UNIWERSYTET ŚLĄSKI W KATOWICACH
WYDZIAŁ INFORMATYKI I NAUKI O MATERIAŁACH
ZAKŁAD SYSTEMÓW KOMPUTEROWYCH

ADRIAN NIEWIADOMSKI
6135

Metody generowania przekrojów pionowych obiektów 3D

PRACA DYPLOMOWA INŻYNIERSKA

PROMOTOR
DR KRZYSZTOF WRÓBEL

SOSNOWIEC 2019

Słowa kluczowe: Three.js, przekrój, wektory normalne

Oświadczenie autora pracy

Ja, niżej podpisany:

imię (imiona) i nazwisko Adrian Niewiadomski
autor pracy dyplomowej pt. Metody generowania przekrojów pionowych obiektów 3D
Numer albumu: 6135
Student Wydziału Informatyki i Nauki o Materiałach Uniwersytetu Śląskiego w Katowicach
kierunku studiów Informatyka inżynierska
specjalności
……...

Oświadczam, że ww. praca dyplomowa:

- została przygotowana przeze mnie samodzielnie[1],
- nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tekst jednolity Dz. U. Z 2006 r. Nr 90, poz. 631, z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- nie zawiera danych i informacji, które uzyskałem/łam w sposób niedozwolony,
- nie była podstawą nadania dyplomu uczelni wyższej lub tytułu zawodowego ani mnie, ani innej osobie.

Oświadczam również, że treść pracy dyplomowej zamieszczonej przeze mnie w Archiwum Prac Dyplomowych jest identyczna z treścią zawartą w wydrukowanej wersji pracy.

Jestem świadomy odpowiedzialności karnej za złożenie fałszywego oświadczenia.

…….. ……..
Data Podpis autora pracy

[1]uwzględniając merytoryczny wkład promotora (w ramach prowadzonego seminarium dyplomowego)

Spis treści

1. Wprowadzenie.

W obecnych czasach komputeryzacja wchodzi w coraz to szersze dziedziny ludzkiej działalności i upraszcza je pozwalając na większą wygodę i wzrost wydajności. Od pewnego czasu komputerowo wspomagane jest tworzenie projektów inżynierskich oraz architektonicznych. Techniki cyfrowe wykorzystywane są też w większości metod pomiarów medycznych wymagających wizualizacji wyników. Pozwalają np. na wizualizację dostępnych złóż oraz analizę terenu w przekrojach geologicznych. Ponadto bez komputerów trudno jest sobie wyobrazić dzisiaj takie branże przemysłu rozrywkowego, jak choćby przemysł filmowy, a tworzenie gier wideo byłoby niemożliwe.

We wszystkich wspomnianych wyżej zastosowaniach istotną rolę odgrywa grafika komputerowa. Pozwala ona na wyświetlanie na ekranie komputera czy telewizora obiektów będących pewnym odwzorowaniem obiektów rzeczywistych. Obiekty te z uwagi na naturę naszego świata są oczywiście trójwymiarowe. Niekiedy, np. w projektowaniu elementów mechanicznych lub w analizach geologicznych, zachodzi potrzeba tworzenia przekrojów wyświetlanych obiektów. Aktualnie dostępne komercyjne oprogramowanie (jak na przykład Paradigm GOCAD, Petrel czy EarthVision) potrafi wykonywać przekroje obiektów geologicznych jednakże jest ono drogie oraz stawia zazwyczaj wysokie wymagania używanemu sprzętowi komputerowemu. Dlatego, opracowanie i zaimplementowanie algorytmu pozwalającego na tworzenie poprawnie wyświetlanych przekrojów obiektów 3d jest pierwszym z celów niniejszej pracy.

Kolejnym istotnym problemem spotykanym w grafice komputerowej jest zagadnienie poprawnego wyświetlania ścian obiektu. Jak zostanie przedstawione w podrozdziałach 2.1 i 2.2. ściany komputerowych obiektów 3d mają swoją orientację. Jeśli jest ona nieprawidłowa obiekt będzie wyświetlany błędnie. Na przykład, wśród obiektów zawartych w bazach geologicznych występują obiekty posiadające ściany o niewłaściwej orientacji. Z tego względu zachodzi potrzeba opracowania algorytmu poprawiającego orientację ścian obiektów 3d, co stanowi drugi z celów tej rozprawy.

