AKADEMIA GÓRNICZO-HUTNICZA

IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI



DOKUMENTACJA UŻYTKOWNIKA

**NARZĘDZIE DO WIZUALIZACJI SIATEK TRÓJWYMIAROWYCH**

WOJCIECH DYMEK

KATARZYNA GŁĄB

KATARZYNA KONIECZNA

EWA MARCZEWSKA

OPIEKUN:

Dr inż. Tomasz Jurczyk

Spis treści

[Podstawowe informacje 4](#_Toc467527383)

[Wymagania aplikacji 4](#_Toc467527384)

[Konfiguracja środowiska 4](#_Toc467527385)

[Kompilacja 4](#_Toc467527386)

[Uruchomienie aplikacji 4](#_Toc467527387)

[Plik konfiguracyjny *user.config.xml* 5](#_Toc467527388)

[Struktura pliku 5](#_Toc467527389)

[Interfejs programistyczny użytkownika API 6](#_Toc467527390)

[Java 6](#_Toc467527391)

[Pakiet geometry 6](#_Toc467527392)

[Pakiet tool 8](#_Toc467527393)

[C++ 10](#_Toc467527394)

[Geometry.h 10](#_Toc467527395)

[Smeshalist.h 12](#_Toc467527396)

[Python 13](#_Toc467527397)

[Moduł geometry.py 13](#_Toc467527398)

[Moduł Smeshalist.py 14](#_Toc467527399)

[Wykorzystanie API w różnych językach programowania 16](#_Toc467527400)

[Java 16](#_Toc467527401)

[Python 16](#_Toc467527402)

[C++ 16](#_Toc467527403)

[Smeshalist Manager 17](#_Toc467527404)

[Statistics 17](#_Toc467527405)

[Options 18](#_Toc467527406)

[Filters 19](#_Toc467527407)

[Types 19](#_Toc467527408)

[Groups 20](#_Toc467527409)

[Quality 21](#_Toc467527410)

[Coordinates 22](#_Toc467527411)

[Spis ilustracji 24](#_Toc467527412)

# Podstawowe informacje

Smeshalist to aplikacja służąca do wizualizacji siatek trójwymiarowych oraz ich elementów. Jej celem jest ułatwienie m. in. tworzenia algorytmów z zakresu geometrii obliczeniowej poprzez wyświetlanie kolejnych kroków rozwiązania, zaimplementowanego w języku Java, C++ lub Python. Program może być uruchamiany na systemie operacyjnym Linuks lub Windows.

## Wymagania aplikacji

Do poprawnego uruchomienia aplikacji zainstalować należy:

1. Java JRE 1.8
2. GLUT – OpenGL Utility Toolkit

# Konfiguracja środowiska

## Kompilacja

Znajdując się w folderze głównym aplikacji uruchomić polecenie *make all.* Zostanie wygenerowany plik wykonywalny *SmeshalistCore*, który odpowiada za komunikację między poszczególnymi częściami systemu oraz wizualizację.

## Uruchomienie aplikacji

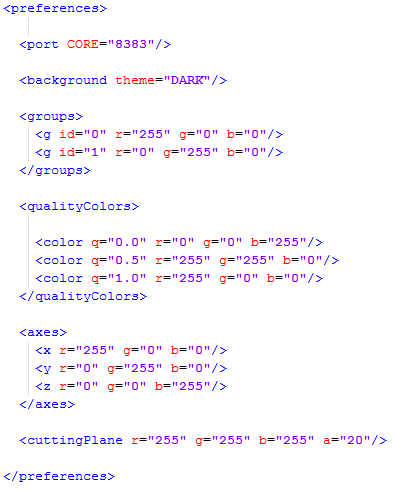
Kroki potrzebne do poprawnego uruchomienia aplikacji:

1. W razie konieczności zmiana pliku konfiguracyjnego *user.config.xml*.
2. Uruchomienie *SmeshalistCore*.

# Plik konfiguracyjny *user.config.xml*

Plik konfiguracyjny powinien znajdować się w tej samej lokalizacji co plik wykonywalny *SmeshalistCore*.

