**DUŻE FUNKCJE:**

Read Texture  **!!**

Load Model **!!**

Draw Bottom !!

Draw Model !!

Draw Plants !!

Draw Rocks !!

Draw Scene !!

Initopengl **!!**

Freeopengl !!

**POJEDYNCZE FUNKCJE:**

**Read Texture:**

Funkcja **readTexture** służy do wczytywania tekstur z plików i przechowywania ich w wektorze **texs**. Oto wyjaśnienie poszczególnych kroków w tej funkcji:

1. **texs.push\_back(0);**: Dodaje wartość 0 na koniec wektora **texs**. Prawdopodobnie inicjalizuje ona miejsce w wektorze, które zostanie później zastąpione rzeczywistym indeksem tekstury.
2. **GLuint tex;**: Deklaruje zmienną **tex** jako uchwyt (handle) tekstury w OpenGL.
3. **glActiveTexture(GL\_TEXTURE0);**: Ustawia jednostkę teksturującą o indeksie 0 jako aktualną jednostkę teksturującą. Oznacza to, że kolejne operacje związane z teksturami będą dotyczyć tej konkretnej jednostki teksturującej.
4. Wczytanie obrazka z pliku przy użyciu biblioteki LodePNG i zapisanie go w wektorze **image**. Pobierane są również szerokość i wysokość obrazka.
5. **glGenTextures(1, &tex);**: Generuje jeden uchwyt tekstury w OpenGL i przypisuje go do zmiennej **tex**.
6. **glBindTexture(GL\_TEXTURE\_2D, tex);**: Powiązuje uchwyt tekstury **tex** z jednostką teksturującą **GL\_TEXTURE\_2D**. Następne operacje na teksturach będą dotyczyć tej jednostki teksturującej.
7. **glTexImage2D(...)**: Wczytuje obrazek z pamięci komputera do pamięci karty graficznej, skojarzonej z uchwytem tekstury **tex**. Używa danych z wektora **image** i informacji o szerokości, wysokości, formatie i typie pikseli obrazka.
8. **glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);** i **glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);**: Ustawia parametry filtrowania tekstur. W tym przypadku, używany jest filtr liniowy (**GL\_LINEAR**), który zapewnia płynne przejścia między pikselami tekstury przy skalowaniu.
9. **texs[tex\_i] = tex;**: Przypisuje wartość uchwytu tekstury **tex** do elementu o indeksie **tex\_i** w wektorze **texs**. W ten sposób, tekstura o indeksie **tex\_i** w **texs** jest zaktualizowana i przechowuje odpowiedni uchwyt tekstury.

Podsumowując, funkcja **readTexture** wczytuje obrazek z pliku, tworzy teksturę w pamięci karty graficznej, ustawia jej parametry, a następnie przypisuje jej uchwyt do odpowiedniego indeksu w wektorze **texs**, umożliwiając dalsze korzystanie z tekstury w innych częściach programu.

Funkcja **initOpenGLProgram** inicjalizuje program OpenGL, ustawiając różne parametry i wczytując tekstury oraz modele do pamięci GPU. Oto wyjaśnienie poszczególnych kroków w tej funkcji:

1. **initShaders();**: Inicjalizuje shadery programu. Prawdopodobnie ta funkcja przygotowuje program do korzystania z shaderów w dalszych operacjach renderowania.
2. **glClearColor(0.85, 0.85, 0.85, 1);**: Ustawia kolor tła sceny na jasnoszary.
3. **glEnable(GL\_DEPTH\_TEST);**: Włącza test głębokości (depth test), który kontroluje, czy piksel zostanie narysowany na podstawie jego odległości od kamery. Pozwala to na realizację efektu przysłaniania geometrycznego.
4. **glEnable(GL\_BLEND);** i **glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);**: Włącza mieszanie (blending) kolorów pikseli na podstawie ich przezroczystości (alpha). Używane są konkretne funkcje mieszania, w tym przypadku **GL\_SRC\_ALPHA** i **GL\_ONE\_MINUS\_SRC\_ALPHA**.
5. Konfiguracja funkcji callback dla interakcji z klawiaturą i myszą. **glfwSetKeyCallback** ustawia funkcję zwrotną (**key\_callback**), która zostanie wywołana po naciśnięciu lub zwolnieniu klawisza. **glfwSetInputMode** ustawia tryb kursora myszy na zablokowany (GLFW\_CURSOR\_DISABLED), co oznacza, że kursor myszy jest ukryty i nie może opuścić okna. **glfwSetCursorPosCallback** ustawia funkcję zwrotną (**mouse\_callback**), która zostanie wywołana po zmianie pozycji kursora myszy.
6. Wczytanie tekstur z plików za pomocą funkcji **readTexture**. Pliki z teksturami mają ścieżki, takie jak "./img/fish.png" i "./img/tank2.png". Każda z wczytanych tekstur jest przypisana do odpowiedniego indeksu w wektorze **texs**, takiego jak TEX\_FISH, TEX\_TANK, itd.
7. Wczytanie modeli z plików za pomocą funkcji **loadModel**. Pliki z modelami mają ścieżki, takie jak "models/fish.fbx" i "models/tank.fbx". Każdy z wczytanych modeli jest przypisany do odpowiedniego indeksu, takiego jak FISH, TANK, itd.
8. Tworzenie obiektów shaderów za pomocą konstruktorów klasy **ShaderProgram**. W tym przypadku, tworzone są trzy obiekty shaderów o nazwach plików "v\_water.glsl", "v\_phong.glsl", "v\_glass.glsl" dla vertex shadera i "f\_water.glsl", "f\_phong.glsl", "f\_glass.glsl" dla fragment shadera. Obiekty shaderów są przypisywane do odpowiednich zmiennych, takich jak **waterShader**, **phongShader** i **glassShader**.

