AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA INFORMATYKI

PROJEKT INŻYNIERSKI

Narzędzie do wizualizacji siatek trójwymiarowych

Tool for visualization of three-dimensional meshes

Autorzy: *Wojciech Dymek, Katarzyna Głąb, Katarzyna Konieczna, Ewa Marczewska*

Kierunek studiów: *Informatyka*

Opiekun pracy: *dr inż. Tomasz Jurczyk*

Kraków, 2017

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „ Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej „sądem koleżeńskim”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście, samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

<podpis dyplomanta>

Spis treści

Cel i wizja produktu

Celem projektu jest stworzenie narzędzia, które w efektywny sposób wykorzysta możliwości współczesnych kart graficznych dla potrzeb renderowanie siatek trójwymiarowych (powierzchniowych oraz objętościowych). Wizualizacja powinna dotyczyć podstawowych typów struktur geometrycznych (wierzchołek, prosta, ściana oraz czworościan). Narzędzie powinno uwzględniać kryteria wizualizacji takie jak ukrywanie oraz kolorowanie elementów w oparciu o ich jakość geometryczną, współrzędne czy też informacje przekazane wraz z przesłaniem struktur do narzędzia np. numer grupy, do której dany element należy.

Charakterystyka problemu

Wizualizacja siatek trójwymiarowych oraz objętościowych jest nieodłącznym elementem implementacji algorytmów geometrii obliczeniowej. Sprawdzenie poprawności wykonania algorytmu wiąże się z koniecznością wykorzystania zewnętrznego narzędzia dostarczającego możliwości reprezentacji graficznej wyników. Aktualnie dostępne narzędzia tego typu posiadają wiele ograniczeń, głównie wydajnościowych, które wpływają na szybkość oraz komfort pracy programisty.

Opisywane w tym dokumencie narzędzie dostarcza możliwości wydajnej wizualizacji wykonywanego algorytmu bezpośrednio z kodu klienta, również dla siatek o dużej liczbie elementów. Wizualizacja wyniku jest możliwa już podczas kolejnych iteracji algorytmu, co pozwala użytkownikowi na większą kontrolę wykonania programu oraz szybsze wykrycie ewentualnych błędów. Zaawansowani użytkownicy mogą również skorzystać z wielu dodatkowych funkcji takich jak możliwość ukrywania elementów oraz ich odpowiedniego kolorowania, czy też importu i eksportu przesłanych elementów do pliku w formacie .obj, co znacznie ułatwi implementację algorytmu.

Studium wykonalności

Wykonana na etapie projektowania systemu analiza wymagań funkcjonalnych oraz niefunkcjonalnych pozwala określić, że projekt jest wykonalny w ramach dostępnych zasobów ludzkich, czasowych i technologicznych.

Analiza ryzyka

Lista potencjalnych zagrożeń, które zostały wykryte na etapie analizy ryzyka została przedstawiona poniżej.

- problemy z zapewnieniem przenośności aplikacji
- nieznajomość wykorzystywanego narzędzi *Protocol Buffers*
- problemy z zapewnieniem wydajności aplikacji przy przesyłaniu dużej ilości elementów
- błędne lub niewystarczające założenia na etapie projektowania rozwiązania
- zbyt mała ilość czasu na implementację pełnej wymaganej funkcjonalności

Zakres funkcjonalności

W rozdziale przedstawione zostały ustalone z klientem wymagania dotyczące działania narzędzia.

