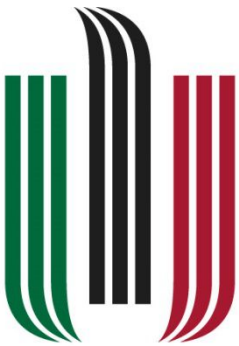

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA INFORMATYKI

DOKUMENTACJA PROCESOWA

Implementacja narzędzia modelowania brył elastycznych

Tool for modelling flexible objects

Autor: *Piotr Baran, Michał Kasprzyk*
Kierunek studiów: informatyka
Opiekun pracy: *doktor inżynier Paweł Topa*

Kraków, 2015

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.

.....

Spis treści

1. Wizja

Celem projektu jest stworzenie aplikacji służącej do symulowania zjawisk występujących w pewnych układach fizycznych zwanych dalej bryłami elastycznymi. Stanowiące temat projektu bryły elastyczne są zbiorami punktów zawieszonych w trójwymiarowej przestrzeni. Punkty te połączone są z sąsiadującymi punktami za pomocą połączeń, których sztywność wpływa na odkształcenia całej bryły. W układach tych występuje również centralny punkt, połączony ze wszystkimi innymi. Przesunięcie go z pierwotnego położenia powoduje zmianę kształtu bryły. Tworzona aplikacja ma za zadanie symulowanie działania takich układów w różnych stanach oraz wizualizowania wyników swoich obliczeń. Uzyskaną wizualizację można następnie wyeksportować do popularnego formatu definicji geometrii *.obj*. Stworzona aplikacja może znaleźć zastosowanie w badaniach naukowych - zachowanie symulowanej bryły elastycznej odpowiada sposobowi budowania skorupki przez jednokomórkowe organizmy z gromady Foraminifera. Aplikacja mogłaby być też podstawą do jakiegoś innego, bardziej złożonego projektu, na przykład profesjonalnego pakietu symulacji fizycznych, czy jako silnik do gry logicznej. Projekt ma zatem potencjał, by istnieć autonomicznie, nie tylko jako przedmiot pracy inżynierskiej.

2. Studium wykonywalności

2.1. Wymagania funkcjonalne

1. **Stworzenie nowego projektu - wygenerowania odpowiedniego układu dla przypadku dwu i trzy wymiarowego**

Użytkownik powinien mieć możliwość stworzenia nowego projektu, co wiąże się z wygenerowaniem odpowiedniego układu. Podstawowych układów powinny być dwa rodzaje: dwu i trzy wymiarowy. Ponadto powinna istnieć możliwość zdefiniowania z jak dużej liczby punktów ma składać się układ.

2. **Wczytanie układu z pliku**

Układ, który został zapisany w jednej z poprzednich sesji korzystania z aplikacji, może zostać wczytany z pliku. Nie powinno wymagać to żadnych dodatkowych ustaleń ze strony użytkownika.

3. **Wizualizacja układu**

Układ, składający się z punktów i połączeń, powinien zostać odpowiednio zwizualizowany. Powinna istnieć możliwość zaznaczania odpowiednich, zwizualizowanych, elementów w celu podglądu szczegółów elementu lub edycji odpowiednich parametrów.

4. **Edycja parametrów elementów i zmiana układu**

Po zaznaczeniu odpowiedniego elementu powinna istnieć możliwość zmiany danych parametrów wpływających na kształt całego układu. Wiąże się to z opracowaniem odpowiedniego algorytmu, opartego o zasady fizyki, obliczającego nowy stan elementów układu. Zmiany powinny zostać zwizualizowane.

5. **Zapis układu**

Użytkownik powinien mieć możliwość zapisu powstałego układu do pliku w celu późniejszej kontynuacji pracy.

Dodatkowe wymagania funkcjonalne

1. Eksport do pliku *.obj*

Użytkownik powinien mieć możliwość weksportowania zwizualizowanego układu do pliku w popularnym formacie *.obj*.

