POLITECHNIKA BIAŁOSTOCKA

WYDZIAŁ INFORMATYKI

PRACA DYPLOMOWA INŻYNIERSKA

TEMAT: EDYTOR MODELI 3D OPARTYCH O WOKSELE

WYKONAWCA: PAWEŁ ALEKSIEJUK

PODPIS: .....

PROMOTOR: DR INŻ. ŁUKASZ GADOMER

PODPIS: .....

BIAŁYSTOK 2022 ROK

Karta dyplomowa

POLITECHNIKA BIAŁOSTOCKA Wydział Informatyki	Studia stacjonarne stacjonarne/niestacjonarne pierwszego stopnia studia I stopnia/studia II stopnia	Nr albumu studenta 105527 Rok akademicki 2021/2022 Kierunek studiów informatyka Specjalność -

Paweł Aleksiejuk
Imiona i nazwisko studenta

TEMAT PRACY DYPLOMOWEJ:

Edytor modeli 3D opartych o woksele

Zakres pracy:

1. Przegląd podobnych rozwiązań dostępnych na rynku.
2. Zdefiniowanie wymagań stawianych wobec rozwiązania.
3. Opracowanie prostego silnika 3D.
4. Stworzenie narzędzia do edycji modelu 3D.
5. Testowanie stworzonego rozwiązania.

Słowa kluczowe (max 5): Woksel, 3D, Edytor Modeli, C++

dr inż. Łukasz Gadomer
Imię i nazwisko, stopień/tytuł promotora - podpis

31.01.2021	28.02.2022	
Data wydania tematu pracy dyplomowej - podpis promotora	*Regulaminowy termin złożenia pracy dyplomowej*	*Data złożenia pracy dyplomowej - potwierdzenie dziekanatu*

Ocena promotora	*Podpis promotora*

Imię i nazwisko, stopień/ tytuł recenzenta	*Ocena recenzenta*	*Podpis recenzenta*

Subject of diploma thesis

3D model editor based on voxels

Summary

Graphics in modern usage is mainly associated with computer representation of data. We most commonly distinguish between two types of graphics: 2D (two-dimensional) and 3D (three-dimensional). 2D graphics involve presenting a visual output based on a two-dimensional object, while 3D graphics, involve a three-dimensional object.

A voxel is a representation of a point in a three-dimensional space. The name voxel originated from a combination of the words, volume and element and is similar to a picture and an element in the case of a pixel.

Interested in these topics, the author decided to trace the creation of 3D graphics back to the model editor, creating one from scratch as part of this work. This editor is intended to allow the user to create a voxel-based 3D model using built-in editing mechanisms.

The motivation for writing this thesis was to create a simple functional 3D engine, along with tools for creating models using this engine. Later on, I plan to extend this project by creating a fully functional 3D game.

Białystok, dnia 19.01.2022 r.

Paweł Aleksiejuk
Imiona i nazwisko studenta
105527
Nr albumu
informatyka, stacjonarne
Kierunek i forma studiów
dr inż. Łukasz Gadomer
Promotor pracy dyplomowej

OŚWIADCZENIE

Przedkładając w roku akademickim 2021/2022 Promotorowi **dr inż. Łukasz Gadomer** pracę dyplomową pt.: **Edytor modeli 3D opartych o woksele**, dalej zwaną pracą dyplomową,**oświadczam, że:**

1) praca dyplomowa stanowi wynik samodzielnej pracy twórczej/~~zespołowej pracy twórczej~~*;
2) wykorzystując w pracy dyplomowej materiały źródłowe, w tym w szczególności: monografie, artykuły naukowe, zestawienia zawierające wyniki badań (opublikowane, jak i nieopublikowane), materiały ze stron internetowych, w przypisach wskazywałem/am ich autora, tytuł, miejsce i rok publikacji oraz stronę, z której pochodzą powoływane fragmenty, ponadto w pracy dyplomowej zamieściłem/am bibliografię;
3) praca dyplomowa nie zawiera żadnych danych, informacji i materiałów, których publikacja nie jest prawnie dozwolona;
4) praca dyplomowa dotychczas nie stanowiła podstawy nadania tytułu zawodowego, stopnia naukowego, tytułu naukowego oraz uzyskania innych kwalifikacji;
5) treść pracy dyplomowej przekazanej do dziekanatu Wydziału Informatyki jest jednakowa w wersji drukowanej oraz w formie elektronicznej;
6) jestem świadomy/a, że naruszenie praw autorskich podlega odpowiedzialności na podstawie przepisów ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. z 2019 r. poz. 1231, późn. zm.), jednocześnie na podstawie przepisów ustawy z dnia 20 lipca 2018 roku Prawo o szkolnictwie wyższym i nauce (Dz. U. poz. 1668, z późn. zm.) stanowi przesłankę wszczęcia postępowania dyscyplinarnego oraz stwierdzenia nieważności postępowania w sprawie nadania tytułu zawodowego;
7) udzielam Politechnice Białostockiej nieodpłatnej, nieograniczonej terytorialnie i czasowo licencji wyłącznej na umieszczenie i przechowywanie elektronicznej wersji pracy dyplomowej w zbiorach systemu Archiwum Prac Dyplomowych Politechniki Białostockiej oraz jej zwielokrotniania i udostępniania w formie elektronicznej w zakresie koniecznym do weryfikacji autorstwa tej pracy i ochrony przed przywłaszczeniem jej autorstwa.

.....
czytelny podpis studenta

*w przypadku pracy zespołowej

Spis treści

Wstęp

Grafika we współczesnym użyciu kojarzy się głównie z komputerowym przedstawieniem danych. Dane te tworzą medium wizualne, które mogą być wyświetlane między innymi na ekranach naszych monitorów komputerowych. Najczęściej rozróżniamy dwie rodzaje grafik: 2D (ang. *two-dimensional*) i 3D (ang. *three-dimensional*). Grafika 2D polega na przedstawieniu medium wizualnego opartego na obiekcie o dwóch wymiarach, zaś grafika 3D, analogicznie na obiekcie o trzech wymiarach.

Wokselem (ang. *voxel*) [1] nazywamy przedstawienie punktu w trójwymiarze. Nazwa woksel jest połączeniem angielskich słów *volume* oraz *element* i jest to analogiczne połączenie do *picture* i *element* w przypadku piksela (ang. *pixel*).

Zainteresowany tymi tematami, autor postanowił prześledzić drogę tworzenia grafiki 3D od strony edytora modeli, tworząc go od postaw w ramach tej pracy. Edytor ten ma pozwolić użytkownikowi na kreację modelu 3D opartego na wokselach, wykorzystując wbudowane mechanizmy edycji.

Motywacją do napisania tej pracy było chęć stworzenia prostego funkcjonalnego silnika graficznego wraz z narzędziem do tworzenia modeli obsługiwanych przez ten silnik. W późniejszym czasie, planuję rozszerzyć ten projekt, tworząc w pełni funkcjonalną grę 3D.

Zakres pracy obejmował:

- Przegląd podobnych rozwiązań dostępnych na rynku.
- Zdefiniowanie wymagań stawianych wobec rozwiązania.
- Opracowanie prostego silnika 3D.
- Stworzenie narzędzia do edycji modelu 3D.
- Testowanie stworzonego rozwiązania.

Rozdział 1 przedstawia 5 istniejących już na rynku edytorów graficznych opartych o woksele, w celu zaznajomienia się z podstawowymi funkcjonalnościami postawionymi przez ich autorów.

Rozdział 2 skupia się na przedstawieniu dogłębnie projektu systemu, w celu zapoznania się z wymaganiami wobec aplikacji.

Rozdział 3 opisuje zastosowane technologie i rozwiązania, tłumacząc logikę i motywację za wyborem każdych z nich.

Rozdział 4 opisuje realizacje aplikacji. Zawiera w sobie informacje na temat użytych i stworzonych struktur danych, opisując implementację i specyfikację każdego z nich oraz zaznajamia z wybranymi implementacjami funkcjonalności.

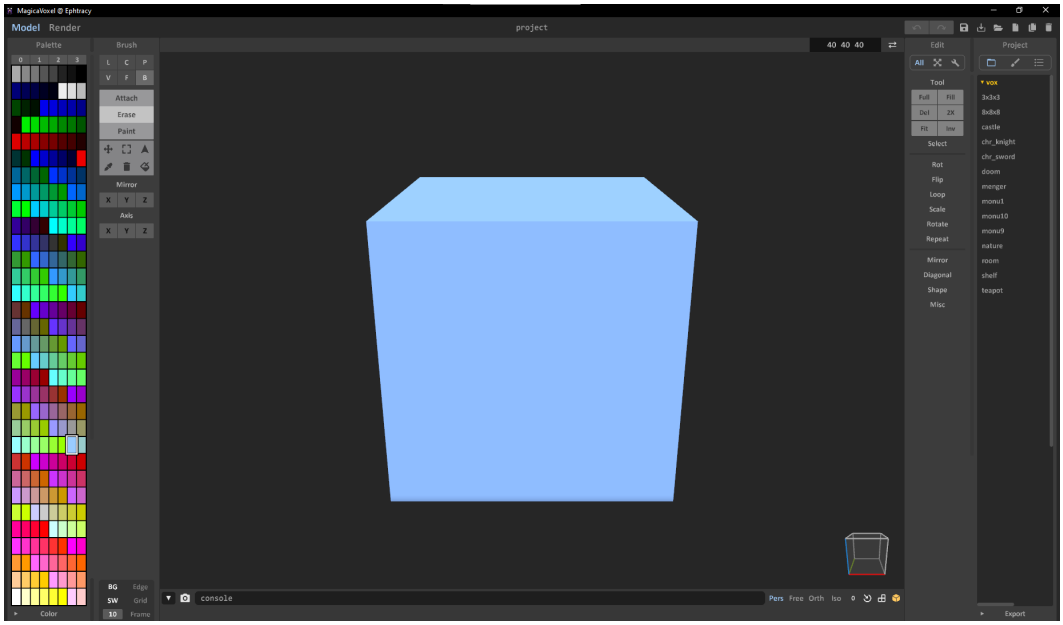
Rozdział 5 stanowi techniczne wprowadzenie do nawigacji aplikacją. W nim znajduje się prezentacja systemu oraz instrukcja użytkownika, której pozwolą na zrozumienie interfejsu oraz nawigacji w aplikacji.

1. Przegląd istniejących rozwiązań

Z uwagi na specjalistyczne zastosowanie stworzonego edytora graficznego, a mianowicie tworzenie specjalnych obiektów obsługiwanych przez wbudowany silnik graficzny, istniejące rozwiązania w głównej mierze mają służyć jako wykaz podstawowych, jak i dodatkowych funkcjonalności do możliwej implementacji w ostatecznym rozwiązaniu.

1.1 MagicaVoxel

MagicaVoxel [9] jest najpopularniejszym darmowym desktopowym edytorem wokseli dostępnym aktualnie na rynku. Stworzony i na bieżąco aktualizowany przez użytkownika o pseudonimie @ephtracy pozwala na nie tylko tworzenie modeli, ale też zdjęć do późniejszego udostępniania. Taka funkcjonalność pozwala na przetestowanie modelu w różnych warunkach, które są edytowalne poprzez parametry w wewnętrznym silniku renderującym. Interfejs rozwiązania został przedstawiony na rysunku 1.1



Rysunek 1.1: Ekran startowy programu MagicaVoxel (Windows), źródło: [9]

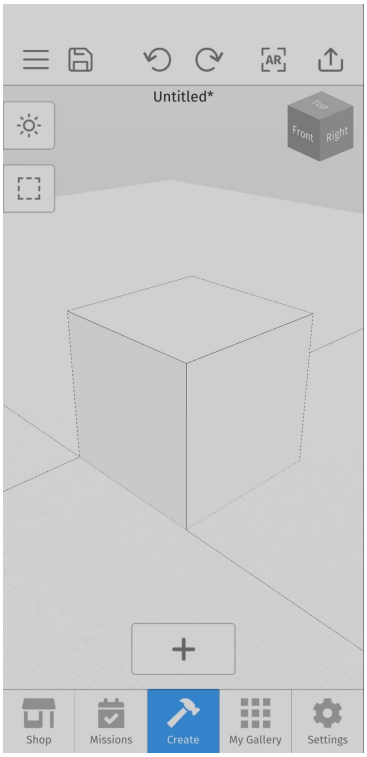
Główne atuty oprogramowania według producenta:

- Zaawansowany wewnętrzny silnik renderujący.
- Całkowicie darmowe oprogramowanie, nawet w przypadku użycia komercyjnego.

MagicaVoxel jest dostępny za darmo na platformach Windows i macOS.