Podsumowując, funkcja **initOpenGLProgram** przygotowuje program OpenGL do renderowania, wczytuje tekstury i modele, ustawia parametry renderowania (takie jak kolor tła, test głębokości, mieszanie kolorów) oraz konfiguruje interakcję z klawiaturą i myszą. Jest to część inicjalizacji programu przed rozpoczęciem właściwej pętli renderowania.

Funkcja **loadModel** służy do wczytywania modeli z plików do pamięci programu. Oto wyjaśnienie poszczególnych kroków w tej funkcji:

1. **models.push\_back(MyVertex());**: Dodaje nowy obiekt typu **MyVertex** do wektora **models**. Ten obiekt będzie przechowywał informacje o wierzchołkach, normalnych, współrzędnych tekstur i indeksach modelu.
2. **Assimp::Importer importer;**: Tworzy obiekt klasy **Assimp::Importer**, który będzie używany do importowania modelu z pliku.
3. **const aiScene\* scene = importer.ReadFile(filename, aiProcess\_Triangulate | aiProcess\_FlipUVs | aiProcess\_GenSmoothNormals);**: Wczytuje model z pliku o podanej ścieżce **filename** za pomocą obiektu **importer**. Wykorzystuje różne flagi **aiProcess\_** do przetwarzania modelu, takie jak **aiProcess\_Triangulate** (konwertowanie na trójkątne), **aiProcess\_FlipUVs** (odwrócenie współrzędnych UV) i **aiProcess\_GenSmoothNormals** (generowanie gładkich normalnych).
4. **std::cout << importer.GetErrorString() << std::endl;**: Wyświetla informacje o ewentualnych błędach wczytywania modelu, jeśli takie wystąpiły.
5. **aiMesh\* mesh = scene->mMeshes[0];**: Pobiera pierwszą siatkę (mesh) z wczytanego modelu. Może istnieć wiele siatek w jednym modelu, ale ta funkcja zakłada, że jest wczytywana tylko jedna siatka.
6. Przechodzi przez wszystkie wierzchołki siatki za pomocą pętli **for**. Dla każdego wierzchołka:
   * Pobiera pozycję wierzchołka (**aiVector3D vertex**) i dodaje ją do wektora **Vertices** w obiekcie **models[model\_i]** jako **glm::vec4**.
   * Pobiera normalną wierzchołka (**aiVector3D normal**) i dodaje ją do wektora **Normals** w obiekcie **models[model\_i]** jako **glm::vec4**.
   * Pobiera współrzędne tekstury wierzchołka (**aiVector3D texCoord**) i dodaje je do wektora **TexCoords** w obiekcie **models[model\_i]** jako **glm::vec2**.
7. Przechodzi przez wszystkie ściany (faces) siatki za pomocą pętli **for**. Dla każdej ściany:
   * Pobiera indeksy wierzchołków ściany (**face.mIndices[j]**) i dodaje je do wektora **Indices** w obiekcie **models[model\_i]**.