2. Większa liczba układów do wygenerowania podczas tworzenia nowego projektu

Domyślnie powinna istnieć możliwość wygenerowania dwóch układów - dwu i trzy wymiarowego. Przydatna jednak mogłaby okazać się możliwość wygenerowania różnych wariantów trójwymiarowego układu. Dla przykładu za podstawowy układ można by uznać półsferę, a za dodatkowy kwadratową płachtę.

2.2. Wymagania niefunkcjonalne

1. Przejrzysta wizualizacja

Zasadniczą część projektu stanowi wizualizacja obliczonych zmian siatki. Dlatego też bardzo ważne jest, aby owa wizualizacja była możliwie najbardziej czytelna i przejrzysta, możliwość przeglądania bryły była łatwa i intuicyjna, podobnie, jak sposób manipulacji układem.

2. Intuicyjny interfejs użytkownika

Graficzny interfejs użytkownika ma na celu wspomaganie modyfikowania parametrów bryły, dlatego powinien pozwalać na wygodne wyszukiwanie pożądanych opcji.

3. Wydajny algorytm

Główną funkcjonalność aplikacji stanowi obliczanie sił występujących w bryle oraz jej odkształceń. Algorytm, który się tym zajmuje powinien zatem być w stanie znajdować poprawne wyniki w odpowiednio krótkim czasie.

2.3. Analiza ryzyka

Ryzyko występuje podczas prac nad każdym projektem. Istotne zatem jest przewidzenie jego źródeł oraz takie zaplanowanie działań, aby zminimalizować prawdopobieństwo wystąpienia czynników

utrudniających ich ukończenie. W przypadku naszego projektu ryzyko dotyczy kwestii opisanych poniżej.

1. Wielkość zespołu

Projekt wykonywany jest w zespole dwuosobowym, co z jednej strony pozwala na utrzymanie lepszej, niż w przypadku szerszych zespołów komunikacji między członkami, z drugiej jednak powoduje, że stosunkowo niewielki zasób ludzki może okazać się niewystarczajacy w przypadku wystąpienia czasochłonnych zadań, które będą powodować, że przez długi okres czasu przyrosty projektu będą stosunkowo niewielki.

2. Trudność zweryfikowania poprawności obliczeń

Badawczy charakter projektu staje się problematyczny podczas testowania wyników projektu. Trudno jest zweryfikować wyniki obliczeń. W prostszych przypadkach możliwe jest ręczne przeprowadzenie obliczeń (do tego raczej tylko w przypadku dwuwymiarowym), jednak szybko układy te stają się zbyt skomplikowane, aby możliwe było takie sprawdzenie poprawności.

3. Planowanie prac

Początkiem wykonywania projektu było sporządzenie wizji zawierającej opis najważniejszych funkcjonalności tworzonego przez nas systemu, jak również początkowych ustaleń odnoszących się do technicznych aspektów implementacji. Aby krótko streścić informacje, dokładniej wyszczególnione w innym dokumencie – oprogramowanie pozwala na obliczanie odkształceń w siatce złożonej z punktów, które następnie są wizualizowane w postaci modelu dwu, lub trójwymiarowego w zależności od wybranego trybu pracy. Program umożliwia też modyfikację własności fizycznych elementów bryły, co umożliwia symulację powstawania różnych kształtów w zależności od panujących warunków.

Wybraną przez nas technologią do stworzenia aplikacji była *Java*. Wynika to z faktu, że w pracy z jego wykorzystaniem mamy największe doświadczenie. Jako, że nasz projekt jest silnie związany z prezentacją danych użytkownikowi oraz interakcją z nim, graficzny interfejs użytkownika musiał zostać stworzony w taki sposób, aby był czytelny, ale jednocześnie umożliwiający rozbudowane możliwości osobie korzystającej z produktu. W związku z tym, ustaliliśmy, że zostanie on wykonany przy użyciu technologii *JavaFX*, ponieważ udostępnia ona rozbudowane narzędzia pozwalające w dość prosty sposób osiągnąć efekty, które w innych technologiach wymagałyby dużo większych nakładów pracy.
W założeniu moduł wizualizacji miał zostać stworzony z wykorzystaniem *WebGL*. Podobnie jak *JavaFX*, jest to technologia o relatywnie wysokim poziomie, tworzenie elementów graficznych za jej pomocą jest dużo łatwiejsze, niż w *OpenGL*.