1.2 Mega Voxels Play

Mega Voxels Play [13] to darmowy mobilny edytor stworzony przez Go Real Games. Tak jak większość edytorów wokselowych, pozwala na podstawowe operacje takie jak dodawanie, usuwanie i malowanie. Aplikacja posiada wbudowany sklep, który pozwala na pobranie gotowych modeli, w celu późniejszego wykorzystania. Interfejs rozwiązania został przedstawiony na rysunku 1.2



Rysunek 1.2: Ekran startowy programu Mega Voxels Play (Android), źródło: [13]

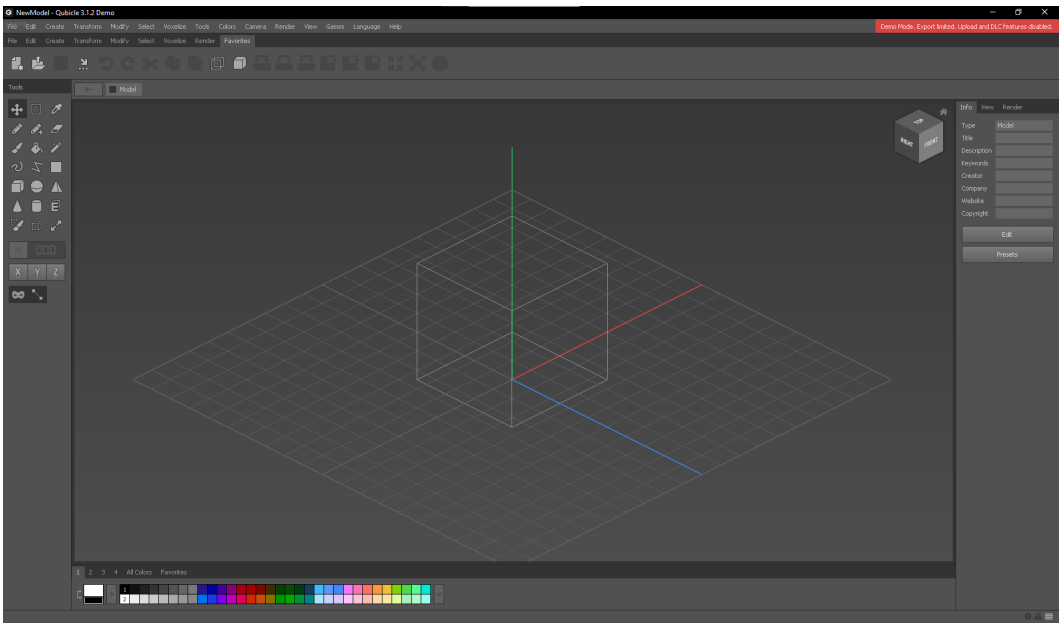
Główne atuty oprogramowania według producenta:

- Duża ilość bazowych modeli do pobrania.
- Prostość w obsłudze.
- Wsparcie dla AR (Rozszerzonej rzeczywistości).
- Różne efekty przetwarzania końcowego.

Mega Voxels Play jest dostępny za darmo na platformach mobilnych (Android i iOS).

1.3 Qubicle

Qubicle [12] jest zaawansowanym desktopowym narzędziem stworzonym przez Minddesk, przeznaczonym do tworzenia wokselowych modeli. Z porównaniem do poprzedników, aplikacja nie posiada limitu wielkości modeli, co pozwala użytkownikom na swobodne tworzenie wielkich modeli, jak i całych terenów. Dodatkowo oprócz standardowego w edytorach formatu .obj (Wavefront File), wspierane są też takie formaty jak .fbx (Autodesk), .dae (Collada). Interfejs rozwiązania został przedstawiony na rysunku 1.3



Rysunek 1.3: Ekran startowy programu Qubicle (Windows, Steam), źródło: [12]

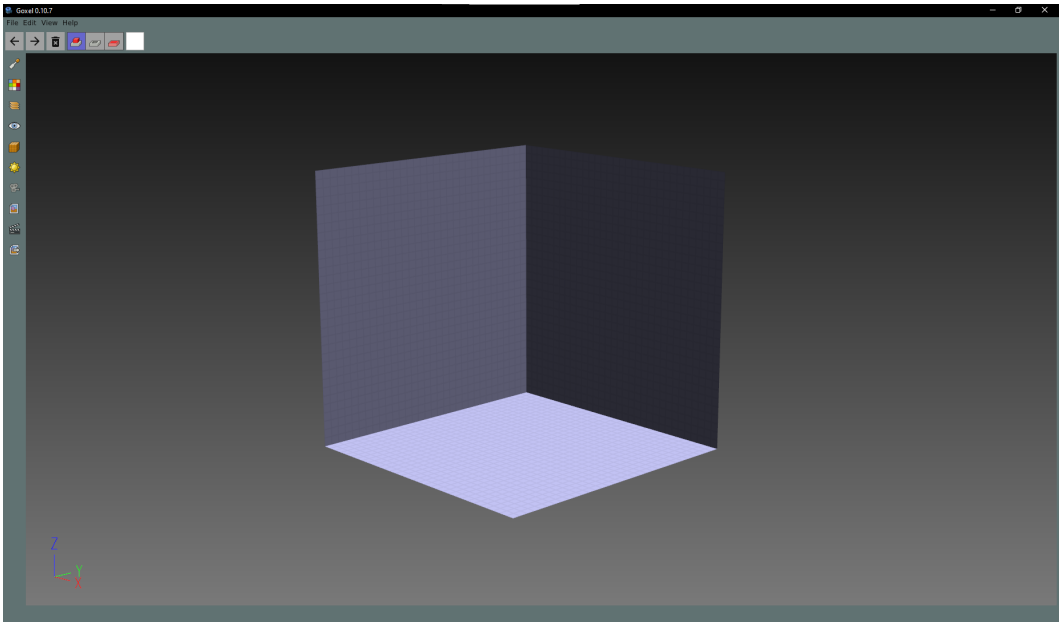
Główne atuty oprogramowania według producenta:

- Bardzo dużo narzędzi do edycji.
- Proste w obsłudze.
- Wbudowane narzędzie do konwersji z modelu siatkowego na model wokselowy.
- Wiele formatów do eksportu modeli.

Qubicle jest dostępny w czterech wersjach na platformach Windows i macOS, wersja okrojona (demo) za darmo, wersja podstawowa (bazowa) za 53.99 PLN, wersja rozszerzona (indie) za 89.99 PLN i pełna opcja (pro) za 410.56 PLN.

1.4 Goxel

Goxel [16] jest otwartym oprogramowaniem do edycji modeli wokselowych na komputery osobiste i urządzenia mobilne stworzone przez użytkownika o pseudonimie @guillaumechereau (GitHub). Główną funkcjonalnością Goxel, jest możliwość tworzenia warstw, w taki sam sposób jak w popularnych aplikacjach do manipulacji obrazami, między innymi takim jaki jest Adobe Photoshop. Interfejs rozwiązania został przedstawiony na rysunku 1.4



Rysunek 1.4: Ekran Startowy programu Goxel (Windows), źródło: [16]

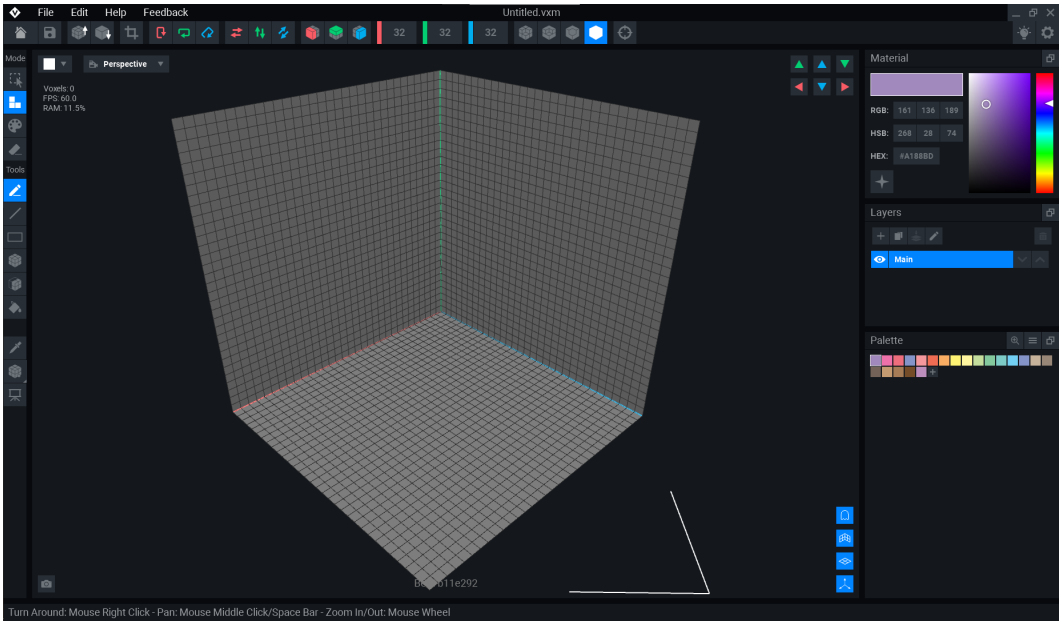
Główne atuty oprogramowania według producenta:

- Nieskończona wielkość sceny.
- Możliwość tworzenia obiektów na różnych warstwach.
- Wieloplatformowość.
- Wiele formatów do eksportu modeli.

Goxel jest dostępny za darmo na platformach Windows, Linux, iOS i macOS, a w przypadku platformy Android za opłatą 25.99 PLN.

1.5 VoxEdit Beta

VoxEdit Beta [23] jest darmowym oprogramowaniem stworzonym przez Pixowl do gry The Sandbox Game. Unikalną funkcjonalnością na tle innych aplikacji do edycji wokseli, jest możliwość montowania szkieletu i jego późniejszej animacji. Interfejs rozwiązania został przedstawiony na rysunku 1.5



Rysunek 1.5: Ekran startowy programu VoxEdit Beta (Windows), źródło: [23]

Główne atuty oprogramowania według producenta:

- Możliwość tworzenia animacji.
- Specjalny tryb edycji bloków.
- Przyjazny interfejs dla użytkownika.

VoxEdit Beta jest dostępny za darmo na platformach Windows i macOS.

2. Projekt systemu

W tym rozdziale przedstawiono główne rozwiązania z zakresu inżynierii oprogramowania i projektowania, takie jak wymagania funkcjonalne i niefunkcjonalne co do niniejszej pracy oraz diagramy, na których zobrazowano działanie systemu oraz zamieszczono ich opis. Tworząc taki projekt systemu, umożliwi on zaznajomienie się z podstawowym działaniem aplikacji.

2.1 Wymagania

Na podstawie informacji zebranych z rozdziału 1 „Przegląd istniejących rozwiązań", jak i wiedzy autora na temat programów graficznych, określone zostały podstawowe wymagania dotyczące aplikacji. Podzielono je na wymagania funkcjonalne oraz niefunkcjonalne.

2.1.1 Wymagania funkcjonalne

Aplikacja stworzona w ramach niniejszej pracy powinna spełniać następujące funkcjonalności:

- Tworzenie modeli 3D.
- Prosty interfejs użytkownika.
- Zmiana pozycji i wielkości okienek.
- Edycja modeli w czasie rzeczywistym.
- Tworzenie własnych opisów materiałów
- Zapis i odczyt modelu.
- Zmiana właściwości oświetlenia.

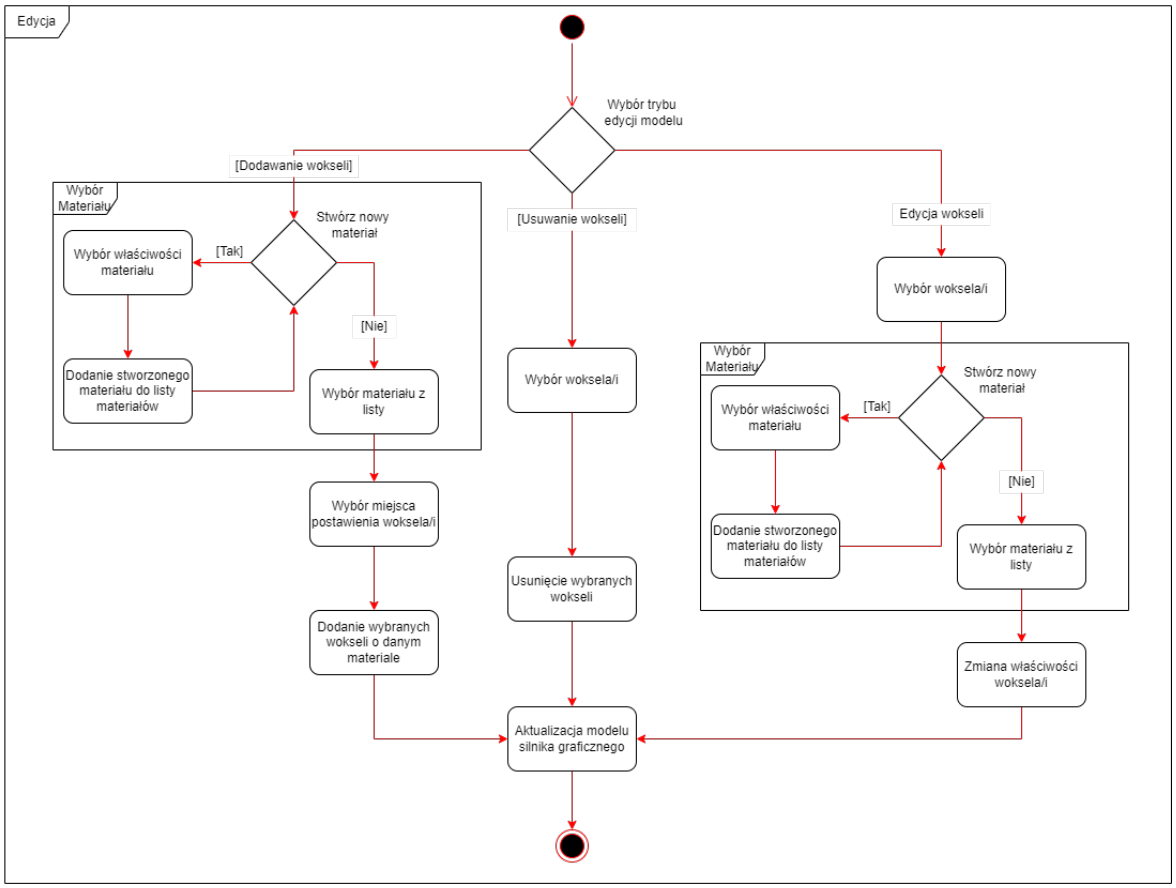
2.1.2 Wymagania niefunkcjonalne

Prócz wymagań funkcjonalnych zdefiniowano szereg wymagań niefunkcjonalnych, które aplikacja powinna spełniać. Należą do nich:

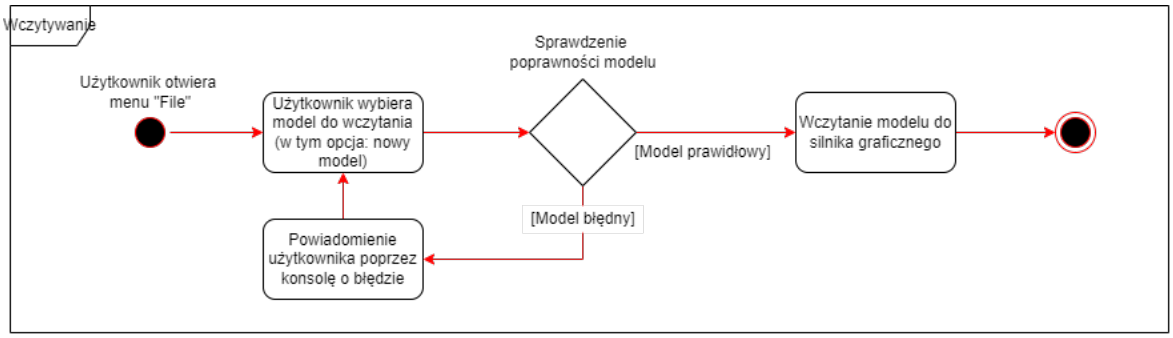
- Możliwość ponownego użycia silnika 3D w innych projektach.
- Wysoka responsywność na zmiany w modelu.
- Działanie na wielu platformach desktopowych (Windows, Linux, macOS).
- Konsola debugująca w czasie rzeczywistym.

2.2 Diagramy stanów

W aplikacji rozróżniamy dwa główne stany, stan gotowości do edycji albo stan wczytywania. Pierwszy z nich, przedstawiony na rysunku 2.1 jest odpowiedzialny za wprowadzanie zmian na obiekcie 3D. Z uwagi na to, że silnik renderuje klatki w czasie rzeczywistym, stan ten jest w ciągłej gotowości, czekający na ingerencje użytkownika. Drugi stan (rysunek 2.2) jest wywoływany w momencie wybrania nazwy pliku do wczytania.



Rysunek 2.1: Diagram stanów dla edycji, źródło: opracowanie własne

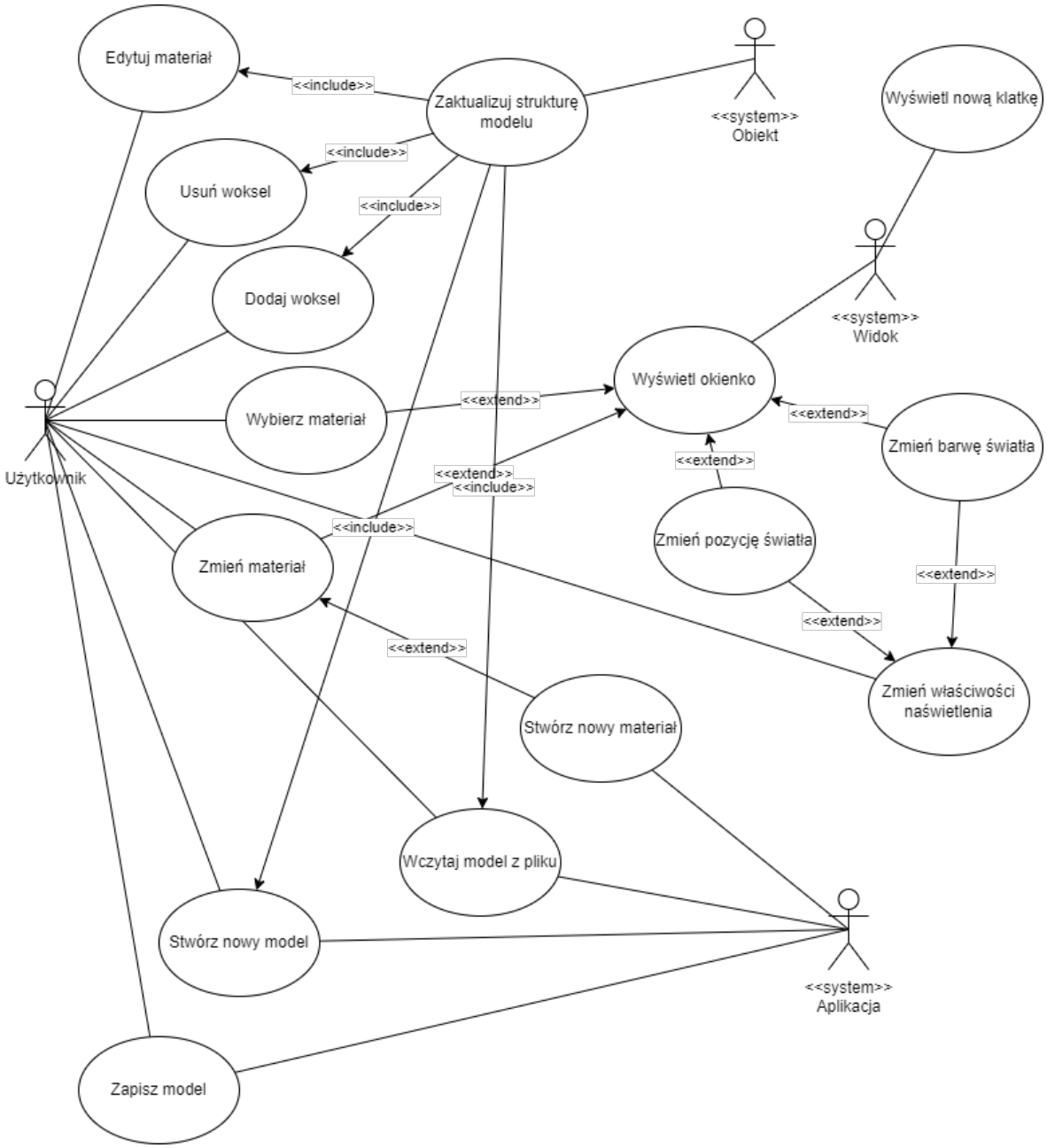


Rysunek 2.2: Diagram stanów dla wczytywania, źródło: opracowanie własne

2.3 Diagram przypadków użycia i opisy

Na rysunku 2.3 przedstawiono diagram przypadków użycia, w którym zawarte są wszystkie najważniejsze wymagania funkcjonalne w formie graficznej.

Głównymi przypadkami użycia tego systemu są metody manipulacji modelu 3D. To one pozwalają nam na dodawanie wokseli, usuwanie ich oraz edytowanie materiału. Przypadki te są odpowiednio opisane w tabeli 2.1, tabeli 2.2 oraz tabeli 2.3.



Rysunek 2.3: Diagram przypadków użycia, źródło: opracowanie własne

Tabela 2.1: Opis przypadku użycia „Dodaj woksel”

Sekcja	Treść
Uczestniczący aktorzy	Użytkownik, Widok, Aplikacja, Obiekt
Warunki wstępne	W okienku „Edit Mode” zaznaczony tryb „Add”
Warunki końcowe	Dodanie woksela do obiektu
Rezultat	Pojawienie się woksela w miejscu wskazanym przez użytkownika
Scenariusz główny	1. Użytkownik wybiera materiał z listy, bądź dodaje swój własny i go zatwierdza. 2. Użytkownik nacelowuje na interesującą go sciankę woksela, w celu postawienia na obok niej nowego woksela, po czym zatwierdza prawym przyciskiem myszy. 3. Aplikacja przekazuje do obiektu dane kliknięcia. 4. Obiekt aktualizuje strukturę danych. 5. Widok zostaje odświeżony w następnej klatce. 6. Użytkownik widzi efekt swojego działania na modelu 3D.
Scenariusz wyjątku	Zdarzenie: Użytkownik nie kliknął na sciankę istniejącego woksela Wynik: Brak dodania woksela do modelu 3D
Zależności czasowe	1. Częstotliwość wykonania: 0 lub więcej na sesję. 2. Typowy czas realizacji: 8.9 ms. 3. Maksymalny czas realizacji: 33,2 ms.
Wartości uzyskane przez aktorów po zakończeniu przypadków użycia	1. Pojawienie się woksela w miejscu i o materiale wybranym przez użytkownika. 2. Obiekt posiada zaktualizowaną strukturę o woksela.

Tabela 2.2: Opis przypadku użycia „Usuń woksel”

Sekcja	Treść
Uczestniczący aktorzy	Użytkownik, Widok, Aplikacja, Obiekt
Warunki wstępne	W okienku „Edit Mode” zaznaczony tryb „Remove”
Warunki końcowe	Usunięcie wskazanego woksela z obiektu
Rezultat	Zniknięcie woksela w miejscu wskazanym przez użytkownika
Scenariusz główny	1. Użytkownik nacelowuje na interesujący go woksel, po czym zatwierdza prawym przyciskiem myszy. 2. Aplikacja przekazuje do obiektu dane kliknięcia. 3. Obiekt zwraca woksel zainteresowania. 4. Aplikacja usuwa zwrócony woksel. 5. Widok zostaje odświeżony w następnej klatce. 6. Użytkownik widzi efekt swojego działania na modelu 3D.
Scenariusz wyjątku	Zdarzenie: Użytkownik nie kliknął w istniejącego woksela Wynik: Brak usunięcia woksela z modelu 3D
Zależności czasowe	1. Częstotliwość wykonania: 0 lub więcej na sesję. 2. Typowy czas realizacji: 5.9 ms. 3. Maksymalny czas realizacji: 16,6 ms.
Wartości uzyskane przez aktorów po zakończeniu przypadków użycia	1. Zniknięcie woksela w miejscu wybranym przez użytkownika. 2. Obiekt posiada zaktualizowaną strukturę bez kliknięgo woksela.

Tabela 2.3: Opis przypadku użycia „Edytuj materiał”

Sekcja	Treść
Uczestniczący aktorzy	Użytkownik, Widok, Aplikacja, Obiekt
Warunki wstępne	W okienku „Edit Mode” zaznaczony tryb „Color”
Warunki końcowe	Zmiana materiału we wskazanym miejscu w obiekcie
Rezultat	Zmiana materiału woksela w miejscu wskazanym przez użytkownika
Scenariusz główny	1. Użytkownik nacelowuje na interesujący go woksel, po czym zatwierdza prawym przyciskiem myszy. 2. Aplikacja przekazuje do obiektu dane kliknięcia. 3. Obiekt zwraca woksel zainteresowania. 4. Aplikacja zmiania kolor zwróconego woksela na ostatni wybrany materiał. 5. Widok zostaje odświeżony w następnej klatce. 6. Użytkownik widzi efekt swojego działania na modelu 3D.
Scenariusz wyjątku	Zdarzenie: Użytkownik nie kliknął w istniejącego woksela Wynik: Brak zmiany koloru woksela w modelu 3D
Zależności czasowe	1. Częstotliwość wykonania: 0 lub więcej na sesję. 2. Typowy czas realizacji: 3.2 ms. 3. Maksymalny czas realizacji: 16,6 ms.
Wartości uzyskane przez aktorów po zakończeniu przypadków użycia	1. Zmiana właściwości materiału woksela w miejscu wybranym przez użytkownika. 2. Obiekt posiada zmienioną specyfikacje materiału w kliknięty wokselu.

3. Zastosowane technologie i rozwiązania

Aby stworzyć aplikację, wpierw trzeba podjąć decyzję dotyczącą technologii w jakiej ma ona powstać. Programy działające w czasie rzeczywistym, wymagają dużej odpowiedzialności ze strony programisty, by zapewnić użytkownikowi jak najpłynniejsze doświadczenie podczas użytkowania. W tym celu, wybrano rozwiązania pozwalające programiście na jak największą ingerencję w sposób działania, jednocześnie pozwalając na wykorzystanie gotowych rozwiązań.