Wymagania funkcjonalne

- Wizualizacja siatek składających się z podstawowych elementów: wierzchołków, krawędzi, trójkątnych ścian oraz czworościanów
- Elementy mogą posiadać dodatkowe atrybuty:
    - liczba całkowita (identyfikator grupy) wpływająca na filtrowanie oraz kolorowanie elementów
    - liczba rzeczywista z zakresu 0 do 1 interpretowana np. jako jakość geometryczna elementu, wpływająca na kolorowanie oraz filtrowanie; wartości z poza zadanego przedziału informują o błędnej wartości parametru
    - etykieta przypisana do elementu
- Udostępniając programistyczny interfejs umożliwiający tworzonym programom przygotowanie zbioru elementów do wizualizacji, przesłanie go do aplikacji wizualizującej (wraz z opisem tekstowym) i wstrzymanie do odpowiedniej reakcji użytkownika.
- Eksport oraz import przesłanych elementów do pliku w formacie .obj
- Zapewnia prezentację statystyk o ilości przesłanych oraz widocznych elementów
- Interfejs programistyczny dostarczony w językach Java, C++ oraz Python
- Możliwość działania w trybie 2D oraz 3D

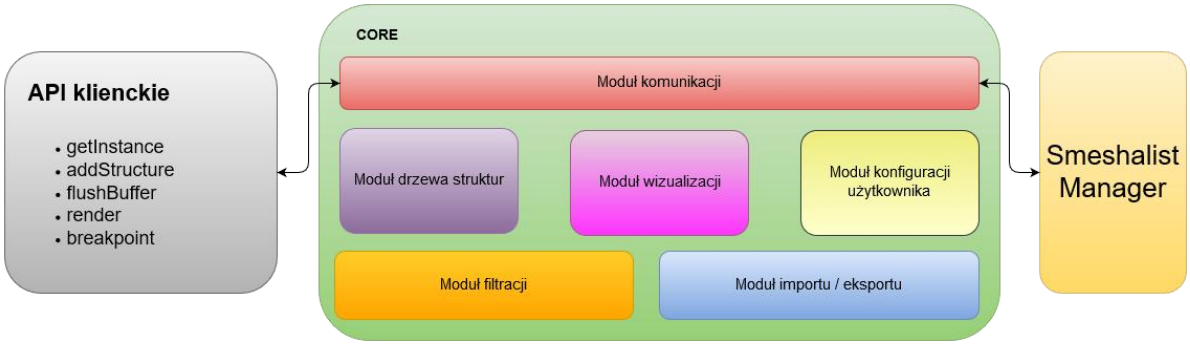
- Kolorowanie elementów siatki w oparciu o współczynnik jakości oraz przynależność elementu do grupy
- Możliwość filtrowania elementów siatki w oparciu o
    - typ elementu
    - jakość elementu
    - przynależność do grupy
    - współrzędne elementów
- Możliwości wizualizacji siatki bieżącej oraz pewnej liczby poprzednio pokazywanych

Wymagania niefunkcjonalne

- Kompatybilność aplikacji na systemie operacyjnym Windows oraz Linux
- Wydajność działania narzędzia w przypadku przesłania dużej liczby elementów
- Łatwa oraz szybka instalacja narzędzia w kodzie użytkownika

Wybrane aspekty realizacji

Architektura aplikacji



Rysunek 1 Architektura aplikacji

Aplikacja została podzielona na następujące moduły zgodnie z powyższym diagramem:

- **Moduł komunikacji CORE** – zapewnia komunikację pomiędzy poszczególnymi komponentami systemu. Do jego głównych zadań należy
  - odbiór wiadomości oraz przekształcenie ich na wewnętrzne obiekty reprezentujące elementy przechowywane w drzewie struktur,
  - przesyłanie wiadomości statystyk do okienka Smeshalist Manager
  - pośredniczenie w filtracji elementów
  - obsługa wiadomości związanych z interakcją pomiędzy systemem a użytkownikiem
- **Moduł struktur CORE -** moduł odpowiedzialny za przechowywanie poszczególnych elementów składających się na siatkę. Dostarcza hierarchiczną strukturę umożliwiającą optymalną wizualizację, filtrację i odpowiednie dodawanie obiektów według rodzajów i grup. Ponadto generuje statystyki odnośnie przechowywanych i widocznych elementów.
- **Moduł filtracji CORE -** odpowiada za przechowywanie oraz zarządzanie zestawami filtrów jak również za filtrację.
- **Moduł wizualizacji CORE –** odpowiada za stworzenie instancji okna, obsługę zdarzeń, głównej pętli programu i sterowanie położeniem kamery.