## Struktura pliku



Rysunek 1 Przykładowa struktura pliku konfiguracyjnego

Konfiguracja znajduje się w węźle *<preferences>*. Dostępne opcje to:

1. *<port>* - umożliwia zmianę portu, na którym nasłuchuje serwer aplikacji. Nowy port należy podać jako atrybut *CORE*. Musi on być zgodny z wartością argumentu przekazywanego do metody *getInstance*. Domyślnie wykorzystywany jest port 8383.
2. *<background > -* konfiguracja koloru tła obszaru roboczego. W przypadku nie podania węzła, bądź nie uzupełnienia atrybutu *theme* domyślnie kolor jest biały. W celu ustawienia ciemnego tła należy dodać powyższy węzeł wraz z atrybutem *theme=”DARK”.*
3. *<groups>* - konfiguracja kolorów zdefiniowanych dla poszczególnych grup. Aby ustawić kolor wybranej grupie należy dodać potomka w postaci węzła *<g id=”X”>* gdzie *X* oznacza numer grupy, wraz z atrybutami *r, g, b.*
4. *<qualityColors>* - umożliwia podanie palety kolorów wraz z odpowiadającymi im wartościami współczynnika jakości. Definicja polega na dodaniu potomka *<color>* wraz z atrybutami *q* – wartość współczynnika jakości oraz trójki *r, g, b*. W przypadku wartości spoza przedziału [0; 1] można zdefiniować dodatkowy kolor poprzez dodanie potomka <*negQualityColor>* wraz z trójką *r, g, b*. W przypadku braku tej wartości zostaną użyta wartość domyślna – (159, 0, 255). Wartości pośrednie w przedziałach wyliczana są jako średnie ważone kolorów zdefiniowanych na końcach odpowiednich przedziałów. W przypadku nieokreślenia kolorów dla jakości równej 0 bądź 1, zostaną użyte wartości domyślne, odpowiednio (0,0,0) oraz (255,255,255).
5. *<axes>* - konfiguracja kolorów poszczególnych osi. Aby ustawić kolor wybranej osi należy dodać potomka odpowiednio *<x>|<y>|<z>*, wraz z podaniem trójki *r, g, b*. W przypadku niezdefiniowania kolorów zostaną użyte wartości domyślne, odpowiednio: dla osi x: (255, 0, 0), y: (0, 255, 0), z: (0, 0, 255).
6. *<cuttingPlane>* - umożliwia zmianę koloru oraz współczynnika przezroczystości płaszczyzn, będących wizualizacji filtrów po współrzędnych. Wartości (R, G, B, A) podawane są jako kolejne atrybuty *r, g, b, a* węzła. W przypadku niezdefiniowania koloru zostanie użyty kolor domyślny – (127, 25, 25, 25).

# Interfejs programistyczny użytkownika API

## Java

### Pakiet geometry

Zawiera klasy stanowiące wewnętrzny dla narzędzia Smeshalist model struktur geometrycznych.

#### Klasa Point3D

Klasa przechowująca współrzędne punktu, będąca składową wszystkich klas struktur geometrycznych.

* konstruktor *Point3D(double x, double y, double z)*
* metody dostępowe do pól współrzędnych: *double getX()*, *double getY()*, *double getZ()*, *void setX(double x)*, *void setY(double y)*, *void setZ(double z)*

#### Klasa Vertex

* Point3D point
* double quality
* String label
* int groupId
* konstruktor Vertex(Point3D point, double number) – tworzy obiekt klasy Vertex
* metody dostępowe do pól:
  + void setPoint(Point3D point)
  + Point3D getPoint()
  + void setQuality(double quality)
  + double getQuality()
  + void setLabel(String label)
  + String getLabel()
  + void setGroupId(int groupId)
  + int getGroupId()

#### Klasa Edge

* Point3D v1
* Point3D v2
* double quality
* String label
* int groupId
* konstruktor Edge(Point3D v1, Point3D v2) – tworzy obiekt klasy Edge
* metody dostępowe do pól:
  + void setV1(Point3D point)
  + Point3D getV1()
  + void setV2(Point3D point)
  + Point3D getV2()
  + void setQuality(double quality)
  + double getQuality()
  + void setLabel(String label)
  + String getLabel()
  + void setGroupId(int groupId)
  + int getGroupId()