3.1 Języki programowania

Aplikacja została napisana w języku C++ w wersji ISO/IEC 14882:2011 [10] (C++11), z wyjątkiem plików cieniowania barw (ang. *shader files*), które zostały napisane w języku GLSL (OpenGL Shading Language) w wersji „330 core" [15]. Są wysokopoziomowymi językami, ze składnią pochodzącą z języka C, pozwalającymi na bezpośredni dostęp do zasobów sprzętowych i funkcji systemowych.

3.2 Środowisko programistyczne

Z uwagi na wymaganie multiplatformowości postawione w etapie projektowania systemu, wszystkie narzędzia programistyczne do tworzenia oprogramowania, powinny te zapotrzebowanie spełniać. Visual Studio Code [21] jest idealnym przykładem narzędzia, będącego multiplatformowe, bezpłatne oraz elastyczne. Głównym atutem tego IDE (ang. *Integrated Development Environment*), jest ogromna biblioteka rozszerzeń. Do stworzenie aplikacji, użyte zostały następujące pozycje:

- C/C++ [19] - wsparcie dla języków C i C++.
- CMake [25] - wsparcie dla języka CMake.
- CMake Tools [20] - integracja programu CMake.
- VSCode Icons [24] - zestaw ikon.
- Path Intellisense [18] - automatyczne kończenie nazw plików.

3.3 Biblioteki

Jedną z największych przewag języka C++ jest łatwość korzystania z bibliotek napisanych w C, C++ lub innych, niezależnie od platformy systemowej. Kontynuując temat multiplatformowości, systemy obsługi okien nie jest wspólny dla każdego z systemów operacyjnych. W celu rozwiązania tego problemu, użyta została biblioteka GLFW [8], pełniąca rolę interfejsu programistycznego aplikacji (ang. *Application Programming Interface*), wspierająca systemy okien takie jak Windows, macOS, X11 oraz Wayland.

Silniki graficzne są zaawansowanymi modułami, które wykonują wiele obliczeń matematycznych. W celu zapewnienia najwyższej wydajności działania tych obliczeń, jak i zapewnieniu zgodności struktur matematycznych pomiędzy kodem pisanym w C++, a kodem pisanym w GLSL, wybrano bibliotekę glm [3].

Do komunikacji pomiędzy użytkownikiem a silnikiem 3D, wykorzystano bibliotekę ImGui [22], pełniącą rolę graficznego interfejsu użytkownika (ang. *GUI Graphical User Interface*. Została stworzona specjalnie na potrzeby szybkich iteracji, potrzebnych między innymi w silnikach gier, aplikacjach 3D czasu rzeczywistego oraz aplikacjach pełnoekranowych.

Zastosowanie OpenGL w aplikacji wymaga użycia biblioteki ładującej, odpowiedzialnej za ładowanie wskaźników, funkcji w czasie rzeczywistym, jąder, jak i rozszerzeń. [14] Jedną z takich bibliotek jest Glad [4], która jest generowana przez użytkownika na podstawie jego potrzeb.

4. Realizacja projektu

Głównym tematem niniejszej pracy jest stworzenie edytora modeli 3D opartego o woksele. W tym rozdziale zostaną zaprezentowane zagadnienia dotyczące aplikacji, takie jak sposób przechowywania danych, wybrane funkcjonalności oraz proces testowania aplikacji.

4.1 Struktury danych

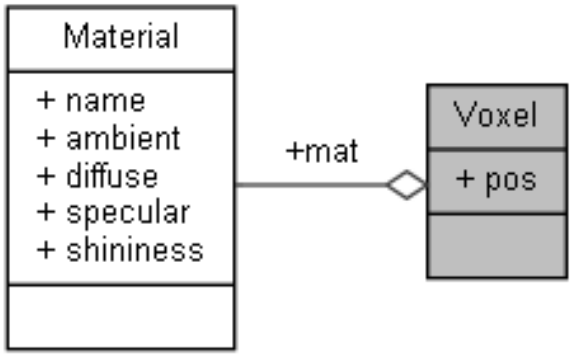
Klasa „Object" (listing 4.1) jest odpowiedzialna za przechowywanie i obsługę danych dotyczących modelu. Jako rozwiązanie do przetrzymywania danych w strukturze, autor zastosował połączenie tablicy haszującej (ang. *hash table*) jako wyznacznik istnienia woksela w danym miejscu oraz wektora wokseli, zawierającego struktury „Voxel" (rysunek 4.1).

```
class Object
{
public:
  Object();
  void Draw(MVP mvp, glm::vec3 cameraPosition, Light light);
  void AddVoxel(glm::ivec3 pos, Material mat);
  void ChangeColor(Voxel *voxel, Material mat);
  void RemoveVoxel(Voxel *voxel);
  void RemoveVoxel(glm::vec3 pos);
  void Reset();
  void Save();
  void Load(std::string objectPath);
  Voxel *CheckRay(glm::vec3 ray_origin, glm::vec3 ray_dir, glm::vec3 ↙
      ↳ &newBlockLoc);
  std::vector<Voxel> GetListOfVoxels();

  std::string name;
private:
  ...
  std::vector<Voxel> m_voxels;
  bool m_hashVoxels[VOXEL_COUNT][VOXEL_COUNT][VOXEL_COUNT];
};
```

Listing 4.1: Fragment kodu klasy „Object" odpowiedzialnego za obsługę modelu 3D

Aplikacja obsługuje dwie własne struktury danych, które są przechowywane, na poziomie dysku. Pozwala to użytkownikowi na dostęp do tych danych, nawet po zakończeniu sesji.



Rysunek 4.1: Diagram struktury „Voxel”, źródło: wygenerowane za pomocą doxygen [17]

4.1.1 Właściwości materiałów

Plik tekstowy z zakończeniem `.mat` odpowiedzialny jest za przechowywanie właściwości materiałów. Tworzony jest za pomocą okna „Material”, poprzez globalną funkcję „saveMaterial” przedstawioną na listingu 4.2. Format pliku `.mat` został przedstawiony na listingu 4.3 wraz z opisem. Właściwości materiałów dostarczonych wraz z aplikacją pochodzą z tabeli osadzonej na stronie devernay.free.fr [6];

```
void saveMaterial(Material mat, const std::string &matName, bool edit)
{
        std::cout << "MATERIAL::SAVE_MATERIAL␣";
        std::string matPath = std::string(FILES_PATH) + matName + ↙
            ↳ MATERIAL_FILE_EXTENSION;
        std::cout << matPath << "␣";
        std::ofstream file(matPath);
        if (file.bad() || file.fail())
        {
                std::cout << "FILE_BAD" << std::endl;
                return;
        }
        file << mat.name << std::endl;
        file << mat.ambient[0] << "␣" << mat.ambient[1] << "␣" << ↙
            ↳ mat.ambient[2] << std::endl;
        file << mat.diffuse[0] << "␣" << mat.diffuse[1] << "␣" << ↙
            ↳ mat.diffuse[2] << std::endl;
        file << mat.specular[0] << "␣" << mat.specular[1] << "␣" << ↙
            ↳ mat.specular[2] << std::endl;
        file << mat.shininess;
        file.close();
        ...
}
```

Listing 4.2: Fragment kodu funkcji zapisującej właściwości materiału do pliku tekstowego

```
obsidian                                    // Material name
0.05375 0.05 0.06625                        // R G B Ambient
0.18275 0.17 0.22525                        // R G B Diffuse
0.332741 0.328634 0.346435                  // R G B Specular
0.3                                         // Shininess
```

Listing 4.3: Przykład pliku .mat wraz z opisem pól

4.1.2 Model 3D

Drugą strukturą danych przechowywaną na dysku jest model 3D. Wszystkie pliki obiektów, które zostaną zapisane z poziomu aplikacji, będą posiadały rozszerzenie `.vxl` stworzone na potrzeby niniejszej pracy. Plik tekstowy `.vxl` tworzony jest przy zapisie postępów pracy do pliku, poprzez wywołanie metody „save” (przedstawionej na listingu 4.4) klasy „Object”. Format pliku `.vxl` został przedstawiony na listingu 4.5 wraz z opisem.

```
void Object::Save()
{
    std::cout << "OBJECT::SAVE␣" << std::string(FILES_PATH) + name + ↙
        ↳ std::string(VOXEL_FILE_EXTENSION) << "␣";
    std::ofstream file(std::string(FILES_PATH) + name + ↙
        ↳ std::string(VOXEL_FILE_EXTENSION));
    if (file.bad() || file.fail())
    {
        std::cout << "FILE_BAD" << std::endl;
        return;
    }
    for (Voxel voxel : m_voxels)
    {
        file << voxel.pos.x << "␣" << voxel.pos.y << "␣" << voxel.pos.z ↙
            ↳ << "␣" << voxel.mat.name << std::endl;
    }
    file.close();
    std::cout << std::endl;
    return;
}
```

Listing 4.4: Fragment kodu metody klasy „Object” zapisującej obiekt do pliku tekstowego

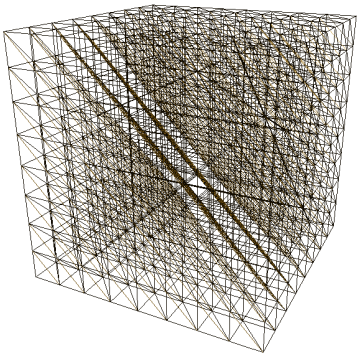
```
...
1 -3 3 pearl                              // x y z .mat
0 -2 3 obsidian                           // x y z .mat
...
2 0 -1 bronze                             // x y z .mat
...
```

Listing 4.5: Fragment pliku .vxl wraz z opisem pól

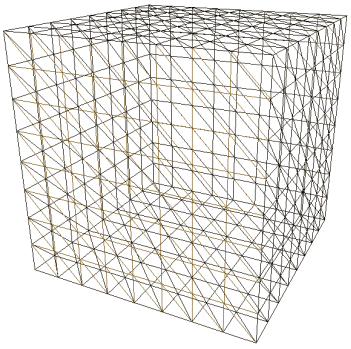
4.2 Implementacja wybranych funkcjonalności

4.2.1 Optymalizacja rysowania obiektu

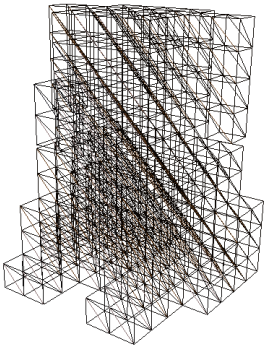
W celu rozwiązania problemu rysowania wokseli w środku obiektu, został użyty prosty algorytm 1. Polega on na iteracji poprzez wszystkie woksele i sprawdzenie na podstawie mapy haszującej, czy sąsiad istnieje. W przypadku istnienia, nie rysujemy tej ścianki, gdyż nie będzie widoczna. Efekt algorytmu ukazany został na rysunkach 4.2.



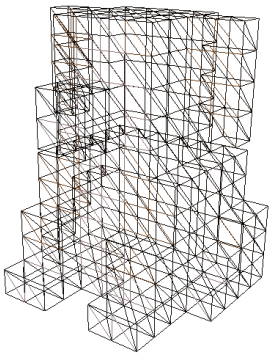
(a) model kostki 8x8x8 przed optymalizacją



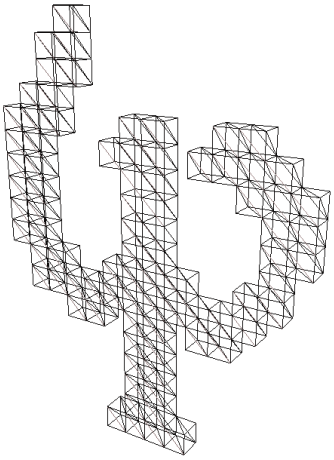
(b) model kostki 8x8x8 po optymalizacji



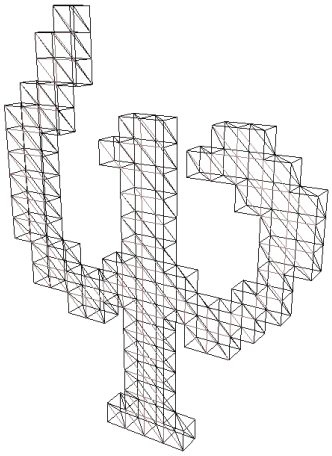
(c) model psa przed optymalizacją



(d) model psa po optymalizacji



(e) model loga przed optymalizacją



(f) model loga po optymalizacji

Rysunek 4.2: Prezentacja optymalizacji modeli szkieletowych, źródło: opracowanie własne

Algorytm 1: Algorytm usuwający sąsiadujące ścianki

for $voxel \in voxels$ **do**
  for $face \in voxel.faces$ **do**
    if $voxel\ is\ not\ adjacent\ to\ the\ face.position\ voxel$ **then**
      $draw\ face$
    end if
  end for
end for

4.2.2 Interakcja z obiektem

Najważniejszą funkcjonalnością do implementacji była interakcja z obiektem. Przyciśnięcie lewego przycisku myszy (`GLFW_MOUSE_BUTTON_LEFT` i `GLFW_PRESS`) w oknie wygenerowanym przez bibliotekę GLFW, pozwala na obliczenie kierunku promienia mającego początek w miejscu kamery. W celu obliczenia tego promienia, zastosowana została metoda „getRayCast" [11], której zadaniem jest przekształcenie punktu 2D z przestrzeni rzutni (ang. *viewport space*) do promienia 3D w przestrzeni świata (ang. *world space*).

```
if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_PRESS)
    {
      glm::vec3 ray_origin = voxelGame->camera->Position;
      glm::vec3 ray_dir = ↙
          ↳ voxelGame->getRayCast(voxelGame->mvp.projection, ↙
          ↳ voxelGame->mvp.view);
      glm::vec3 newBlockLoc = glm::vec3(0.f);
      Voxel *t_voxel = voxelGame->object->CheckRay(ray_origin, ray_dir, ↙
          ↳ newBlockLoc);

      ...
    }
```

Listing 4.6: Fragment kodu klasy „VoxelGame" odpowiedzialnego za obsługę lewego przycisku myszy

W celu określenia intersekcji promienia z wokselem, w klasie „Object" zaimplementowano metodę „CheckRay" zmodyfikowaną na potrzeby tego testu przecięcia z artykułu „An Efficient and Robust Ray-Box Intersection Algorithm" [26]. Modyfikacja ta dodała możliwość obliczenia nie tylko pozycji woksela, ale też i jego ścianki. W zależności od trybu edycji użytkownika, w przypadku intersekcji z wokselem, wykonywane są następujące metody (przedstawione również na listingu 4.7):

- „ChangeColor" (implementacja w sekcji „Zmiana materiału woksela") służąca do zmiany koloru woksela.