- **Moduł konfiguracji użytkownika –** jego zadaniem jest odczyt konfiguracji użytkownika dostarczonej w pliku *user.config.xml*. W przypadku braku któregokolwiek parametru konfiguracji ustawiana jest wartość domyślna.
- **Smeshalist Manager –** odpowiada za interakcję z użytkownikiem.

Stos technologiczny

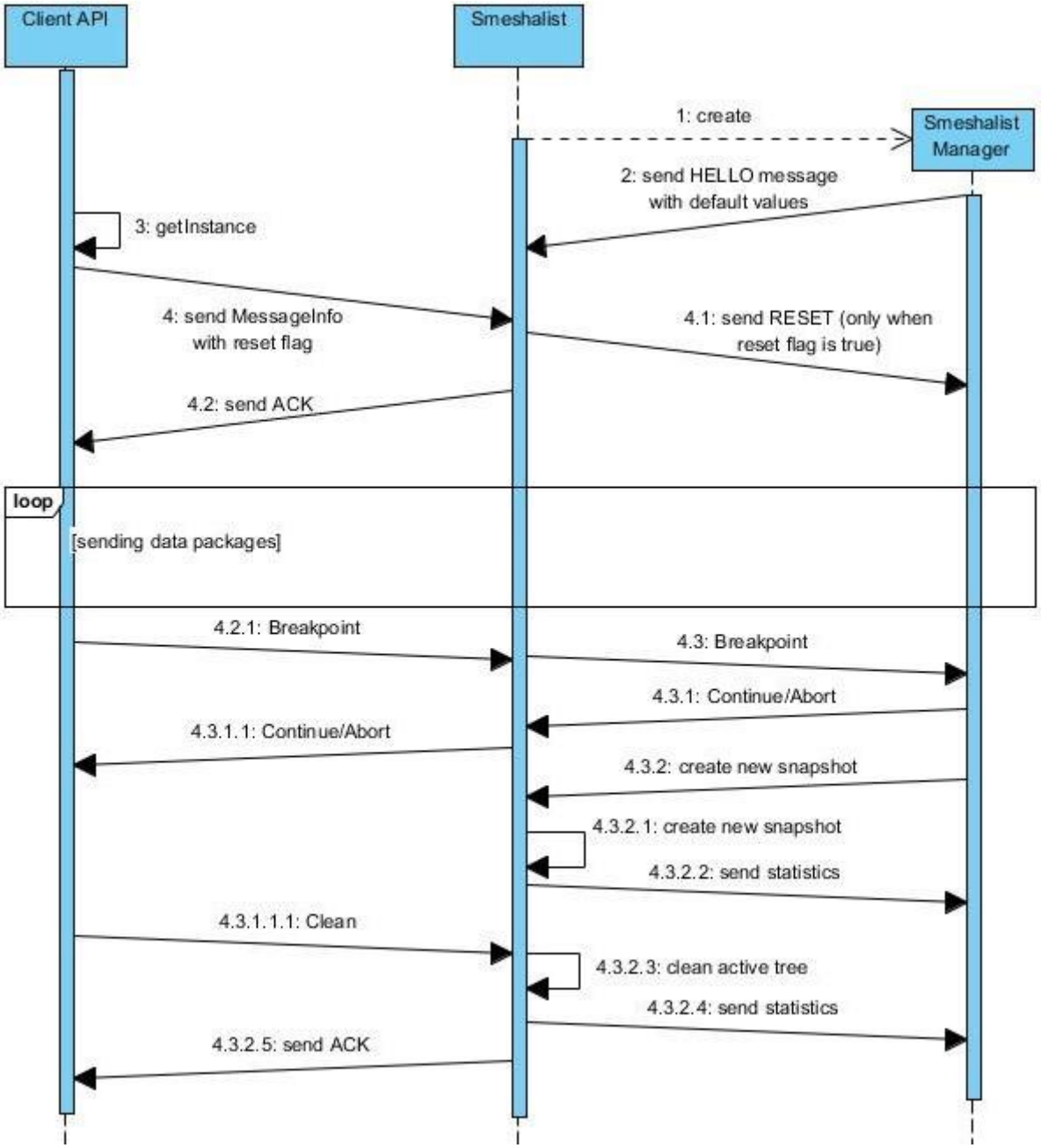
Aplikacja zaimplementowana zostało w języku C++ z wykorzystaniem następujących narzędzi i bibliotek:

- OpenGL
- GLU
- GLUT
- Protocol Buffers

Ponadto dostarczony został interfejs programistyczny w języku Java, Python oraz C++ kompatybilne z platformami Windows i Linux.

Komunikacja wewnątrz systemu

Ze względu na podział systemu na moduły konieczny jest ściśle zdefiniowany sposób komunikacji między nimi. W projekcie wykorzystano w tym celu narzędzie Protocol Buffers, służące do serializacji wiadomości przesyłanych w aplikacji przy pomocy protokołu UDP.



Rysunek 2 Diagram sekwencji przedstawiający przykładowy przebieg komunikacji pomiędzy głównymi składowymi systemu - API klienckim, aplikacją Smeshalist oraz Smeshalist Managerem

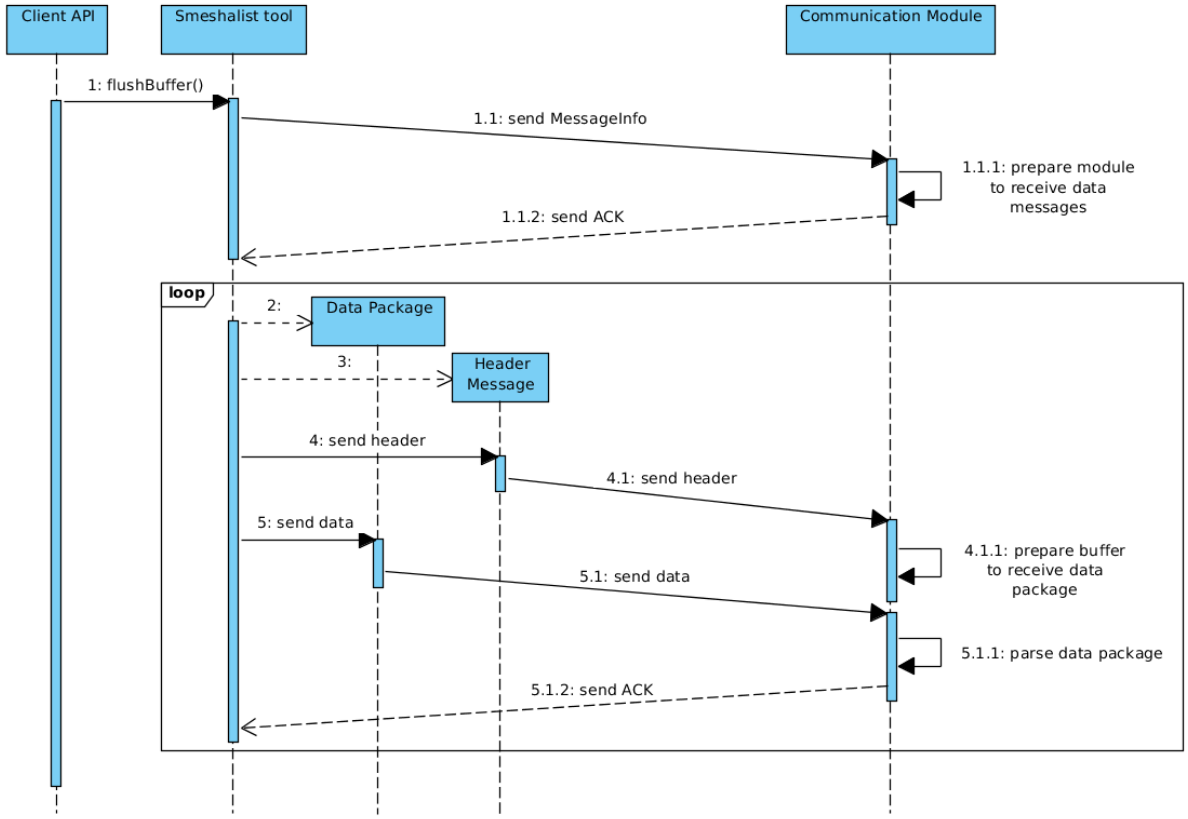
Przebieg komunikacji między modułami

Praca z narzędziem rozpoczyna się od uruchomienia modułu wizualizacji, który odpowiedzialny jest za działanie głównej pętli programu, czyli uruchomienie wszystkich pozostałych modułów CORE oraz okna Smeshalist Manager.

Komunikacja inicjalizowana jest przez Smeshalist Manager zaraz po jego uruchomieniu. Wysyła on wiadomość *ManagerToCoreMessage* z typem ustawionym na *HELLO* oraz zawartością zakładki *Options*, dzięki czemu moduł komunikacji uzyskuje port do komunikacji zwrotnej oraz domyślne ustawienia opcji.