Podsumowując, funkcja **loadModel** używa biblioteki Assimp do wczytania modelu z pliku i przekształca go na format, który można wykorzystać do renderowania w programie OpenGL. Wierzchołki, normalne, współrzędne

Funkcja **freeOpenGLProgram** służy do zwalniania zasobów używanych przez program OpenGL, takich jak tekstury i shadery. Oto wyjaśnienie poszczególnych kroków w tej funkcji:

1. **freeShaders();**: Wywołuje funkcję **freeShaders**, która zwalnia zasoby związane ze shaderami. Nie jest to widoczne w podanym kodzie, ale można założyć, że ta funkcja jest odpowiedzialna za zwolnienie pamięci zajmowanej przez obiekty shaderów.
2. **glDeleteTextures(1, &texs[TEX\_TANK]);**: Usuwa teksturę o indeksie **TEX\_TANK** z pamięci GPU poprzez wywołanie funkcji **glDeleteTextures**. Podobnie jak w przypadku innych tekstur, takich jak **TEX\_ROCK1**, **TEX\_ROCK2**, itd., te linie kodu usuwają odpowiednie tekstury.

Podsumowując, funkcja **freeOpenGLProgram** ma na celu zwolnienie zasobów związanych z programem OpenGL przed zakończeniem działania programu. Jest to ważne, aby zapobiec wyciekom pamięci i zwolnić zasoby używane przez GPU.

Funkcja **drawBottom** służy do rysowania dna akwarium w programie OpenGL. Oto wyjaśnienie poszczególnych kroków w tej funkcji:

1. Definicja danych wierzchołków: Zdefiniowane są tablice zawierające współrzędne wierzchołków (**bottomVertices**), normalne wierzchołków (**bottomVertexNormals**) oraz współrzędne tekstur (**bottomTexCoords**) dla dna akwarium. Te dane określają kształt i wygląd dna akwarium.
2. Użycie shadera wodnego: Wywołanie **waterShader->use()** aktywuje używany shader wodny. Oznacza to, że operacje renderowania będą wykonywane zgodnie z definicją tego shadera.
3. Przesłanie macierzy projekcji, widoku i modelu: Przesłanie macierzy projekcji (**P**), widoku (**V**) i modelu (**M**) do shadera wodnego za pomocą odpowiednich funkcji **glUniformMatrix4fv**. Te macierze określają transformacje przestrzeni trójwymiarowej.
4. Włączenie atrybutów wierzchołków: Włączenie atrybutów wierzchołków (**vertex**, **normal**, **texCoord**) w shaderze wodnym za pomocą **glEnableVertexAttribArray** i **glVertexAttribPointer**. To pozwala OpenGL na skojarzenie danych wierzchołków z odpowiednimi atrybutami w shaderze.
5. Ustawienie parametrów tekstury: Ustawienie parametrów powtarzania tekstury (**GL\_REPEAT**) za pomocą **glTexParameteri**. To definiuje, jak tekstura powinna być powielana na powierzchni dna akwarium.
6. Aktywowanie i powiązanie tekstury: Wywołanie **glActiveTexture(GL\_TEXTURE0)** aktywuje jednostkę teksturującą 0. Następnie wywołanie **glBindTexture(GL\_TEXTURE\_2D, texs[TEX\_BOTTOM])** powiązuje konkretną teksturę (**texs[TEX\_BOTTOM]**) z jednostką teksturującą. To oznacza, że ta tekstura będzie używana podczas renderowania dna akwarium.
7. Przesłanie wartości uniform dla tekstury: Przesłanie wartości 0 do uniform **tex** w shaderze wodnym za pomocą **glUniform1i(waterShader->u("tex"), 0)**. Oznacza to, że jednostka teksturująca 0 jest używana do renderowania tekstury dna akwarium.
8. Rysowanie wierzchołków: Wywołanie **glDrawArrays(GL\_TRIANGLES, 0, bottomVertexCount)** rysuje wierzchołki dna akwarium jako trójkąty.
9. Wyłączenie atrybutów wierzchołków: Wywołanie **glDisableVertexAttribArray** dla atrybutów wierzchołków (**vertex**, **normal**, **texCoord**) w shaderze wodnym wyłącza ich użycie po zakończeniu rysowania.

Podsumowując, funkcja **drawBottom** rysuje wodny fragment dna akwarium z wykorzystaniem odpowiedniego shadera, danych wierzchołków i tekstur.

Funkcja **drawModel** służy do rysowania modeli w programie OpenGL. Oto wyjaśnienie poszczególnych kroków w tej funkcji:

1. Użycie shadera wodnego: Wywołanie **waterShader->use()** aktywuje używany shader wodny. Oznacza to, że operacje renderowania będą wykonywane zgodnie z definicją tego shadera.
2. Przesłanie macierzy projekcji, widoku i modelu: Przesłanie macierzy projekcji (**P**), widoku (**V**) i modelu (**M**) do shadera wodnego za pomocą odpowiednich funkcji **glUniformMatrix4fv**. Te macierze określają transformacje przestrzeni trójwymiarowej.
3. Przesłanie wartości świateł: Przesłanie wartości świateł (**light1** i **light2**) do shadera wodnego za pomocą funkcji **glUniform4fv**. Oznacza to, że informacje o świetle zostaną uwzględnione podczas renderowania modelu.
4. Włączenie atrybutów wierzchołków: Włączenie atrybutów wierzchołków (**vertex** i **texCoord**) w shaderze wodnym za pomocą **glEnableVertexAttribArray** i **glVertexAttribPointer**. To pozwala OpenGL na skojarzenie danych wierzchołków z odpowiednimi atrybutami w shaderze.
5. Aktywowanie i powiązanie tekstury: Wywołanie **glActiveTexture(GL\_TEXTURE0)** aktywuje jednostkę teksturującą 0. Następnie wywołanie **glBindTexture(GL\_TEXTURE\_2D, texs[texture])** powiązuje konkretną teksturę (**texs[texture]**) z jednostką teksturującą. To oznacza, że ta tekstura będzie używana podczas renderowania modelu.
6. Przesłanie wartości uniform dla tekstury: Przesłanie wartości 0 do uniform **tex** w shaderze wodnym za pomocą **glUniform1i(waterShader->u("tex"), 0)**. Oznacza to, że jednostka teksturująca 0 jest używana do renderowania tekstury modelu.
7. Rysowanie modelu: Wywołanie **glDrawElements** z odpowiednimi parametrami rysuje model za pomocą indeksów (**models[model\_i].Indices**) i danych wierzchołków (**models[model\_i].Vertices.data()** i **models[model\_i].TexCoords.data()**).
8. Wyłączenie atrybutów wierzchołków: Wywołanie **glDisableVertexAttribArray** dla atrybutów wierzchołków (**vertex**, **texCoord**) w shaderze wodnym wyłącza ich użycie po zakończeniu rysowania.

Podsumowując, funkcja **drawModel** rysuje model z wykorzystaniem shadera wodnego, danych wierzchołków i tekstury. Światło i macierze transformacji są również przekazywane do shadera, aby uwzględnić oświetlenie i przekształcenia przestr

Funkcja **drawScene** jest główną pętlą rysującą w programie OpenGL. Oto wyjaśnienie poszczególnych kroków w tej funkcji:

1. **glClear**: Wywołanie **glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)** czyści bufor koloru i bufor głębokości, aby przygotować je do kolejnego renderowania.
2. Ustawienie kamery: Tworzenie macierzy widoku (**V**) za pomocą **glm::lookAt**, która określa położenie kamery (**cameraPos**), punkt na który kamera jest skierowana (**cameraPos + cameraFront**) oraz wektor wskazujący kierunek w górze (**cameraUp**). Macierz projekcji (**P**) jest tworzona za pomocą **glm::perspective**, która definiuje perspektywiczne przekształcenie.
3. Rysowanie świateł: Rysowanie dwóch świateł na podstawie pozycji świateł (**light1** i **light2**). Każde światło ma własną macierz modelu (**Ml1** i **Ml2**). Wywołanie **drawLight** z odpowiednimi parametrami rysuje światło na podstawie tych macierzy.
4. Rysowanie dna: Wywołanie **drawBottom** z odpowiednimi parametrami rysuje dno na podstawie macierzy projekcji (**P**), widoku (**V**) i modelu (**M**).
5. Rysowanie modeli ryb: Dla każdej ryby tworzone są odpowiednie macierze modelu (**Mf**, **Mf1**, itd.) za pomocą operacji transformacji, takich jak przesunięcie (**glm::translate**) i obrót (**glm::rotate**). Następnie wywołanie **drawModel** z odpowiednimi parametrami rysuje model ryby na podstawie tych macierzy.
6. Rysowanie skał i roślin: Wywołanie **drawRocks** i **drawPlants** rysuje skały i rośliny na podstawie odpowiednich macierzy projekcji i widoku.
7. Rysowanie akwarium: Wywołanie **drawTank** i **drawGlass** rysuje akwarium i szkło na podstawie odpowiednich macierzy projekcji, widoku i modelu.
8. **glfwSwapBuffers**: Wywołanie **glfwSwapBuffers(window)** wymienia bufor przedni z tylnym, co powoduje wyświetlenie aktualnie wyrenderowanej klatki.

To jest ogólny opis funkcji **drawScene**, która rysuje całą scenę w programie OpenGL. Szczegółowe implementacje funkcji **drawLight**, **drawBottom**, **drawModel**, **drawRocks**, **drawPlants**, **drawTank** i **drawGlass** nie są dostępne w podanym kodzie, więc nie mogę podać dokładnych szczegółów na ich temat.

Początek formularza