#### Klasa TriangleFace

* Point3D v1
* Point3D v2
* Point3D v3
* double quality
* String label
* int groupId
* konstruktor TriangleFace(Point3D v1, Point3D v2, Point3D v3) – tworzy obiekt klasy TriangleFace
* metody dostępowe do pól:
  + void setV1(Point3D point)
  + Point3D getV1()
  + void setV2(Point3D point)
  + Point3D getV2()
  + void setV3(Point3D point)
  + Point3D getV3()
  + void setQuality(double quality)
  + double getQuality()
  + void setLabel(String label)
  + String getLabel()
  + void setGroupId(int groupId)
  + int getGroupId()

#### Klasa Block

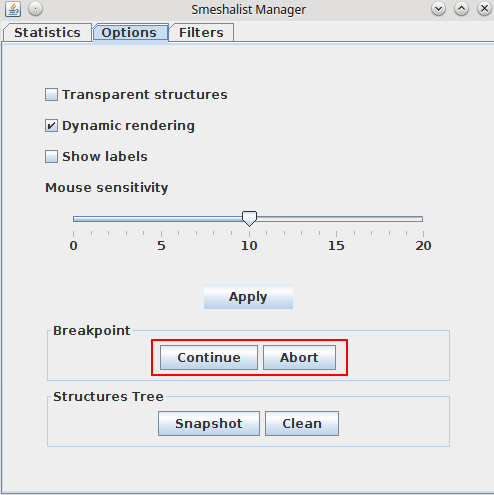
* Point3D v1
* Point3D v2
* Point3D v3
* Point3D v4
* double quality
* String label
* int groupId
* konstruktor Block(Point3D v1, Point3D v2, Point3D v3, Point3D v4) – tworzy obiekt klasy Block
* metody dostępowe do pól:
  + void setV1(Point3D point)
  + Point3D getV1()
  + void setV2(Point3D point)
  + Point3D getV2()
  + void setV3(Point3D point)
  + Point3D getV3()
  + void setV4(Point3D point)
  + Point3D getV4()
  + void setQuality(double quality)
  + double getQuality()
  + void setLabel(String label)
  + String getLabel()
  + void setGroupId(int groupId)
  + int getGroupId()

### Pakiet tool

#### Klasa Smeshalist

Jest to główna klasa narzędzia, dostarczająca metod umożliwiających dodawanie geometrii do wizualizacji.

* *Smeshalist getInstance()* – statyczna metoda, tworzy instancję lub zwraca istniejącą instancję narzędzia działającego na domyślnym porcie numer 8383
* *Smeshalist getInstance(int portNumber)* – statyczna metoda, tworzy instancję lub zwraca istniejącą instancję narzędzia działającego na porcie przekazanym jako argument wywołania metody
* *void destroySmeshalist()* – statyczna metoda, wołana po zakończeniu pracy z narzędziem Smeshalist, zamyka socket służący do komunikacji wewnętrznej
* *void addGeometry(Vertex vertex)*– metoda dodaje daną strukturę do bufora danych, które będą przesłane do wizualizacji
* *void addGeometry(Edge edge)*
* *void addGeometry(TriangleFace triangleFace)*
* *void addGeometry(Block block)*
* *void flushBuffer()* – przesyła zgromadzone w buforze struktury geometryczne do modułu wizualizacji
* *void breakpoint()* – metoda wstrzymująca działanie algorytmu do momentu wybrania jednej z dwóch opcji (*Continue, Abort*) dostępnych w oknie Smeshalist Manager



* *void render()* – metoda wymuszająca wyświetlenie przesłanych struktur w przypadku odznaczonej w oknie Smeshalist Manager opcji *Dynamic rendering*
* *void clean()* – metoda czyszcząca zawartość aktywnego drzewa struktur

#### Przykład użycia

**package** test;  
  
**import** geometry.Point3D;  
**import** tool.Smeshalist;  
  
**public class** Example01 {  
 **public static void** main(String[] args) {  
 Smeshalist tool = Smeshalist.getInstance();  
   
 **for** (**double** x=0; x<10; x+=0.1) {  
 **for** (**double** z=0; z<10; z+=0.1) {  
 **double** y = x\*z/10.0;  
 Point3D point = **new** Point3D(x, y, z);  
 point.setQuality(y/10.0);  
 tool.addGeometry(point);  
 }  
 }  
   
 tool.flushBuffer();  
   
 Smeshalist.destroySmeshialist();  
 }  
}

## C++

### Geometry.h

#### Klasa Geometry

Klasa bazowa wszystkich dostępnych geometrii. Zawiera elementy służące do opisania każdej z nich.