4. Przyjęta metodyka pracy

Podczas pracy nad projektem wykorzystywany był przyrostowy model tworzenia oprogramowania. Kolejne etapy tworzenia aplikacji zależały raczej od aktualnie opracowywanej funkcjonalności aniżeli od sztywno wytyczonych ram czasowych. Z tego powodu opisywane poniżej etapy prac biorą swoje tytuły od funkcjonalności, które były podczas nich implementowane.

Ponieważ projekt tworzony był przez dwie osoby, potrzebny był podział obowiązków, który umożliwiłby efektywne tworzenie aplikacji. Każdy z nas rozpoczynał ten projekt z jakąś wizją tego, czym chciałby się zajmować w procesie implementacji programu. W związku z tym, po wydzieleniu pewnych zadań, które odpowiadały stworzeniu pewnych obszarów aplikacji, obydwaj wybraliśmy te z nich, które najbardziej odpowiadały naszym zainteresowaniom. W ogólności można wyróżnić główne obszary naszej aktywności: Piotr Baran zajmował się w większości *backendem*, czyli modelem aplikacji oraz algorytmem obliczającym zmianę stanu układu, zaś Michał Kasprzyk pracował głównie nad *frontendem* - modułem wizualizacji, graficznym interfejsem użytkownika. Były to jednak ramy dość umowne, ponieważ obydwaj często wykonywaliśmy pomniejsze zadania w obszarach innych niż wyżej wymienione. Ponadto wszelkie większe zadania, niezależnie od tego, kto dokonywał właściwej implementacji, były wzajemnie konsultowane, aby poznać spojrzenie drugiej osoby na daną część aplikacji, na jej budowę wewnętrzną, zewnętrzny interfejs oraz wymagania, które ma spełniać. System ten utrzymywany był do końca procesu tworzenia oprogramowania. W trakcie prac wyróżniane były kolejne zadania do wykonania, następnie były one omawiane i implementowane.

Pierwotnie zastanawialiśmy się nad wykorzystaniem jakiegoś rozbudowanego system zarządzania zadaniami, ostatecznie jednak na żaden z nich się nie zdecydowaliśmy. Wynikło to prawdopodobnie z faktu, że w przypadku projektu prowadzonego przez dwie osoby łatwo jest konsultować stan prac. W naszym zespole nie mieliśmy też wyróżnionego lidera, który miałby zarządzać zadaniami i przypisywać je do poszczególnych osób. W przypadku regularnej pracy w kilkuosobowym zespole narzędzie takie byłoby niezbędne, tutaj jednak nie znalazłoby odpowiedniego zastosowania.

5. Etapy pracy

W zgodzie ze sposobem organizacji prac, przeprowadzane były poszczególne etapy projektu. Każdy z etapów zakładał ukończenie części aplikacji, którą zajmowaliśmy się w tym etapie . Poniżej podano wszystkie iteracje wraz z szacunkowymi miesiącami wykonywania, bądź rozpoczęcia danego etapu oraz krótkie opisy tego, czego one dotyczyły i jak przebiegały.

Pomiędzy etapami, w których dodawane były nowe funkcjonalności odbywały się jedno, bądź dwutygodniowe okresy, podczas których kod był sprzątany i refaktoryzowany. Usuwane były rozwiązania zastosowane tymczasowo, tylko po to, żeby sprawdzić, czy będą działać, a zastępowane były bardziej zgodnymi z dobrymi zasadami tworzenia oprogramowania.

Pomiędzy niektórymi z etapów odbywały się również spotkania z klientem mające na celu prezentację postępów prac, jak również uściślenie wymagań dotyczących funkcjonalności, które mieliśmy zamiar implementować w następnej kolejności.