Niniejsza praca składa się z ośmiu rozdziałów. Pierwszy stanowi niniejszy wstęp. W rozdziale drugim została omówiona problematyka poruszana w dalszej części pracy. Trzeci rozdział ma na celu krótkie omówienie historii możliwości wyświetlania grafiki 3d

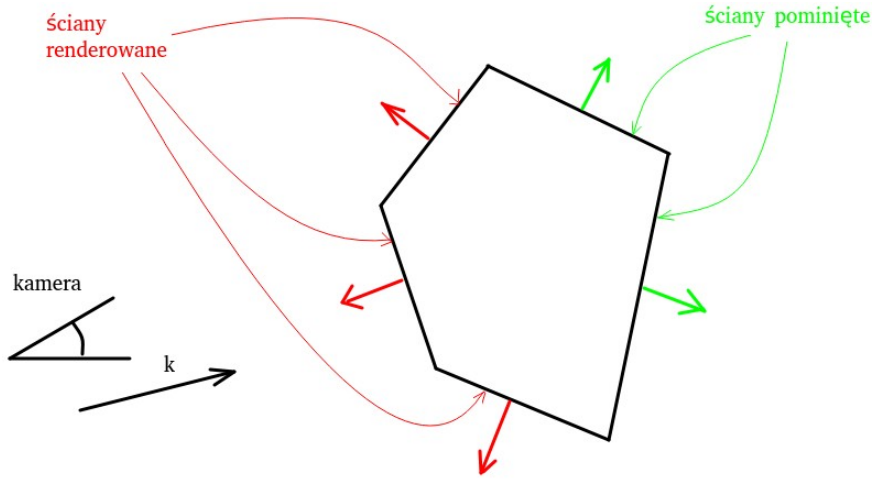
na stronach WWW. Kolejne rozdziały zawierają rezultaty pracy własnej autora. W rozdziale czwartym przedstawiono projekt systemu informatycznego, przy czym szczególny nacisk położono na opis funkcjonalności dostępnych dla użytkowników. W kolejnym rozdziale zaproponowano algorytmy mające rozwiązywać postawione w pracy problemy. Struktura aplikacji oraz interfejs użytkownika został przedstawiony w rozdziale szóstym. Szczegóły implementacji algorytmów oraz osiągnięte rezultaty zostały umieszczone w rozdziale siódmym. Ósmy, ostatni rozdział stanowi podsumowanie pracy.

2. Problematyka poruszana w pracy.

Jak już wspomniano wcześniej, niniejsza praca ma na celu opracowanie i implementację algorytmu tworzącego pionowe przekroje obiektów 3d oraz algorytmu poprawiającego kolejność wierzchołków w ścianach tych obiektów. Rozdziały 2.1 i 2.2 mają przybliżyć czytelnikowi problematykę związaną z tymi zagadnieniami.

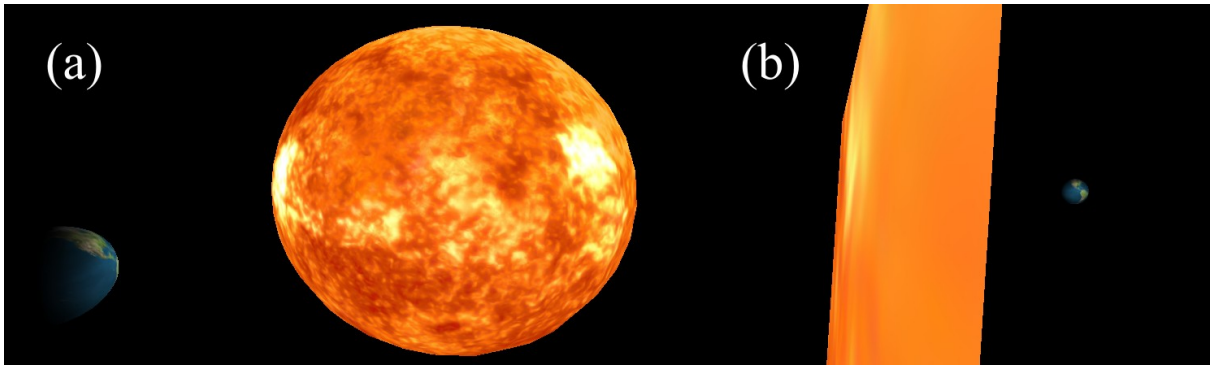
2.1. Obiekty grafiki 3d i generowanie ich przekrojów.