* *void SetQuality(double q)* – ustawia pole *quality*
* *double GetQuality()* – zwraca wartość pola *quality*
* *void SetGroupId(int id)* – ustawia pole *group\_id*
* *int GetGroupId()* – zwraca wartość pola *group\_id*
* *void SetLabel(string l)* – ustawia pole *label*
* *string GetLabel()* – zwraca wartość pola *label*

#### Klasa Point3D

Klasa przechowująca współrzędne punktu, będąca składową wszystkich klas struktur geometrycznych.

* konstruktor Point3D()
* konstruktor *Point3D(double x, double y, double z)*
* metody dostępowe do pól współrzędnych: *double GetX()*, *double GetY()*, *double GetZ()*, *void SetX(double x)*, *void SetY(double y)*, *void SetZ(double z)*

#### Klasa Vertex

* Point3D point
* konstruktor Vertex()
* konstruktor Vertex(double x, double y, double z)
* konstruktor Vertex(Point3D point)
* metody dostępowe do pól:
  + void SetPoint(Point3D point)
  + Point3D GetPoint()

#### Klasa Edge

* Point3D v1
* Point3D v2
* konstruktor Edge()
* konstruktor Edge(Point3D v1, Point3D v2) – tworzy obiekt klasy Edge
* metody dostępowe do pól:
  + void SetV1(Point3D point)
  + Point3D GetV1()
  + void SetV2(Point3D point)
  + Point3D GetV2()

#### Klasa Face

* Point3D v1
* Point3D v2
* Point3D v3
* konstruktor Face()
* konstruktor Face(Point3D v1, Point3D v2, Point3D v3) – tworzy obiekt klasy Face
* metody dostępowe do pól:
  + void SetV1(Point3D point)
  + Point3D GetV1()
  + void SetV2(Point3D point)
  + Point3D GetV2()
  + void SetV3(Point3D point)
  + Point3D GetV3()

#### Klasa Block

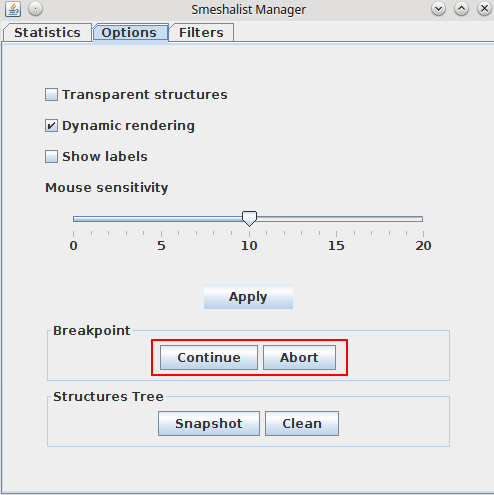
* Point3D v1
* Point3D v2
* Point3D v3
* Point3D v4
* konstruktor Block()
* konstruktor Block(Point3D v1, Point3D v2, Point3D v3, Point3D v4) – tworzy obiekt klasy Block
* metody dostępowe do pól:
  + void SetV1(Point3D point)
  + Point3D GetV1()
  + void SetV2(Point3D point)
  + Point3D GetV2()
  + void SetV3(Point3D point)
  + Point3D GetV3()
  + void SetV4(Point3D point)
  + Point3D GetV4()

### Smeshalist.h

#### Klasa Smeshalist

Jest to główna klasa narzędzia, dostarczająca metod umożliwiających dodawanie geometrii do wizualizacji.