Rozpoczęcie pracy z narzędziem ze strony API klienckiego odbywa się wywołaniem przeładowanej metody *getInstance()*. W tej metodzie do modułu komunikacji wysyłana jest wiadomość *MessageInfo* z typem ustawionym na *HARD_RESET* bądź *NO_RESET* w zależności czy użytkownik chce zresetować zawartość drzewa struktur w module struktur. Moduł komunikacji w odpowiedzi odsyła wiadomość *MessageInfo* z typem ustawionym na *ACK*. Następnie, moduł komunikacji w przypadku otrzymania wiadomości typu *HARD_RESET* wysyła do SmeshalistManager'a wiadomość *CoreToManagerMessage* z typem ustawionym na *HARD_RESET* w celu zresetowania zawartości okna Smeshalist Manager.

Z poziomu API klienckiego dodawane są struktury geometryczne i wysyłane po wywołaniu metody flushBuffer(). Moduł API klienckiego wysyła *MessageInfo* z typem ustawionym na *DATA* do modułu komunikacji, który potwierdza jej otrzymanie wiadomością *MessageInfo* z typem ustawionym na *ACK*. Następnie dane organizowane są w pakiety (wiadomość *DataPackage,* która zawiera listy struktur) i wysyłane do modułu komunikacji. Każdy pakiet poprzedzany jest nagłówkiem (wiadomość *Header*) z polem opisującym wielkość danych (*sizeOfData*) oraz informacją czy pakiet jest ostatnim pakietem z serii (*endOfData*). Po otrzymaniu tego pakietu moduł komunikacji wysyła potwierdzenie otrzymania zestawu danych (wiadomość *MessageInfo* z typem ustawionym na *ACK*).



Rysunek 3 Diagram sekwencji opisujący algorytm przesyłania paczek z danymi do modulu komunikacji CORE