- „RemoveVoxel” (implementacja w sekcji „Usunięcie woksela”) służąca do usunięcia woksela.
- „AddVoxel” (implementacja w sekcji „Dodanie woksela”) służąca do postawienia woksela na ściance obliczonej przez „CheckRay”.

```
if (t_voxel)
      {
        if (voxelGame->stateHandler->GetColorMode())
          voxelGame->object->ChangeColor(t_voxel, ↙
              ↳ loadMaterial(voxelGame->activeMaterialName));
        if (voxelGame->stateHandler->GetRemoveMode())
          voxelGame->object->RemoveVoxel(t_voxel);
        if (voxelGame->stateHandler->GetAddMode())
        {
          voxelGame->object->AddVoxel(t_voxel->pos + newBlockLoc, ↙
              ↳ loadMaterial(voxelGame->activeMaterialName));
        }
      }
```

Listing 4.7: Dalszy fragment kodu z listingu 4.6

4.2.3 Dodanie woksela

Najczęściej używaną funkcjonalnością przez użytkownika będzie dodawanie wokseli. Odbywa się to poprzez przekazanie pozycji woksela docelowego oraz właściwości materiału. Zanim woksel zostanie dodany do modelu 3D, sprawdzane jest wpierw, czy nie wykracza poza ustalone ograniczenia, jak i czy woksela nie ma już w tym miejscu (w przypadku dodania manualnego). Gdy już pozycja woksela przejdzie pozytywnie weryfikacje, zostanie ona dodana do modelu. Implementacja tej funkcjonalności została przedstawiona na listingu 4.8.

```
void Object::AddVoxel(glm::ivec3 pos, Material mat)
{
    glm::ivec3 t_pos = glm::ivec3(VOXEL_COUNT / 2 + pos.x, VOXEL_COUNT ↙
        ↳ / 2 + pos.y, VOXEL_COUNT / 2 + pos.z);
    if (t_pos.x < 0 || t_pos.x > VOXEL_COUNT)
    {
        std::cout << "OBJECT::ADD_VOXEL::POS::X␣Out␣of␣bounds␣" << ↙
            ↳ std::endl;
        return;
    }
    if (t_pos.y < 0 || t_pos.y > VOXEL_COUNT)
    {
        std::cout << "OBJECT::ADD_VOXEL::POS::Y␣Out␣of␣bounds␣" << ↙
            ↳ std::endl;
        return;
    }
    if (t_pos.z < 0 || t_pos.z > VOXEL_COUNT)
    {
        std::cout << "OBJECT::ADD_VOXEL::POS::Z␣Out␣of␣bounds␣" << ↙
            ↳ std::endl;
```

```
        return;
    }
    if (m_hashVoxels[t_pos.x][t_pos.y][t_pos.z])
    {
        std::cout << "OBJECT::ADD_VOXEL␣Voxel␣already␣here" << std::endl;
        return;
    }
    Voxel t_voxel;
    t_voxel.pos = pos;
    t_voxel.mat = mat;
    m_voxels.push_back(t_voxel);
    m_hashVoxels[t_pos.x][t_pos.y][t_pos.z] = true;
    std::cout << "OBJECT::ADD_VOXEL␣(" << t_voxel.pos.x << ",␣"
              << t_voxel.pos.y << ",␣" << t_voxel.pos.z << ")␣("
              << t_voxel.mat.name << ")" << std::endl;
}
```

Listing 4.8: Fragment kodu klasy „Object” odpowiedzialnego za dodawanie woksela

4.2.4 Usunięcie woksela

Z uwagi na zwrócenie wskaźnika do woksela poprzez funkcję sprawdzającą promień pochodzący z kamery, implementacja usunięcia woksela jest prostolinijna. Polega ona na ustawieniu `false` w tablicy haszującej oraz usunięciu obiektu „Voxel” na podstawie adresu zwróconego woksela.

```
void Object::RemoveVoxel(Voxel *voxel)
{
    glm::ivec3 t_pos = glm::ivec3(VOXEL_COUNT / 2 + voxel->pos.x, ↙
        ↳ VOXEL_COUNT / 2 + voxel->pos.y, VOXEL_COUNT / 2 + ↙
        ↳ voxel->pos.z);
    m_hashVoxels[t_pos.x][t_pos.y][t_pos.z] = false;
    m_voxels.erase(m_voxels.begin() + (voxel - &m_voxels.front()));
}
```

Listing 4.9: Fragment kodu klasy „Object” odpowiedzialnego za usunięcie woksela

4.2.5 Zmiana materiału woksela

Analogicznie jak w przypadku Usunięcie woksela, edycja właściwości woksela sprowadza się do użycia wskaźnika przekazanego do funkcji. Posiadając adres woksela, zmieniamy wartość pola na specyfikację podanego materiału przez użytkownika.

```
void Object::ChangeColor(Voxel *voxel, Material mat)
{
    voxel->mat = mat;
}
```

Listing 4.10: Fragment kodu klasy „Object” odpowiedzialnego za zmianę materiału woksela

4.3 Testowanie aplikacji

Aplikacja została też przetestowana pod względem działania. Z uwagi na postawione wymagania dotyczące responsywności, edytor zbudowany został z wbudowanymi narzędziami do mierzenia wydajności (okno „Debug"), jak i debugowania (konsola). Testowanie aplikacji podzielono na 3 części:

1. Testowanie przez autora w trakcie budowania aplikacji - W tej fazie, głównym motywem testowania było, profilowanie aplikacji przy użyciu narzędzi do mierzenia wydajności. Zostało zauważone, że w przypadku sprzętu o niższej wydajności, stworzenie dużego modelu 3D w znaczny sposób spowalniało działanie aplikacji. Problem został rozwiązany przy użyciu zaktualizowanego algorytmu rysowania wokseli 1.

2. Testowanie przez osoby potencjalnie korzystające z wbudowanego silnika 3D powstałego w niniejszej pracy w innych aplikacjach - Ta faza poświęcona była funkcjonalnościom wejścia-wyjścia. Przetestowano między innymi formaty zapisanych danych przez aplikację, pod kątem czytelności przez człowieka, jak i komputer. Przetestowano prędkość renderowania dla małych, jak i dużych obiektów 3D.

3. Testowanie przez osoby używające edytorów graficznych - W tej części styczność z aplikacją miały osoby korzystające z programów graficznych na co dzień. Do tej grupy wliczają się uczniowie szkół plastycznych i graficznych oraz studenci kierunków graficznych i gier komputerowych. W tej fazie testowania, założeniem było uzyskanie informacji na temat brakujących funkcjonalności według osób testujących. Zebrane uwagi, zostaną uwzględnione w kolejnych wersjach edytora.

5. Dokumentacja techniczna

Dokumentacja techniczna aplikacji stanowi ważny element użytkownika do nawigacji po systemie. Zapewnia ona wprowadzenie do wyglądu aplikacji, jak i opisuje w jaki sposób osiągnąć pożądany efekt w aplikacji.

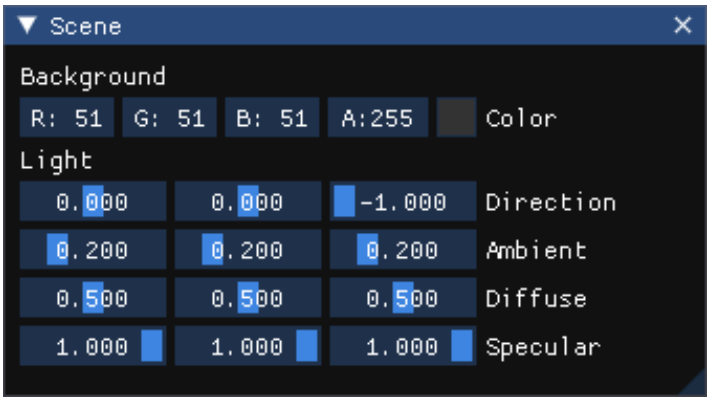
5.1 Prezentacja systemu

Głównym założeniem interfejsu była prostota i customizowalność. Osiągnięte to zostało poprzez wyeksponowanie modelu, który jest renderowany w czasie rzeczywistym przez silnik 3D edytora.

Rysunek 5.1: Ekran startowy programu, źródło: opracowanie własne

5.1.1 Okno „Scene”

By użytkownikowi dać więcej możliwości ustawienia wyglądu wyjściowego, dodano opcję zmiany wartości oświetlenia, jak i tła aplikacji. Wygląd okna „Scene” został ukazany na rysunku 5.2.



Rysunek 5.2: Okno zmiany ustawień sceny, źródło: opracowanie własne

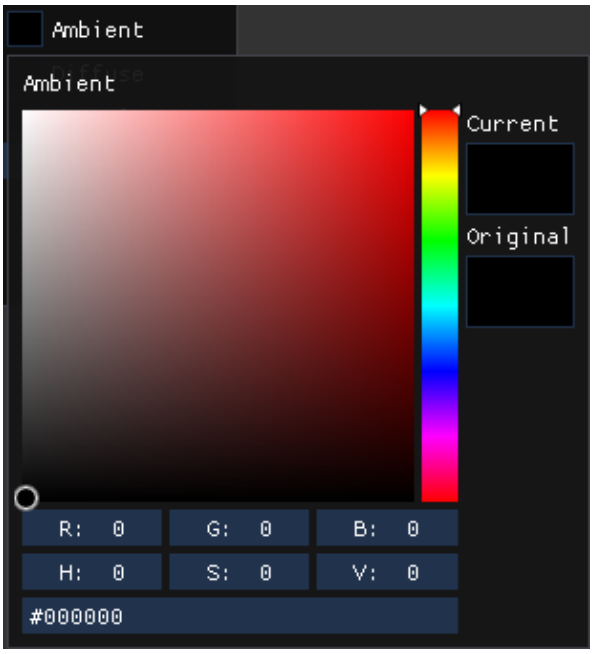
5.1.2 Okno „Material”

Zmiana aktywnego materiału odbywa się poprzez wybór z listy dostępnych, jak i stworzonych materiałów w oknie „Material” (rysunek 5.3).

Rysunek 5.3: Okno zmiany bieżącego materiału, jak i jego edycji, źródło: opracowanie własne

5.1.3 Podgląd wybranego koloru

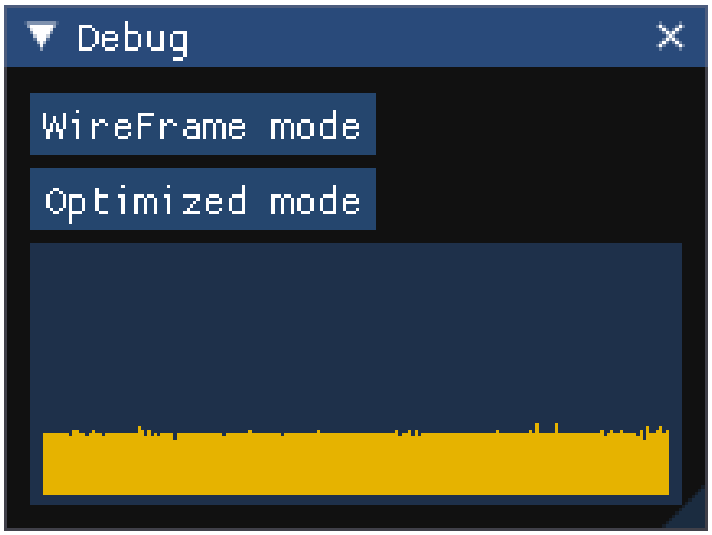
W celu ułatwienia wyboru koloru podczas tworzenia materiału oraz wyboru tła sceny, parametry „Color” w przypadku okna „Scene”, jak i „Ambient”, „Diffuse” i „Specular” w przypadku okna „Material” posiadają opcję wyboru ręcznego z kwadratu kolorów (rysunek 5.4).