* *Smeshalist& GetInstance()* – statyczna metoda, tworzy instancję lub zwraca istniejącą instancję narzędzia działającego na domyślnym porcie numer 8383
* *void AddGeometry(Vertex &vertex)*– metoda dodaje daną strukturę do bufora danych, które będą przesłane do wizualizacji
* *void AddGeometry(Edge &edge)*
* *void AddGeometry(Face &face)*
* *void AddGeometry(Block &block)*
* *void FlushBuffer()* – przesyła zgromadzone w buforze struktury geometryczne do modułu wizualizacji
* *void Breakpoint()* – metoda wstrzymująca działanie algorytmu do momentu wybrania jednej z dwóch opcji (*Continue, Abort*) dostępnych w oknie Smeshalist Manager



* *void Render()* – metoda wymuszająca wyświetlenie przesłanych struktur w przypadku odznaczonej w oknie Smeshalist Manager opcji *Dynamic rendering*
* *void Clean()* – metoda czyszcząca zawartość aktywnego drzewa struktur

## Python

### Moduł geometry.py

#### Point3D

Klasa przechowująca współrzędne punktu, będąca składową wszystkich klas struktur geometrycznych.

* quality – pole typu double, wskaźnik jakości struktury
* label – pole typu string, jest to etykieta struktury
* groupId – pole typu int, identyfikator grupy, do której należy struktura
* *\_\_init\_\_(self, x, y, z)* – konstruktor klasy, gdzie x, y, z to współrzędne punktu

#### Edge

* quality – pole typu double, wskaźnik jakości struktury
* label – pole typu string, jest to etykieta struktury
* groupId – pole typu int, identyfikator grupy, do której należy struktura
* \_\_init\_\_(self, v1, v2) – konstruktor klasy, gdzie v1, v2 są typu *Point3D*

#### TriangleFace

* quality – pole typu double, wskaźnik jakości struktury
* label – pole typu string, jest to etykieta struktury
* groupId – pole typu int, identyfikator grupy, do której należy struktura
* \_\_init\_\_(self, v1, v2, v3) – konstruktor klasy, gdzie v1, v2, v3 są typu *Point3D*

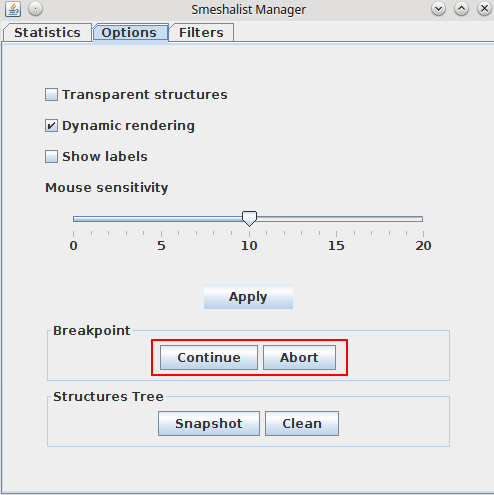
#### Block

* quality – pole typu double, wskaźnik jakości struktury
* label – pole typu string, jest to etykieta struktury
* groupId – pole typu int, identyfikator grupy, do której należy struktura
* \_\_init\_\_(self, v1, v2, v3, v4) – konstruktor klasy, gdzie v1, v2, v3, v4 są typu *Point3D*

### Moduł Smeshalist.py

Jest to główna klasa narzędzia, dostarczająca metod umożliwiających dodawanie geometrii do wizualizacji.

* *getInstance(portNumber)* – tworzy instancję lub zwraca istniejącą instancję narzędzia działającego na porcie przekazanym jako argument wywołania metody
* *addPoint3D(point3D)* – metoda dodaje daną strukturę do bufora danych, które będą przesłane do wizualizacji
* *addEdge(edge)*
* *addTriangleFace(triangleFace)*
* *addBlock(block)*
* *flushBuffer()* – przesyła zgromadzone w buforze struktury geometryczne do modułu wizualizacji
* *breakpoint()* – metoda wstrzymująca działanie algorytmu do momentu wybrania jednej z dwóch opcji (*Continue, Abort*) dostępnych w oknie Smeshalist Manager



* *render()* – metoda wymuszająca wyświetlenie przesłanych struktur w przypadku odznaczonej w oknie Smeshalist Manager opcji *Dynamic rendering*
* *clean()* – metoda czyszcząca zawartość aktywnego drzewa struktur

## Wykorzystanie API w różnych językach programowania

### Java

Aby wykorzystać narzędzie w języku Java należy dołączyć do projektu archiwuj jar znajdujące się w katalogu *JavaAPI*.

