата со графичкиот кориснички интерфејс.

Краен резултат