5.1. Etap wstępny - stworzenie wizji projektu, wybór nazwy kodowej, założenie repozytorium, wstępny podział zadań (maj, czerwiec 2015)

Początek prac nad projektem zakładał typowe dla tego rodzaju przedsięwzięć działania, między innymi stworzenie wizji projektu, przeprowadzenie analizy ryzyka oraz przyrządzenie studium wykonywalności. Po zdefiniowaniu funkcjonalności aplikacji i zidentyfikowaniu zagrożeń mogących wystąpić podczas jej implementacji.

Ponadto należało wybrać odpowiednią nazwę kodową. W idei pozwalać ma ona na szybką identyfikację projektu. Powinna być krótką i zwięzła, a jednocześnie zawierającą pewną ilość informacji o zawartości projektu oraz być w jakiś sposób związaną z jego tematyką. W celu ułatwienia jej wyboru skorzystaliśmy z pomocy kilku serwisów internetowych generujących akronimy z podanych słów. Szukaliśmy odpowiadającego nam skrótowca powstałego z części tytułu projektu w języku angielskim: flexible objects modelling. Spośród kilkuset

wygenerowanych słów wybraliśmy jedno, które naszym zdaniem najlepiej opisywało charakter pracy, a jednocześnie posiadało ładne brzmienie. Ostatecznie zdecydowaliśmy się na *FlexO.*
Następnym wymaganym krokiem było założenie repozytorium projektu. Powstało ono w serwisie *GitHub*. Jego nazwą jest nazwa kodowa projektu. Ostatecznie nastąpił wstępny podział obszarów aplikacji pomiędzy członków zespołu. Umożliwiło to przystąpienie do dalszej pracy.

5.2. Wybór odpowiedniej technologii do wizualizacji (czerwiec 2015)

Po stworzeniu poglądowego szkieletu graficznego interfejsu użytkownika rozpoczęliśmy próby integracji kodu stworzonego z wykorzystaniem technologii *JavaFX* z obszarem zdolnym do wyświetlania obiektów *WebGL*. Okazało się, że pogodzenie tych dwóch technologii nie jest zadaniem tak trywialnym, jak mogłoby się wydawać. Najprostszym sposobem, jaki udało nam się znaleźć byłoby otwarcie wewnątrz naszego programu fragmentu okna przeglądarki, za pomocą którego mielibyśmy wyświetlać wizualizacje. Jednak o ile niektóre strony internetowe działały poprawnie, o tyle elementy WebGL na stronach, czy w przykładowych plikach nie były wyświetlane. Mimo prób nie udało nam się znaleźć zadowalającego rozwiązania. Ostatecznie zdecydowaliśmy się na znalezienie innej technologi pozwalającej na prezentację obliczeń naszej aplikacji.

Kolejnym rozwiązaniem, które zaczęliśmy rozważać było wykorzystanie technologii *OpenGL*. Udało nam się znaleźć bibliotekę *JOGL* umożliwiającą tworzenie *canvas OpenGL* wewnątrz interfejsu użytkownika stworzonego w technologiach takich, jak *Swing*, czy *AWT*. Metoda ta jednak nie współpracowała z wybranym przez nas *JavaFX*. Powodowała też problemy z zależnościami w stworzonym przez nas projekcie.

W trakcie poszukiwań biblioteki zdolnej do współpracy z *JavaFX* okazało się, że biblioteka ta sama w sobie pozwala na wyświetlanie elementów trójwymiarowych i dostarcza podstawowe, trójwymiarowe modele. Udostępnia ona API przypominające *WebGL* z poziomu *Javy*, dzięki czemu łatwe jest łączenie komponentów odpowiedzialnych za kontrolę,

takich, jak przyciski czy okienka, z modułem wizualizacji. Technicznie są elementami tego samego typu. Mając na uwadze poprzednie niepowodzenia zdecydowaliśmy się skorzystać z tego rozwiązania podczas dalszego rozwoju projektu.