### Python

Aby wykorzystać narzędzie w języku Python należy zaimportować do skryptu moduły *Smeshalist* oraz *geometry* znajdujące się w katalogu *PythonAPI.*

### C++

Aby wykorzystać narzędzie w języku C++ należy zlinkować projekt z biblioteką *Smeshalist.a* znajdującym się w katalogu *CppAPI.*

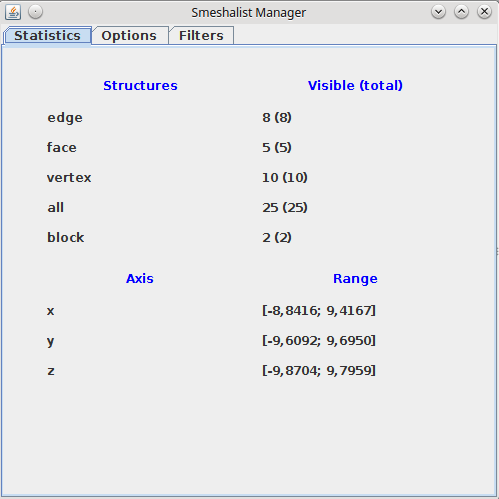
# Smeshalist Manager

Smeshalist Manager to okienko umożliwiające kontrolę nad strukturami wyświetlanymi w obszarze roboczym oraz wyświetlanie informacji o tych strukturach.

## Statistics

W zakładce statystyk wyświetlane są:

* ilość przesłanych i zwizualizowanych struktur danego typu
* współrzędne prostopadłościanu ograniczającego

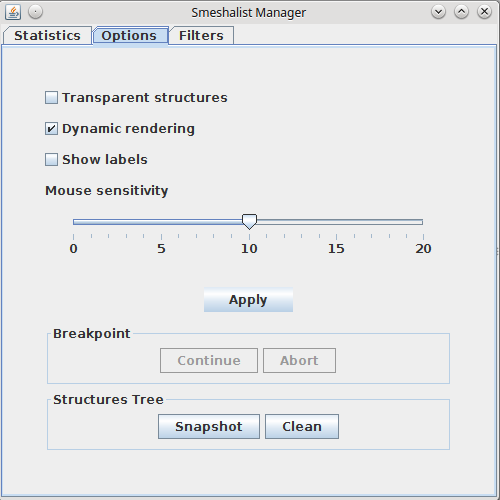


Rysunek 3 Smeshalist Manager - Statystyki

## Options

W zakładce opcji użytkownik ma możliwość:

* włączyć/wyłączyć dynamiczne renderowanie (*Dynamic rendering*)
* włączyć/wyłączyć wyświetlanie przezroczystych struktur (*Transparent structures*)
* włączyć/wyłączyć wyświetlanie etykiet struktur (*Show labels*)
* ustawić czułość myszki (*Mouse sensitivity*)
* zareagować na wywołanie metody *Breakpoint* (przyciski *Continue* i *Abort*)
* zrobić nowy zrzut drzewa struktur (przycisk *Snapshot*)
* wyczyścić aktywne drzewo struktur (przycisk *Clean*)



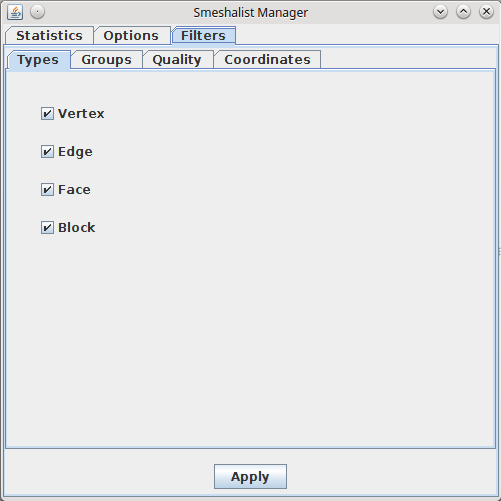
Rysunek 4 Smeshalist Manager - Opcje

## Filters

Zakładka filtrów oferuje użytkownikowi możliwość filtrowania wyświetlonych struktur z wykorzystaniem czterech rodzajów filtrów. Zmiany w filtrach należy zatwierdzić przyciskiem *Apply*, co spowoduje uruchomienie filtrowania.