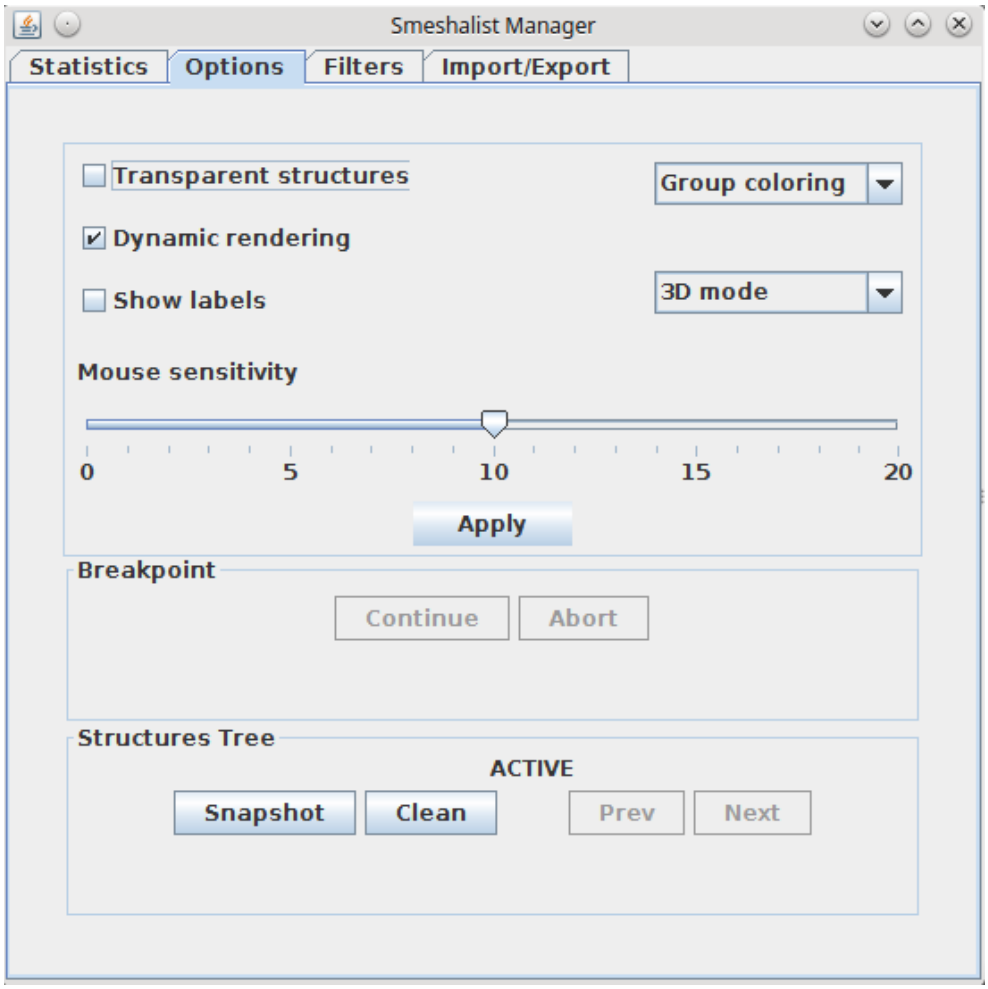
Po dodaniu przesłanych struktur do drzewa struktur, moduł komunikacji wysyła do Smeshalist Menagera wiadomość *CoreToManagerMessage* z typem ustawionym na *STATISTICS* oraz zawartością statystyk *statisticsInfo:*

- liczba elementów w drzewie i wyświetlonych w module wizualizacji
- współrzędne prostopadłościanu ograniczającego obszar roboczy wyświetlanych struktur
- informacje o ID grup wyświetlanych struktur oraz kolorze przypisanym do danej grupy
- nazwa aktualnie wyświetlanego zrzutu drzewa struktur
- flaga *oldestSnapshot* określająca czy wyświetlany aktualnie zrzut drzewa struktur jest najstarszym z przechowywanych zrzutów

W momencie gdy w module wizualizacji są już wyświetlone struktury, z poziomu Smeshalist Manger'a można dokonać ich filtrowania. Po wybraniu filtrów w oknie Smeshalist Manager oraz zatwierdzeniu ich przyciskiem *Apply* następuje wysłanie wiadomości *MangerToCoreMessage* z typem ustawionym na *FILTERS* oraz zawartością filtrów:

- filtr po jakości *QualityFilter*, zawierający listę warunków będących czwórką spełniającą równanie: $a\ [lewy\ operator]\ QUALITY\ [prawy\ operator]\ b$
    - a to wartość ograniczająca jakość od dołu
    - lewy operator
    - prawy operator
    - b to wartość ograniczająca jakość od góry
- filtr po typie *TypesFilter*, zawierający mapę z listą typów i informacją czy jest dany typ jest zaznaczony
- filtr po współrzędnych *CoordinatesFilter*, zawierający
    - listę warunków będących piątką spełniającą równanie: $Ax + By + Cz\ [operator]\ D$, gdzie:
        - A to *xValue*
        - B to *yValue*
        - C to *zValue*
        - operator
        - D to *constant*
    - spójnik łączący warunki *conjunction* o możliwych wartościach:
        - *AND*
        - *OR*
- filtr po grupach *GroupsFilter*

Moduł komunikacji po otrzymaniu tej wiadomości parsuje ją i wywołuje odpowiednie metody modułu filtracji w celu zastosowania filtrów. Następnie przeliczane są statystyki w module drzewa struktur i przesyłana jest do Smeshalist Manager'a wiadomość *CoreToManagerMessage* ze zaktualizowanymi statystykami.



Rysunek 4 Smeshalist Manager - Opcje

Smehalist Manager umożliwia korzystanie z dodatkowych opcji dostępnych w zakładce *Options*. Po zatwierdzeniu wybranych opcji wysyłana jest wiadomość *ManagerToCoreMessage* z typem ustawionym na *OPTIONS* oraz zawartością zakładki w postaci wiadomości *OptionsInfo* zawierającej:

- flagę czy wyświetlane struktury mają być transparentne
- flagę czy dynamiczny rendering ma być włączony; w przypadku odznaczenia tej opcji w celu wyświetlenia przesłanych struktur użytkownik musi wywołać z poziomu API klienckiego metodę *render()*
- flagę czy mają być wyświetlane etykiety struktur
- wartość parametru "czułość myszki"
- typ kolorowania struktur, dostępne są opcje kolorowania po jakości i po ID grupy
- tryb wizualizacji (3D lub 2D)

W zakładce *Options* użytkownik ma także możliwość wykonania zrzutu drzewa struktur oraz wyboru wyświetlanego zrzutu. Po kliknięciu przycisku *Snapshot* wysyłana jest wiadomość

ManagerToCoreMessage z typem ustawionym na *SNAPSHOT*. Moduł komunikacji wywołuje odpowiednie metody modułu drzewa struktur w celu wykonania zrzutu. Kliknięcie przycisków *Prev* lub *Next* powoduje wysłanie wiadomości *ManagerToCoreMessage* z typem ustawionym na odpowiednio *PREV_TREE* lub *NEXT_TREE*. Moduł komunikacji koordynuje wyświetlenie odpowiedniego zrzutu drzewa struktur, a następnie wysyła w odpowiedzi do Smeshalist Manager'a wiadomość ze statystykami.

Jeżeli z poziomu API klienckiego zostanie wywołana metoda *breakpoint()*, nastąpi zawieszenie wykonania algorytmu do momentu odpowiedniej reakcji z poziomu okna Smeshalist Manager. W zakładce *Options* użytkownik ma możliwość:

- wznowić wykonanie algorytmu kliknięciem przycisku *Continue* (wiadomość *ManagerToCoreMessage* typu *CONTINUE*)
- zakończyć wykonanie algorytmu kliknięciem przycisku *Abort* (wiadomość *ManagerToCoreMessage* typu *ABORT*)

Narzędzie udostępnia możliwość importu oraz eksportu struktur z plików w formacie OBJ. W tym celu należy wybrać odpowiednią opcję z zakładki *Import/Export* w oknie Smeshalist Manager. Powoduje to wysłanie wiadomości *ManagerToCoreMessage* typu odpowiednio *IMPORT* lub *EXPORT* zawierającej ścieżkę do pliku OBJ do modułu komunikacji. Moduł komunikacji koordynuje wykonanie importowania lub eksportowania struktur geometrycznych.

Organizacja pracy

Metodyka

Metodyka zastosowana przez nas w projekcie to model przyrostowy. Prace zostały podzielone na trzy przyrosty, w których każdy kolejny dostarczał nowej funkcjonalności narzędzia.