Wybór i testowanie funkcjonalności technologii wizualizacji było najbardziej czasochłonnym etapem prac w początkowej fazie projektu. Dlatego też opis owego procesu jest dość rozległy. Posiadając działający prototyp modułu wizualizacji, możliwe było rozpoczęcie pracy nad pozostałymi komponentami.

5.3. Stworzenie podstawowej architektury aplikacji (czerwiec 2015)

Podczas tworzenia architektury aplikacji musieliśmy rozważyć, jakie rozwiązania są najbardziej optymalne dla rozwiązywanego problemu, a ponadto w jaki sposób będą się one wiązać z zastosowaną przez nas technologią.

W naturalny sposób zwróciliśmy się ku rozwiązaniu *Model – Widok – Kontroler*. *Widok* oraz *Kontroler* są domyślnymi elementami aplikacji posiadających graficzne interfejsy użytkownika stworzone w technologii *JavaFX*, w związku z czym implementacja wzorca jest znacznie ułatwiona.

W naszej aplikacji *Model* składa się z klas odpowiadających za punkty znajdujące się na powierzchni bryły, węzły, oraz połączenia między punktami. Zawierają one dodatkowe parametry, które zmieniają wpływające na bryłę, jak na przykład moduł Younga.

Widok odpowiada za wygląd interfejsu użytkownika oraz samej wizualizacji. Poprzez dostarczenie odpowiednich, wyświetlonych elementów pozwala wpływać na stan układu poprzez zmianę pewnych jego parametrów.

Kontroler dostarczony przez *JavaFX* pozwala na rozdzielenie warstw *Modelu* i *Widoku*. Zmienia on stan *Modelu* w zależności od zmian poczynionych w *Widoku*. Istnienie *Kontrolera* gwarantuje, że nie wystąpią bezpośrednie zależności pomiędzy pozostałymi warstwami, a, co za tym

idzie, nie zostaną wykonane żadne operacje, które nie zostały uwzględnione w logice programu uwzględnionej w tej części aplikacji.

5.4. Implementacja modelu danych (wrzesień 2015)

Model danych jest fundamentem wszelkiej logiki aplikacji, toteż musiał zostać zaimplementowany na samym początku procesu tworzenia projektu. Jego zadaniem jest przechowywanie w obiektach danych wykorzystywanych podczas wykonywania obliczeń przez aplikację. Stanowi również podstawę do stworzenia modułu wizualizacji, gdyż stan obiektów w modelu jest przedstawiany za pomocą odpowiadających mu obiektów modułu wizualizacji. Ponadto jest wykorzystywany do zapisywania lub wczytywania układu z pliku.

Etap implementacji był jednym z najmniej złożonych w procesie tworzenia aplikacji. Wymagał jedynie zaprojektowania niezbyt skomplikowanej hierarchii klas przechowujących dane, a następnie ich zaimplementowania. Oprócz posiadania pól zawierających przechowywane dane oraz pewnych metod, pozwalających na łatwy dostęp do nich, nie zawiera żadnej logiki wykonującej bardziej złożone operacje.

5.5. Wprowadzenie modułu wizualizacji (wrzesień, listopad 2015)

Szczegółowy opis konstrukcji modułu wizualizacji znajduje się w dokumentacji technicznej. Ponieważ jednak był to moduł, którego implementacja okazała się nie lada wyzwaniem, zostanie mu poświęcone dość dużo uwagi również w tym dokumencie.

Na wstępie należy zaznaczyć, że moduł ten tworzony był w oparciu o technologię znajdującą się ciągle w fazie rozwoju. Z jednej strony technologia ta jest wygodna w użyciu i umożliwia uzyskania estetycznych wizualnie rezultatów za pomocą używania API wysokiego poziomu, co znacząco upraszcza i przyspiesza pracę. Z drugiej jednak, pewne niedopracowania, czy może przede wszystkim niepełna znajomość młodej technologii, utrudniały sfinalizowanie pracy nad

modułem i przyczyniły się do zauważalnego opóźnienia w planowanych postępach prac.