Obiekty grafiki komputerowej w największym uproszczeniu składają się z wierzchołków, czyli tablicy zawierającej współrzędne punktów w przestrzeni 3d oraz tablicy ścian zawierającej numery wierzchołków tworzących/wchodzących w skład danej ściany. Powoduje to, że obiekty grafiki 3d są puste w środku. Na podstawie kolejności wierzchołków danego trójkąta wyznaczany jest jego wektor normalny, będący jego orientacją [1]. Podczas tworzenia obrazu wykorzystywana jest orientacja ścian rysowanych obiektów. Mianowicie, jeśli wektor normalny jest zwrócony w kierunku obserwatora, to ściana jest rysowana. Jeżeli natomiast jest on zwrócony w kierunku przeciwnym, wówczas przyjmowane jest, że ściana jest "tyłem" do obserwatora i nie jest ona rysowana [2]. Schemat działania algorytmu przedstawiono na rysunku 1.



Rys. 1. Schemat renderowania i pomijania ścian obiektu 3d. Jeśli iloczyn skalarny kierunku zwrócenia kamery k oraz wektora normalnego danej ściany jest mniejszy od 0 to ściana jest rysowana. W przeciwnym wypadku jest ona pomijana podczas renderowania.

Przedstawiony algorytm wraz z faktem, że obiekty grafiki 3d są w środku puste powoduje, że w celu uzyskania przekrojów tych obiektów nie można ich po prostu obciąć. W takim wypadku przez wnętrze renderowanego obiektu zobaczylibyśmy obiekty położone za obiektem albo samo wnętrze tego obiektu, którego przekrój robimy. Taka sytuacja jest przedstawiona na rysunku 2b. Widzimy obiekt za obiektem, dla którego nie renderujemy przednich ścian.

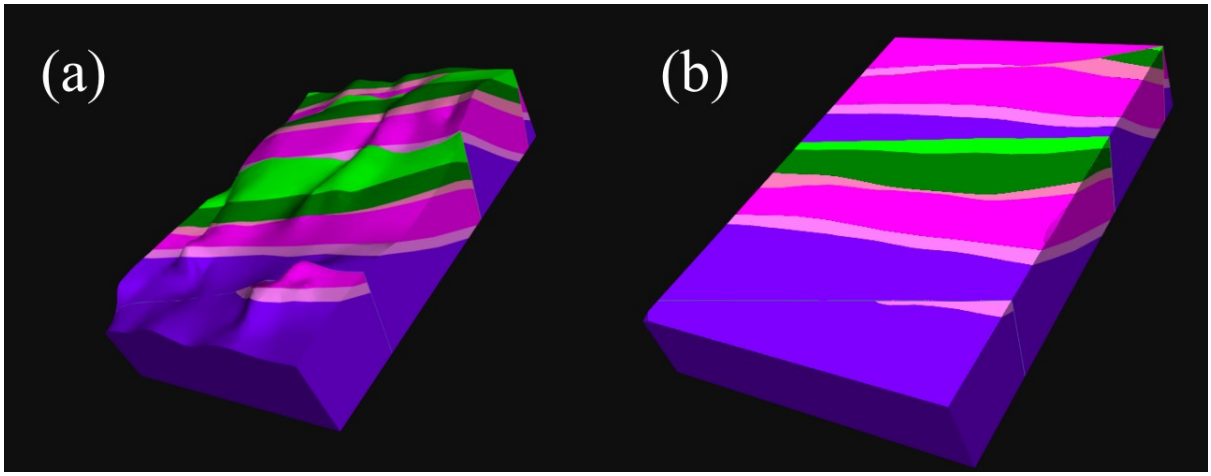


Rys. 2. Dwa obiekty grafiki 3d (Słońce i Ziemia). (a) Widok obiektów z zewnątrz. (b) Widok po przybliżeniu. Przez wnętrze i tylne ściany Słońca widoczna jest Ziemia. Wynika to z faktu, że obiekty grafiki 3d są puste w środku.