Przyrost I

Pierwszy etap obejmował zebranie precyzyjnych wymagań odnośnie funkcjonalności narzędzia. Zaprojektowana została architektura systemu oraz dokonany został podział na moduły. Następnie zaimplementowany został prototypu narzędzia działającego pod systemem Linux, który dostarczał podstawowej funkcjonalności tj. wizualizacji podstawowych typów struktur, oraz ich filtracji. Zaimplementowany został interfejs programistyczny w języku Java w zakresie zdefiniowana wewnętrznego formatu struktur oraz komunikacji z modułem komunikacji CORE oraz okienko Smeshalist Manager w zakresie wyświetlania statystyk

Przyrost II

Celem drugiego przyrostu było przeniesienie podstawowej funkcjonalności na platformę Windows oraz dopracowanie aspektów wizualnych aplikacji oraz dalszy rozwój funkcjonalności Java API i okienka Smeshalist Manager. Podczas tego etapu ukończona została implementacja interfejsu programistycznego w języku Java. Zostały wykonane prace nad poprawnym działaniem filtracji oraz zaimplementowane została funkcjonalność okienka Smeshalist Manager. Z powodu problemów podczas próby automatycznego budowania narzędzia na platformie Windows funkcjonalność ta nie została przeniesiona na ten system operacyjny.

Przyrost III

Podczas tego etatu funkcjonalność interfejsu użytkownika została przeniesiona na język C++ oraz Python. Dodana została możliwość importu oraz eksportu drzewa struktur do pliku .obj. Zapewniona została możliwość konfiguracji podstawowych parametrów narzędzia np. kolorów dla poszczególnych grup. Funkcjonalność narzędzia została przeniesiona na system Windows. Ponadto zaimplementowane zostały dodatkowe wymagania określone przez klienta po zapoznaniu się z pierwszym prototypem takie jak wyświetlanie etykiet struktur, możliwość przejścia w tryb 2D czy też ustalenia przezroczystości struktur.

Wyniki projektu

Rysunek 5 Przykładowy wynik działania narzędzia Smeshalist

Wyniki, ocena użyteczności

Wynikiem projektu jest narzędzie spełniające wszystkie założone na etapie projektowania wymagania funkcjonalne oraz niefunkcjonalne. Produkt został przetestowany pod względem wydajnościowym jak również kompatybilności z systemami Linux i Windows. Spełnia również wymaganie co do użyteczności i prostoty konfiguracji.

Ponadto została dostarczona również kompletna dokumentacja procesowa, użytkownika oraz techniczna. Dodatkowo każda implementacja programistycznego interfejsu użytkownika jest odpowiednio udokumentowana w komentarzach kodu źródłowego.

Propozycje rozwoju systemu

Narzędzie można ulepszyć o następujące aspekty:

- rozszerzenie zbioru rysowanych struktur o kolejne elementy
- zapamiętywanie sesji użytkownika w narzędziu
- poprawienie użyteczności interfejsu graficznego okna Smeshalist Manager
- rozszerzenie możliwości konfiguracji narzędzia np. o wybór skrótów klawiszowych wykorzystywanych w nawigacji po oknie wizualizacji
- umożliwienie dokonywania pomiaru kątów między zwizualizowanymi elementami
- dodanie obsługi kolejnych formatów plików przy importowaniu i eksportowaniu struktur
- dodanie możliwości manipulacji właściwościami wyświetlonych struktur

Spis ilustracji

Źródła

1. Hawkings Kevin, Astle Dave. *OpenGL. Programowanie gier*, Helion, 2003r.
2. Matulewski Jacek, Dziubak Tomasz, Sylwestrzak Marcin, Płoszajczak Radosław. *Grafika - Fizyka - Metody numeryczne. Symulacje fizyczne z wizualizacją 3D*, Wydawnictwo Naukowe PWN, 2010r.
3. Google Developers. *Developers Guide | Protocol Buffers*, dostępna w Internecie: http://developers.google.com/protocol-buffers/docs/overview [dostęp: 5.01.2017]