Podstawy pod moduł zostały położone już podczas wyboru odpowiedniej technologii wizualizacji. Wtedy powstał pierwszy prototyp modułu, który był w stanie otworzyć okno aplikacji i wyświetlić w nim kilka nieruchomych obiektów, które uprzednio zostały na sztywno ustawione w kodzie aplikacji.

Kolejnym etapem rozwoju modułu było dodanie możliwości przesuwania oraz obracania kamery wokół środku wizualizowanego układu, a także umożliwienie przybliżania i oddalania widoku za pomocą kółka myszki. Okazało się to być pewnym wyzwaniem, bowiem biblioteka dostarcza kilku rozwiązań na przykładowe ‘przesunięcie kamery’, a znalezienie tej właściwej nie należało do zadań najłatwiejszych. Wprowadzona została również opcja zaznaczania wizualizowanych obiektów, co powodowało zmianę ich koloru.

Wtedy wyszedł na jaw jeden z największych i najbardziej czasochłonnych do rozwiązania problemów, jakie napotkaliśmy w czasie pracy nad projektem. Podczas obracania kamery można było zauważyć, że obiekty wyświetlane są na ekranie w kolejności takiej, jak zostały stworzone. To znaczy, że najmłodsze obiekty zawsze były na wierzchu, a najstarsze zawsze na spodzie, niezależnie od tego, pod jakim kątem skierowana była kamera. Dość szybko udało nam się poprawnie nazwać problem - dotyczył on bufora głębokości. W teorii powinno było dać się go włączyć lub wyłączyć w bardzo prosty sposób, jednak zmiana nie przynosiła żadnego skutku. Rozwiązaniem okazało się w zasadzie znalezienie błędu dotyczącego ustawień kamery. Właściwość *near clip* została ustawiona na zbyt małą wartość co skutkowało niepoprawnym działaniem bufora głębokości.

Po rozwiązaniu problemu z buforem głębi dodany został antyaliasing, dzięki któremu wizualizowane obiekty stały się gładsze. Do tej pory mówiąc o wizualizowanych obiektach mieliśmy na myśli kule, reprezentujące węzły znajdujące się w modelu danych. Następnym krokiem było przedstawianie połączeń pomiędzy węzłami. Niestety w wśród kilku dostępnych, prostych, trójwymiarowych modeli, zabrało cylindrów zdolnych do połączenia dwóch konkretnych, wskazanych punktów. Wymagało to zatem samodzielnej implementacji z

wykorzystaniem obracania, przesuwania i zmiany rozmiaru cylindrów. Dokładny sposób obliczania położenia cylindrów przedstawiony jest w dokumentacji technicznej.

5.6. Opracowanie i implementacja algorytmu (grudzień 2015)

Szczegółowy opis działania algorytmu obliczającego odkształcenia bryły znajduje się w dokumentacji technicznej projektu. Opis ten można streścić w następujący sposób: algorytm jest zachłanny, przesuwając każdy węzeł w miejsce, w którym działa na niego najmniejsza wypadkowa siła pochodząca od sąsiednich węzłów. Po przesunięciu wszystkich węzłów sprawdzana jest równowaga układu, to jest, czy działające na każdy z punktów siły równoważą się. Jeżeli tak jest, oznacza to, że obliczony został pożądany stan końcowy i wynik może zostać przedstawiony użytkownikowi. Jeśli jednak któryś z węzłów nie jest zrównoważony, wszystkie powyższe kroki są kontynuowane do skutku.

Opracowanie powyższego algorytmu, pomimo jego prostoty było jednym z bardziej czasochłonnych zajęć podczas prac nad projektem. Był on wynikiem kilku spotkań podczas których prezentowaliśmy wzajemnie pewne pomysły na temat tego, jak ów algorytm miałby działać, długo jednak nie mogliśmy dojść do porozumienia. Finalna postać modułu zajmującego się obliczaniem odkształceń po około dwóch tygodniach, podczas których odbyło się kilka parogodzinnych spotkań, na których omawialiśmy w dużej mierze kwestię algorytmu.