Jeśli tworzona aplikacja wykorzystuje bibliotekę Three.js, w przypadku braku cieniowania i bez teksturowania można starać się rozwiązać ten problem za pomocą właściwości materiału THREE.DoubleSide, która pozwala na renderowanie ścian zwróconych również w „drugą" stronę – od obserwatora. Pod płaszczyzną przekroju widzimy wówczas tylne ściany wyglądającą jak wnętrze obiektu. Brak cieniowania jednak uniemożliwia stworzenie wrażenia trójwymiarowości obiektów. Dwa obiekty, których część obcięto przy zastosowaniu tzw. clippingPlanes przedstawiono na rysunku 3. Przykłady zastosowania clippingPlanes można znaleźć w dokumentacji biblioteki Three.js [3].

Rys. 3. Obiekty obcięte przy użyciu narzędzi biblioteki Three.js. W obu przypadkach obcięto ściany przednie i użyto obustronnego materiału. Dla obiektu po stronie lewej użyto materiału podstawowego-stąd brak cieniowania i wrażenia trójwymiarowości. W przypadku obiektu po stronie prawej widzimy ściany wewnętrzne.

W swojej pracy inżynierskiej, absolwent Instytutu Informatyki UŚ, Maciej Ciosk używa innego sposobu generowania przekrojów poziomych obiektów geologicznych. Mianowicie używa On dwóch scen i trzech kamer. Pierwsza kamera jest kamerą perspektywiczną, która znajduje się na scenie głównej i generuje obraz ściętego obiektu. Druga i trzecia kamera to kamery ortogonalne, które znajdują się na scenie buforowej. Są one umieszczone odpowiednio nad i pod modelem. Kamera znajdująca się nad modelem obcina obiekt od miejsca przekroju do góry. Natomiast ta znajdująca się pod modelem obcina obiekt w dół. Następnie program porównuje otrzymane obrazy i tworzy teksturę nakładaną na obcinany obiekt na scenie głównej. Tekstura generowana jest w następujący sposób: Na podstawie obrazu drugiej kamery tworzona jest tekstura wyjściowa. Miejsca na obrazie z trzeciej kamery, które są czarne reprezentują obszar gdzie nie ma obiektu i są ustawiane jako przezroczyste na teksturze. Zainteresowany czytelnik znajdzie dokładniejszy opis procedury wraz z obrazami przekrojów we wspomnianej wcześniej pracy dyplomowej [4]. Wybrany przekrój otrzymany tą metodą przedstawiono na rys. 4b.

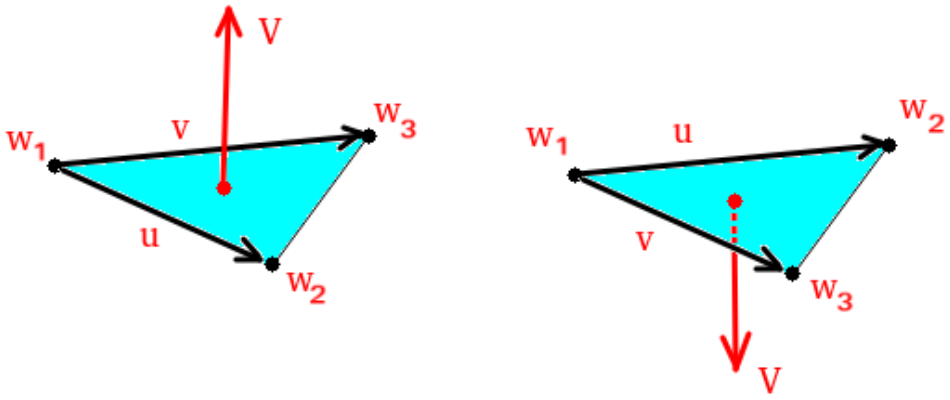


Rys. 4. Zbiór obiektów 3d „mapa ćwiczeniowa”. (a) Widok mapy bez tworzenia przekroju. (b) Widok uzyskanego przekroju poziomego. Źródło: M. Ciosk, „Projektowanie webowych aplikacji graficznych w technologii WebGL”, Sosnowiec 2018.