### Types

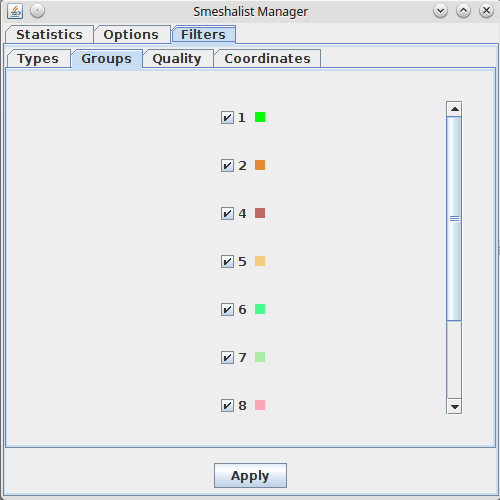
Umożliwia filtrowanie po typie struktury (Vertex, Edge, Face, Block).



Rysunek 5 Smeshalist Manager - Filtr Typy

### Groups

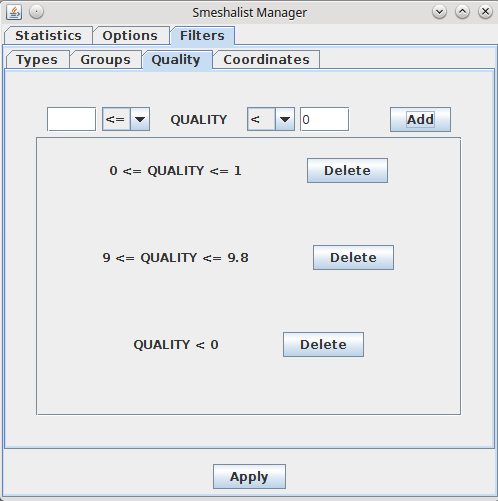
Umożliwia filtrowanie po ID grupy.



Rysunek 6 Smeshalist Manager - Filtr Grupy

### Quality

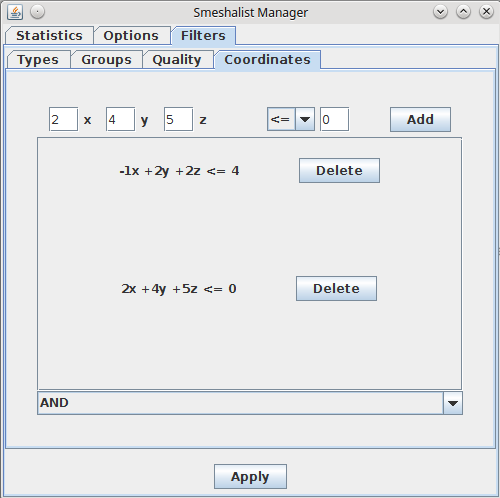
Umożliwia filtrowanie po właściwości Quality każdej struktury. Użytkownik dodaje warunek przyciskiem *Add*. Wszystkie warunki są łączone spójnikiem LUB. Aby usunąć odpowiedni warunek należy użyć odpowiadającemu mu przycisku *Delete*.



Rysunek 7 Smeshalist Manager - Filtr Jakość

### Coordinates

Umożliwia filtrowanie po współrzędnych struktur. Użytkownik dodaje warunek przyciskiem *Add*. Aby usunąć odpowiedni warunek należy użyć odpowiadającemu mu przycisku *Delete.* Warunki mogą być łączone spójnikiem I albo LUB. Filtr działa tylko na struktury, które w całości spełniają warunek (a nie np. tylko jeden z wierzchołków ściany).



Rysunek 8 Smeshalist Manager - Filtr Współrzędne

# Spis ilustracji

[Rysunek 1 Przykładowa struktura pliku konfiguracyjnego 5](#_Toc470208700)

[Rysunek 3 Smeshalist Manager - Statystyki 19](#_Toc470208701)

[Rysunek 4 Smeshalist Manager - Opcje 20](#_Toc470208702)

[Rysunek 5 Smeshalist Manager - Filtr Typy 21](#_Toc470208703)

[Rysunek 6 Smeshalist Manager - Filtr Grupy 22](#_Toc470208704)

[Rysunek 7 Smeshalist Manager - Filtr Jakość 23](#_Toc470208705)

[Rysunek 8 Smeshalist Manager - Filtr Współrzędne 24](#_Toc470208706)