Sama implementacja również okazała się być dość czasochłonnych przedsięwzięciem, głównie z powodu nieprzewidzianych dylematów wynikających z chęci utrzymania modułu obliczeniowego w stylu tworzenia oprogramowania w wybranej technologii. Dodatkowe problemy wyniknęły z powodu trudności w dokładnym zdefiniowaniu sposobu obliczania sił występujących w układzie.

5.7. Implementacja modułu zapisu i wczytywania układu z pliku (grudzień 2015)

Zapis i odczyt układu z pliku jest jedną z kluczowych funkcjonalności aplikacji. Jeszcze przed jej implementacją należało podjąć decyzję o tym w jakim formacie mają zostać zapisane pliki aplikacji. W zasadzie już na wstępie zdecydowano, że układy powinny być zapisywane do plików w formacie *.xml. P*ozwala to użytkownikowi na ich podgląd i edycję, a także stworzenie dowolnych układów początkowych nie przewidzianych w obecnej wersji aplikacji. W module wykorzystywana jest technologia *JAXB - Java Architecture for XML Binding*. Do poprawnego jej działania do odpowiednich pól i metod zostały dodane odpowiednie adnotacje.

Ponadto aby umożliwić użytkownikowi wczytywanie lub zapisywanie plików wymagane było zmodyfikowanie głównego okna aplikacji. Dodany został pasek menu wraz z odpowiednimi wpisami. Powstało również odpowiednie podmenu pozwalające użytkownikowi na stworzenie nowego układu wybranego typu. Po wybraniu odpowiedniego typu przez użytkownika wyświetlane jest dodatkowe okienko pozwalające na sprecyzowanie z ile węzłów składać się ma układ. W praktyce jest to jedno z okien dialogowych dostarczanych przez *JavaFX*.

5.8. Umożliwienie eksportu układu do pliku *.obj* (styczeń 2015)

Etap wprowadzenia funkcjonalności eksportu układu do pliku .*obj* w dużej mierze opierał się na znalezieniu odpowiedniego rozwiązania tego problemu. *JavaFX* nie uwzględnia możliwości eksportu elementów trójwymiarowych - przynajmniej nie do tego formatu. Wymagane zatem było skorzystanie z dodatkowej biblioteki.

Pierwszą znalezioną biblioteką zdolną do takiego eksportu była *FXyz*. Rozszerza ona możliwość *JavaFX* między innymi dodając nowe elementy trójwymiarowe, jak również oferuje funkcjonalności podobne do poszukiwanej. Pojawiło się jednak kilka problemów związanych z tą biblioteką. Po pierwsze metoda odpowiedzialna za zapisywanie do pliku .*obj* wymagała obiektu klasy *Mesh*. Klasa ta jest dostępna w *JavaFX*, jednak twórcy nie udostępnili możliwości uzyskania obiektów *Mesh* z

wykorzystaniem obiektów klas *Sphere* czy *Cylinder*, które są wykorzystywane w projekcie. Ponadto dodanie tej biblioteki do projektu nie było trywialne.

Drugą biblioteką zdolną rozwiązać ten problem okazała się *JCSG*. Jej główną atrybutem jest możliwość wykonywania operacji binarnych na obiektach trójwymiarowych. Nie było to w kwestii eksportu kluczowe, jednak okazało się przydatne do połączenia wszystkich elementów w jeden reprezentujący cały układ. Aby to zrobić trzeba było jednak stworzyć nowe obiekty odpowiedników klas *Sphere* oraz *Cylinder* w tej bibliotece, a następnie z każdego z nich stworzyć obiekt klasy *CSG* dostarczany przez bibliotekę. Było to potrzebne do stworzenia jednego wspólnego obiektu, który następnie mógł zostać zapisany do pliku w formacie *.obj*. Poniżej znajduje się zrzut ekranu z programu Blender po imporcie przykładowego, wyeksportowany układu.