Rysunek 5.4: Okno wyboru wartości koloru wraz z podglądem po prawej stronie, źródło: opracowanie własne

5.1.4 Okno „Debug”

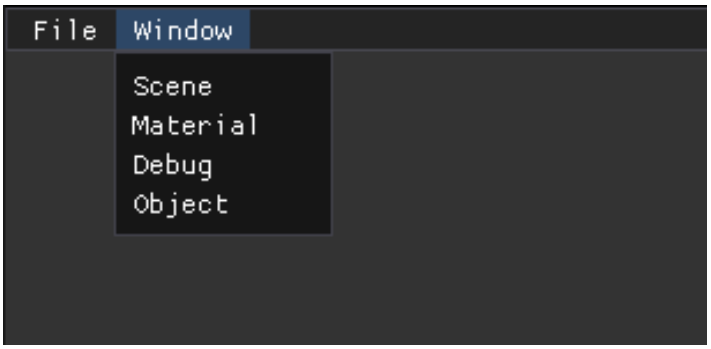
Okno „Debug” (rysunek 5.5) powstało specjalnie z uwagą na wymagania czasowe dotyczące głównych funkcjonalności. Przycisk „WireFrame mode” przełącza w silniku 3D sposób renderowania obiektów na tylko krawędzie, zaś przycisk „Optimized mode” pozwala na wyłączenie zoptymalizowanego algorytmu rysowania modelu. Pod przyciskami w czasie rzeczywistym kreślony jest wykres czasu renderowania pojedynczej klatki (ang. *frame time*).



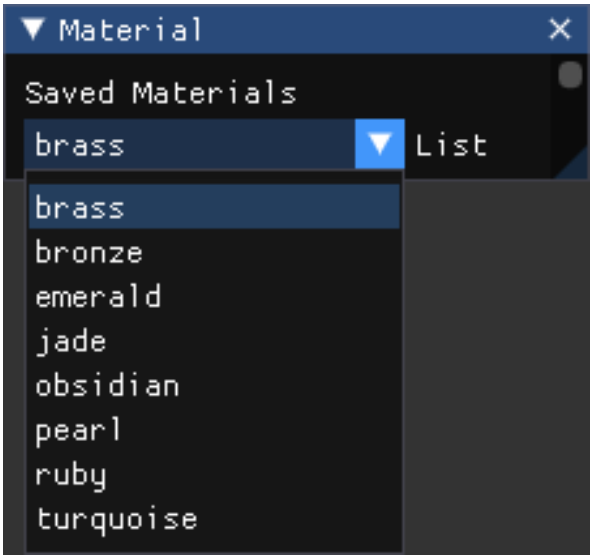
Rysunek 5.5: Okno z informacjami o działaniu silnika graficznego, źródło: opracowanie własne

5.1.5 Zarządzanie oknami

Przedstawione powyżej okna aplikacji, są dostępne z poziomu paska nawigacji (rysunek 5.6) za pomocą którego można je pokazywać lub chować. Dla każdego z tych okienek, użytkownik może zmienić ich pozycję, wielkość, jak i zminimalizować (rysunek 5.8), według własnego uznania. Na rysunku 5.7 przedstawione zostało zmniejszone okno „Material" z rysunku 5.3.



Rysunek 5.6: Pasek nawigacyjny z wybraną opcją „Window", źródło: opracowanie własne



Rysunek 5.7: Okno „Material" ze zmienionym rozmiarem przez użytkownika, źródło: opracowanie własne

Rysunek 5.8: Okno „Material" w wersji zminimalizowanej, źródło: opracowanie własne

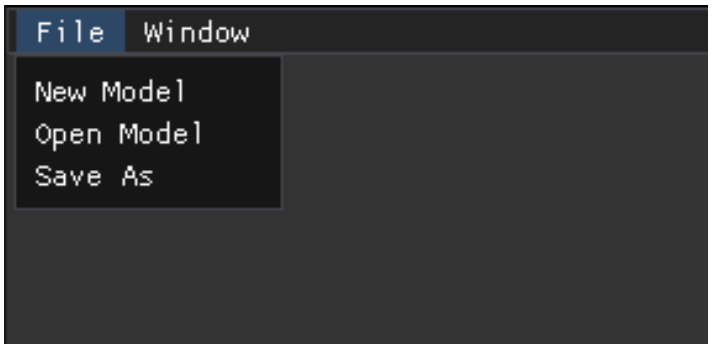
5.1.6 Okno „Edit Modes”

Głównym oknem do zmiany trybów interakcji z modelem 3D jest „Edit Modes” ukazany na rysunku 5.9. Służy ono do zmiany głównych trybów edycji na modelu 3D. Przez zaznaczenie jednej opcji z „Add”, „Remove” lub „Color”, kliknięcie na istniejący model na ekranie będzie miał inny rezultat.

Rysunek 5.9: Okno trybu edycji modelu 3D, źródło: opracowanie własne

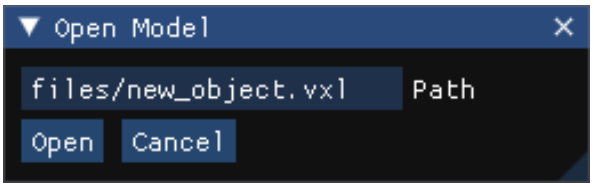
5.1.7 Okna manipulacji plikami

Pasek z rysunku 5.6 posiada jeszcze jedną opcję, a mianowicie „File”, odpowiedzialną za wczytywanie modeli z pliku, zapisywanie ich oraz tworzenie nowych. Opcję tej zakładki są przedstawione na rysunku 5.10

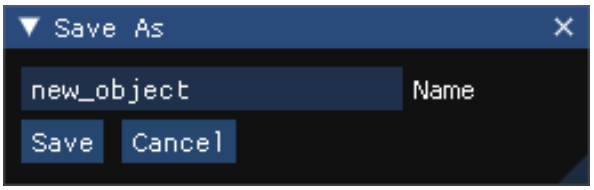


Rysunek 5.10: Pasek nawigacyjny z wybraną opcją „File”, źródło: opracowanie własne

W przypadku opcji „Open Model” przedstawionej na rysunku 5.11 oraz „Save As” na rysunku 5.12, użytkownik wprowadza ścieżkę względną do pliku w celu jego wczytania, bądź w przypadku opcji drugiej, jego nazwę. Opcja „New Model” powoduje usunięcie wszelkich postępów pracy nad aktualnym obiektem, by przywrócić stan początkowych modelu 3D. Operacja ta jest niemożliwa do cofnięcia.



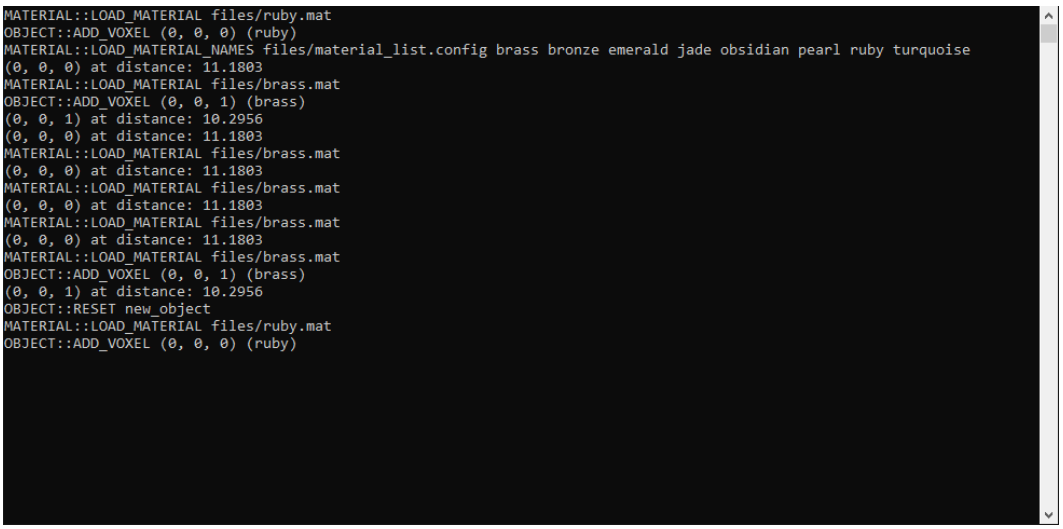
Rysunek 5.11: Okno wczytania modeli do edytora, źródło: opracowanie własne



Rysunek 5.12: Okno zapisu aktualnego modelu 3D do pliku, źródło: opracowanie własne

5.1.8 Konsola

W tymi miejscu, znajdują się wszystkie informacje na temat czynności użytkownika w aplikacji. Głównym założeniem konsoli jest możliwość określenia przyczyny krytycznego wyłączenia edytora. Narzędzie te jest uruchamiane i wyłączane wraz z edytorem.



Rysunek 5.13: Konsola aplikacji, źródło: opracowanie własne

5.2 Instrukcja użytkownika

W pełni zbudowana aplikacja w wersji przygotowanej w ramach niniejszej pracy jest dostępna pod adresem `https://github.com/DuDuSteo/VoxelGameEngine/releases/tag/v1.0.0` [7]. Po pobraniu pliku `VoxelEditor_v1.0.0_WIN_64.zip`, wystarczy tylko wyodrębnić zawartość.

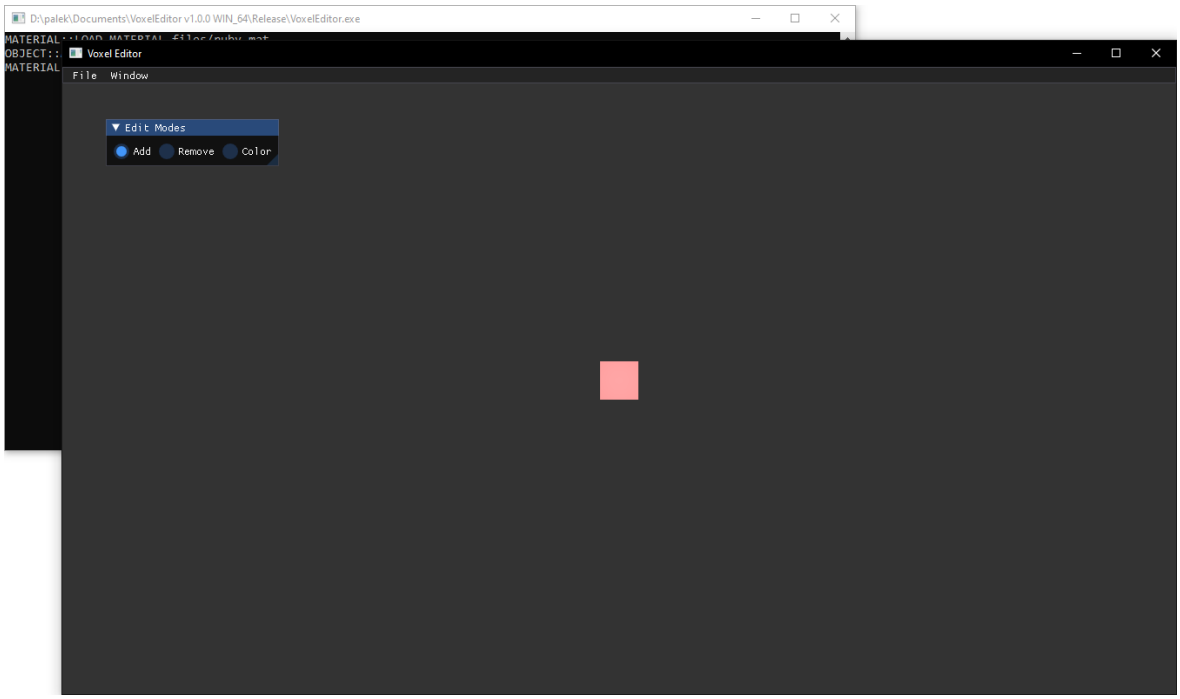
Edytor widnieje pod nazwą `VoxelEditor.exe` w przypadku systemów `Windows`. Aplikacja przyjęła standardowy wśród edytorów graficznych model poruszania się kamerą:

- `Środkowy przycisk myszy + Ruch kursorem` - zmiana kąta kamery.
- `Środkowy przycisk myszy + Shift + Ruch kursorem` - zmiana pozycji kamery.

Interakcja z modelem odbywa się poprzez `lewy przycisk myszy`.

5.2.1 Uruchomienie programu

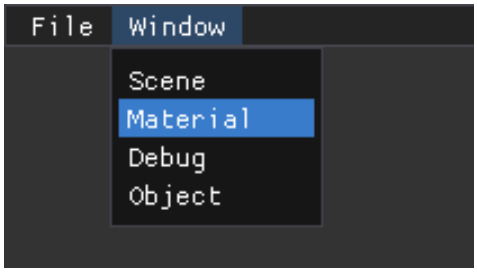
Aby uruchomić program, należy dwukrotnie nacisnąć lewym przyciskiem myszy na `VoxelEditor.exe`. Po uruchomieniu programu użytkownik zostaje przywitany ekranem startowym aplikacji przedstawionym na rysunku 5.14. Wszystkie ukryte okna, przedstawione w podrozdziale „Prezentacja systemu" znajdują się w `Pasek narzędziowy -> Window`.



Rysunek 5.14: Edytor modeli po uruchomieniu wraz z konsolą, źródło: opracowanie własne

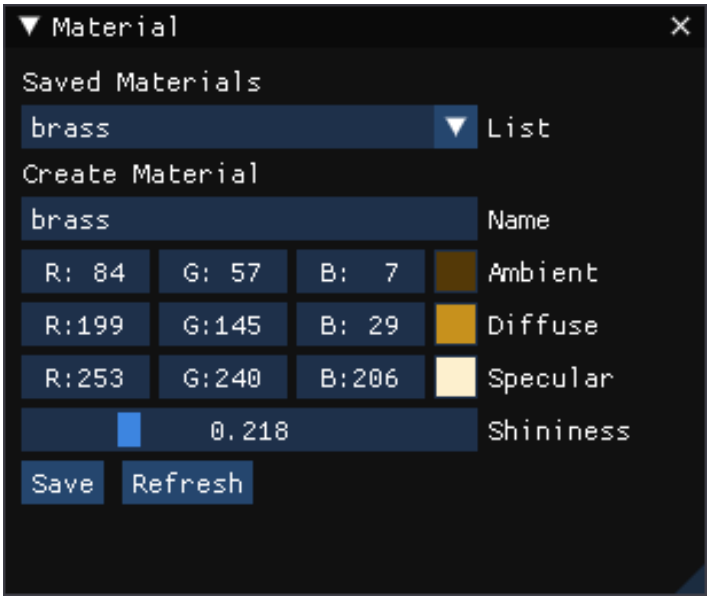
5.2.2 Dodanie nowego materiału

Dodanie nowego materiału do listy materiałów w aplikacji, odbywa się poprzez okno „Material”. W podstawowym widoku edytora, by otworzyć okno materiałów, należy posłużyć się paskiem narzędziowym, wybierając w nim `Window -> Material` jak na rysunku 5.15.

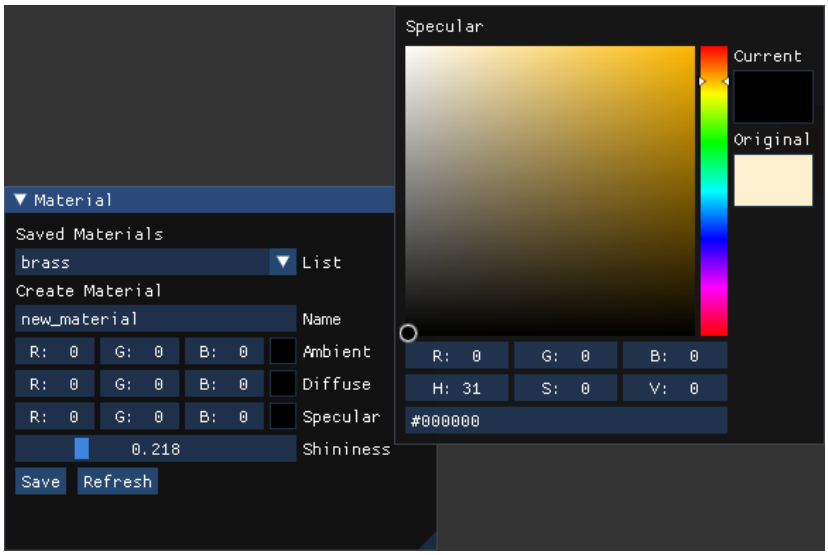


Rysunek 5.15: Wybór okna „Material” z paska narzędziowego, źródło: opracowanie własne

Specyfikacja materiałów opiera się na implementacji w silniku cieniowania Phonga (ang. *Phong shading*) [2]. Po wyborze nazwy, użytkownik wypełnia danymi pola odpowiedzialne za odbicie światła otoczenia (ang. *ambient*), rozproszonego (ang. *diffuse*) oraz odbijanego zwierciadlanie (ang. *specular*) (rysunek 5.16). Możliwe jest też wybranie koloru z okna kolorów, które jest widoczne po kliknięciu w kwadrat przedstawiony na rysunku 5.17.



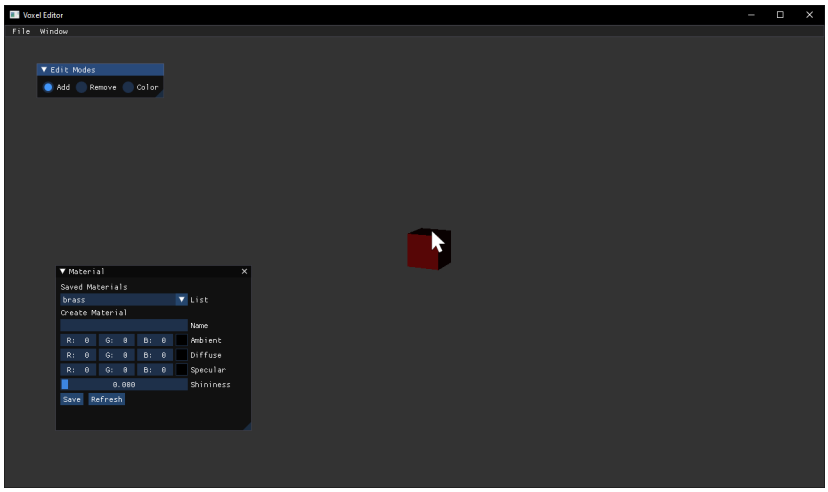
Rysunek 5.16: Prezentacja właściwości materiału w oknie „Material”, źródło: opracowanie własne



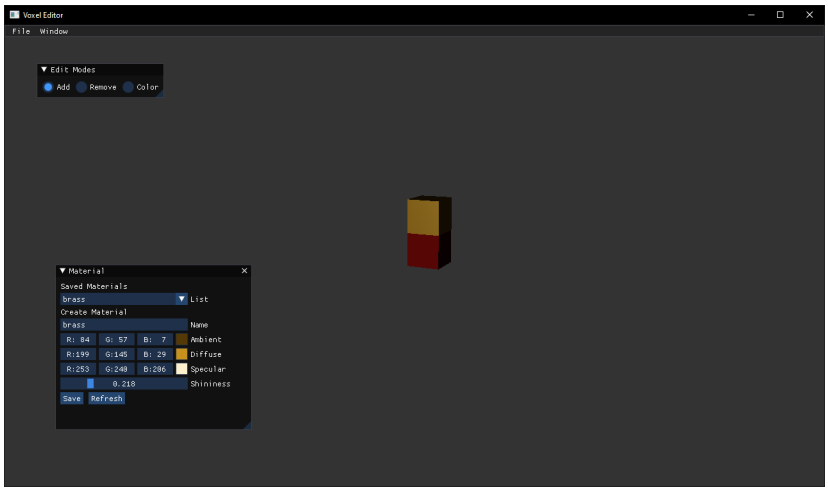
Rysunek 5.17: Okno wyboru koloru z kwadratu kolorów, źródło: opracowanie własne

5.2.3 Dodanie woksela

Aby postawić woksel, użytkownik musi mieć włączony tryb „Add” w oknie „Edit Modes” i wtedy najeżdża kursorem na interesującą go ściankę (rysunek 5.18), w celu postawienia woksela. Nowo utworzony woksel, będzie posiadał wszystkie właściwości aktywnego materiału jak na rysunku 5.19.



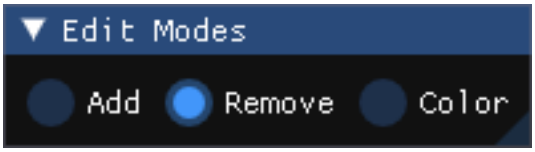
Rysunek 5.18: Kliknięcie na wybraną ściankę, źródło: opracowanie własne



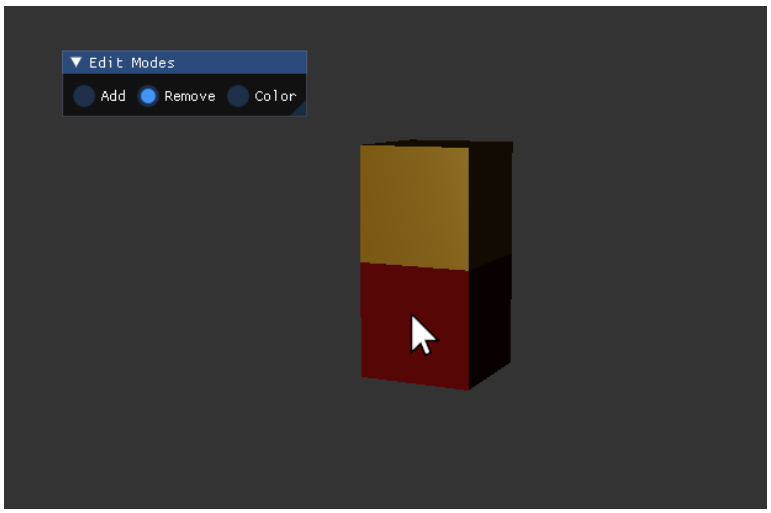
Rysunek 5.19: Wynik interakcji z opcją „Add”, źródło: opracowanie własne

5.2.4 Usunięcie woksela

Zaznaczając opcję „Remove” w oknie „Edit Modes” (rysunek 5.20), wszelkie interakcje z modelem przez użytkownika spowodują usunięcie woksela w miejscu kliknięcia. Proces operacji przedstawiony został na rysunkach 5.21 i 5.22.



Rysunek 5.20: Kliknięcie na wybraną ściankę, źródło: opracowanie własne

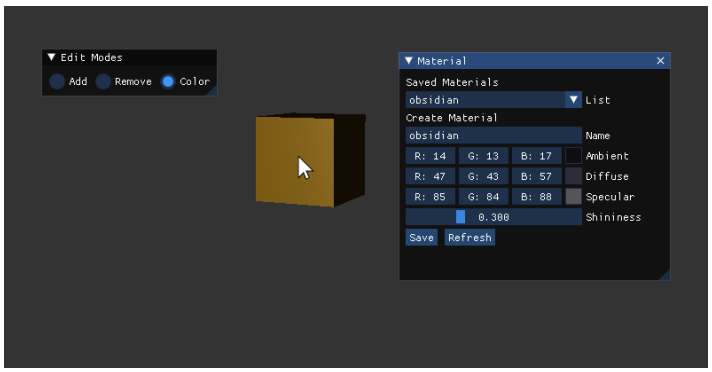


Rysunek 5.21: Kliknięcie na wybranego woksela w celu usunięcia, źródło: opracowanie własne

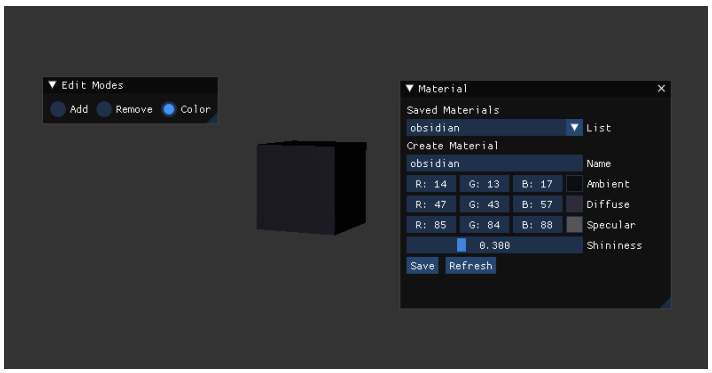
Rysunek 5.22: Wynik interakcji z opcją „Remove”, źródło: opracowanie własne

5.2.5 Zmiana materiału woksela

Ostatnią opcją w oknie „Edit Modes” jest opcja „Color” odpowiadająca za „przemalowanie” kliknięte go woksela na aktywny materiał. Efekt operacji zmiany materiału został pokazany na rysunku 5.23 i 5.24.



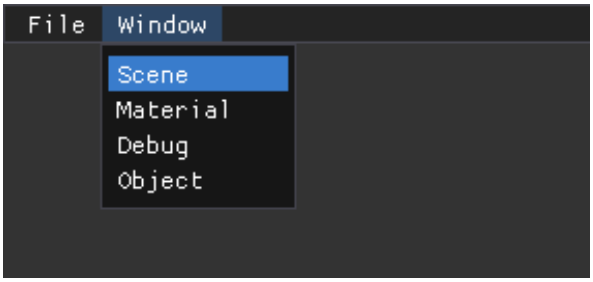
Rysunek 5.23: Kliknięcie na wybranego woksela w celu zmiany materiału, źródło: opracowanie własne



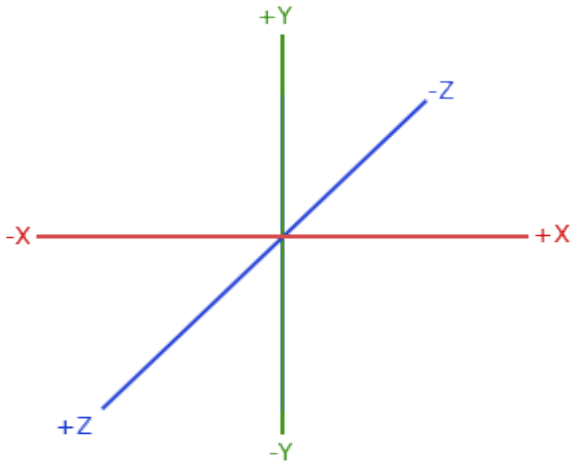
Rysunek 5.24: Wynik interakcji z opcją „Color”, źródło: opracowanie własne

5.2.6 Zmiana oświetlenia

Do zmiany ustawień oświetlenia służy okno „Scene” dostępne z poziomu paska narzędzi `Window -> Scene` (rysunek 5.25). Należy również podkreślić, że OpenGL korzysta z praworęcznego systemu układu współrzędnych, co oznacza że dodatnia oś z jest skierowana w naszym kierunku [5].

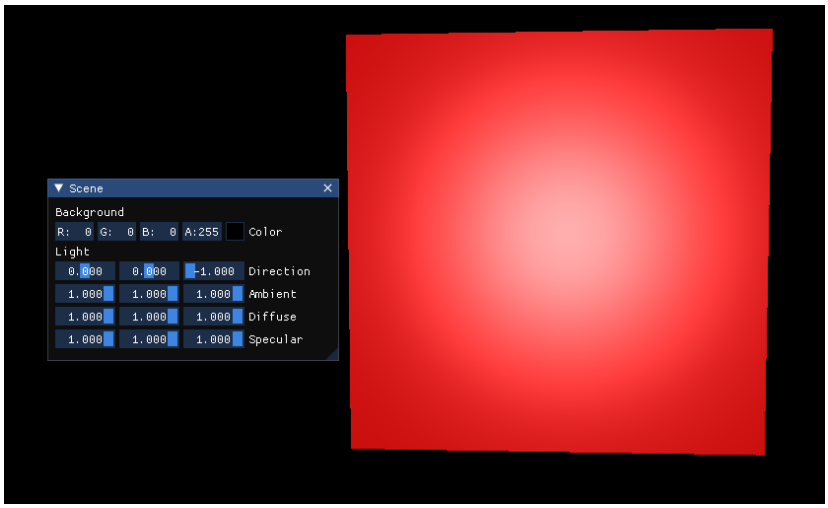


Rysunek 5.25: Wybór okna „Scene” z paska narzędziowego, źródło: opracowanie własne



Rysunek 5.26: Przykład praworęcznego systemu układu współrzędnych, źródło: [5]

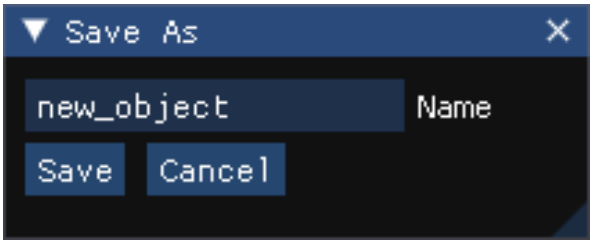
Na rysunku 5.27 przedstawiono efekt oświetlenia, posiadającego kierunek światła o trójwymiarowym wektorze znormalizowanym skierowanym równolegle do kąta patrzenia kamery (x: 0, y: 0, z: -1), jak i parametrach ustawionych na kolor biały dla światła otoczenia (ang. *ambient*), rozproszonego (ang. *diffuse*) oraz odbijanego zwierciadlanie (ang. *specular*). Zmiany na parametrach w oknie „Scene” aktualizowane są na silniku 3D w czasie rzeczywistym.



Rysunek 5.27: Przykładowe ustawienia oświetlenia przy użyciu okna „Scene”, źródło: opracowanie własne

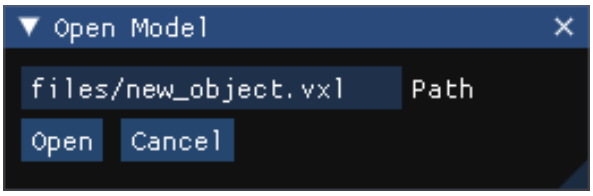
5.2.7 Zapis i odczyt

Zapis i odczyt odbywa się poprzez okna z listy rozwijanej „File” w pasku narzędziowym. W celu zapisania efektów pracy na modelem 3D, należy wybrać w pasku narzędziowym `File -> Save As`. Po kliknięciu „Save As”, w aplikacji pojawi się okno odpowiedzialne za zapis pliku (rysunek 5.28). Użytkownik wpisuje wybraną nazwę pliku w polu tekstowym i zatwierdza przyciskiem „Save”. Stworzony plik znajduje się w folderze `./files/` z rozszerzeniem `.vxl`.



Rysunek 5.28: Okno wyboru nazwy pliku do zapisu, źródło: opracowanie własne

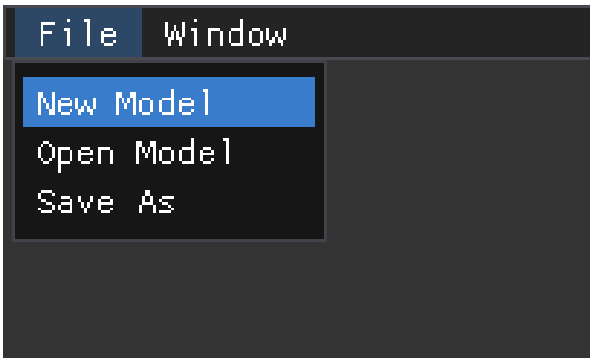
Analogicznie odczyt odbywa się poprzez otworzenie okna „Open Model”, znajdującego się `File -> Open Model`, jak na przedstawionym rysunku 5.29. W przypadku operacji wczytania z pliku, użytkownik przekazuję w polu tekstowym ścieżkę i rozszerzenie, aniżeli nazwę pliku.



Rysunek 5.29: Okno wyboru ścieżki pliku do odczytu, źródło: opracowanie własne

5.2.8 Tworzenie nowego modelu 3D

Aby stworzyć nowy model, bądź usunąć aktualne efekty pracy w edytorze, należy wybrać w pasku narzędziowym `File -> New Model` przedstawioną na rysunku 5.30. Kliknięcie opcji „New Model” usunie wszystkie aktualne postępy pracy i przywróci model 3D do stanu początkowego.



Rysunek 5.30: Przycisk stworzenia nowego modelu 3D w pasku narzędziowym, źródło: opracowanie własne

Podsumowanie

Tematem niniejszej pracy było stworzenie aplikacji pozwalającej na kreację modeli 3D opartych o woksele. Część teoretyczna opierała się na zapoznaniu z podstawowymi funkcjonalnościami istniejących już rozwiązań, jak i działaniem tych aplikacji graficznych, na podstawie których powstał projektu systemu.

W ramach pracy powstał prosty silnik 3D posiadający podstawowe funkcjonalności potrzebne do edycji modelów 3D, takie jak dodawanie nowych obiektów, usuwanie ich, oraz zmiana ich właściwości. Dostęp do tych funkcjonalności odbywa się poprzez interfejs okienkowy, pełniący role narzędzia do edycji modelu. W ten sposób powstała aplikacja pozwoliła na osiągnięcie w pełni celu niniejszej pracy, którym było stworzenie edytora modeli 3D opartych o woksele.

Wykonanie niniejszej pracy pozwoliło wyciągnąć szereg wniosków. Analiza istniejących rozwiązań na rynku w znaczny sposób ułatwiła określenie wymagań stawionych w projekcie systemu. Zaprojektowanie rozwiązania przed rozpoczęciem implementacji ułatwiło późniejszy proces tworzenia aplikacji, potwierdzając istotę przygotowania projektu systemu. Dokumentacja techniczna pozwoliła osobom odpowiedzialnym za testowanie na łatwą konfigurację i nawigację po oprogramowaniu.

Mimo, że aplikacja pozwala na tworzenie i edycję modeli 3D, spełniając jednocześnie cel pracy, nie można wykluczyć dalszego rozwoju tego rozwiązania. Z uwagi na użycie silnika graficznego, możliwe jest ponowne wykorzystanie silnika w celu stworzenia gry komputerowej. Pozwoli to na jednoczesne posiadanie narzędzia do tworzenia modeli, operujące na tym samym silniku graficznym, co gra komputerowa, pozwalając na łatwe tworzenie obiektów kompatybilnych z grą. Możliwe jest też rozszerzenie funkcjonalności samego edytora, dodając pożądane rozwiązania takie jak możliwość cofnięcia lub przywrócenia ostatniej czynności, czy możliwość wyeksportowania większej ilości parametrów silnika graficznego.

Bibliografia

[1] Tomas Akenine-Moller and Naty Hoffman Eric Haines. *Real-Time Rendering, Fourth Edition*. CRC Press, s. 578, 2018.

[2] Sam Bus and Samuel R. Buss. *3D Computer Graphics: A Mathematical Introduction with OpenGL*. Cambridge University Press, s. 69-87, 2003.

[3] G-Truc Creation. OpenGL Mathematics (GLM). `https://glm.g-truc.net/0.9.9/index.html`, stan z 18.02.2021 r.

[4] Dav1dde. glad. `https://glad.dav1d.de/`, stan z 18.02.2021 r.

[5] Joey de Vries. *Learn OpenGL: Learn modern OpenGL graphics programming in a step-by-step fashion*. KW Publishers, s. 50-53, 89, 95-106, 2020.

[6] Frédéric Devernay. OpenGL/VRML Materials. `http://devernay.free.fr/cours/opengl/materials.html`, stan z 18.02.2021 r.

[7] Paweł Aleksiejuk DuDuSteo. 3D model editor based on voxels - diploma thesis release version. `https://github.com/DuDuSteo/VoxelGameEngine/releases`, stan z 16.01.2022 r.

[8] elmindreda. GLFW. `https://www.glfw.org/`, stan z 18.02.2021 r.

[9] ephtracy. MagicaVoxel. `https://ephtracy.github.io`, stan z 04.02.2021 r.

[10] International Organization for Standardization. ISO/IEC 14882:2011 standard. `https://www.iso.org/standard/50372.html`, stan z 18.02.2021 r.

[11] Anton Gerdelan. Mouse Picking with Ray Casting. `https://antongerdelan.net/opengl/raycasting`, stan z 18.02.2021 r.

[12] Minddesk Software GmbH. Qubicle. `https://www.minddesk.com`, stan z 04.02.2021 r.

[13] LLC Go Real Games. Mega Voxels Play. `https://www.megavoxels.com`, stan z 04.02.2021 r.

[14] The Khronos Group. OpenGL Loading Library. https://www.khronos.org/opengl/wiki/OpenGL_Loading_Library, stan z 18.02.2021 r.

[15] The Khronos Group. The OpenGL® Shading Language specification. https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.3.30.pdf, stan z 18.02.2021 r.

[16] guillaumechereau. Goxel. https://goxel.xyz, stan z 04.02.2021 r.

[17] KDE. Doxygeng. https://www.doxygen.nl/index.html, stan z 21.08.2021 r.

[18] Christian Kohler. Path Intellisense extension for Visual Studio Code. https://marketplace.visualstudio.com/items?itemName=christian-kohler.path-intellisense, stan z 18.02.2021 r.

[19] Microsoft. C/C++ extension for Visual Studio Code. https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools, stan z 18.02.2021 r.

[20] Microsoft. CMake Tools extension for Visual Studio Code. https://marketplace.visualstudio.com/items?itemName=ms-vscode.cmake-tools, stan z 18.02.2021 r.

[21] Microsoft. Visual Studio Code. https://code.visualstudio.com, stan z 18.02.2021 r.

[22] ocornut. Dear ImGui. https://github.com/ocornut/imgui, stan z 18.02.2021 r.

[23] Pixowl. VoxEdit Beta. https://www.voxedit.io, stan z 04.02.2021 r.

[24] VSCode Icons Team. vscode-icons extension for Visual Studio Code. https://marketplace.visualstudio.com/items?itemName=vscode-icons-team.vscode-icons, stan z 18.02.2021 r.

[25] twxs. CMake extension for Visual Studio Code. https://marketplace.visualstudio.com/items?itemName=twxs.cmake, stan z 18.02.2021 r.

[26] Amy Williams, Steve Barrus, R. Keith Morley, and Peter Shirley. An efficient and robust ray-box intersection algorithm. *Journal of Graphics GPU and Game Tools*, 10(1):49–54, January 2005.

Spis tabel

Spis rysunków

Spis listingów

Spis algorytmów