Na podstawie rysunków 3 i 4 mogło by się wydawać, że utworzenie przekroju może być wykonane za pomocą obcięcia obiektu i nałożenia tekstury uzyskanej z drugiej kamery lub też metodą opisaną skrótowo powyżej. Tak jednak nie jest. Metoda prezentowana w [4] nie może być użyta dla uzyskania dowolnego przekroju, gdyż w ogólności kamery druga i trzecia będą widzieć także te części obiektu, które wystają ponad (i pod) obiekt z tyłu. Uniemożliwia to poprawne wygenerowanie tekstury.

2.2. Wektory normalne i ich znaczenie w grafice komputerowej.

Jak wspomniano w poprzednim podrozdziale, obiekt 3d składa się z wierzchołków i ścian zawierających numery wierzchołków. Okazuje się, że kolejność wierzchołków w ścianie nie jest bez znaczenia. Algorytm wyznaczający wektory normalne sprowadza się do wyznaczenia dwóch wektorów **u** i **v** będących różnicami odpowiednio współrzędnych wierzchołków w_2 oraz w_3 i wierzchołka w_1. Następnie obliczany jest iloczyn wektorowy tych wektorów. Wynik iloczynu to poszukiwany wektor normalny. Z właściwości iloczynu wektorowego wiadomo, że zamiana kolejności **u** i **v** skutkuje zmianą zwrotu wektora normalnego na przeciwny (rysunek 5). Widzimy zatem, że wierzchołki muszą być ustawione w kolejności przeciwnej do wskazówek zegara [5].



Rys. 5. Wpływ kolejności wierzchołków w ścianie na zwrot wektora normalnego (orientacji ściany).

W przypadku obiektów omawianych w niniejszej pracy wszystkie one będą oglądane z zewnątrz. W takim wypadku wszystkie wektory normalne obiektu powinny być zwrócone na zewnątrz. W przeciwnym wypadku obiekt nie będzie wyświetlany poprawnie i może być zniekształcony. Ponadto zdarza się, że obiekt zachowuje się w sposób nieoczekiwany, np. jeśli chcemy obracać obiekt wokół własnej osi to wygląda on jakby wykonywał obroty w przeciwnym kierunku. Dla obiektów zamkniętych i wypukłych można by wyznaczać środek obiektu i środek ściany. Następnie tworzyć wektor od środka obiektu do ściany. Na podstawie iloczynu skalarnego tego wektora oraz wyznaczonego wektora normalnego można by sprawdzać, czy wynik jest większy od zera. Jeśli nie, to należy obrócić wektor (zmienić jego zwrot na przeciwny) oraz zamienić kolejność wierzchołków w ścianie. Takie rozwiązanie jednak jest ograniczone jedynie do najprostszych obiektów 3d.

3. Rozwój grafiki na stronach internetowych.

Historia grafiki 3d na stronach internetowych sięga roku 1994, kiedy to jeden z projektantów sieci WWW, Dave Raggett wprowadził język modelowania wirtualnej rzeczywistości **VRML** (Virtual Reality Modeling Language). Język ten został stworzony z myślą o tworzeniu trójwymiarowych stron internetowych. Kolejne wersje języka pozwalały na tworzenie wirtualnych światów wraz z animacjami i dźwiękami [6]. Ponadto był on wykorzystywany przy tworzeniu symulacji, wizualizacjach projektów architektonicznych, prezentacji procesów przemysłowych, zjawisk fizycznych, chemicznych i biologicznych itp. Możliwości VRML były dostępne dla użytkownika przy użyciu przeglądarki internetowej, po zainstalowaniu odpowiedniej wtyczki. Mimo sporego zainteresowania, VRML nigdy nie był intensywnie wykorzystywany, m.in. ze względu na niewielkie prędkości łącz Internetowych osiąganych w latach 90 tych.

W roku 2001, sukcesorem języka VRLM został język **X3D**. Jest on rozszerzeniem języka VRLM, który umożliwia stosowanie go w wizualizacjach inżynierskich CAD czy druku 3d [7]. Pozwala on również na kodowanie sceny w formactach XML, JSON i binarnym. Może pośredniczyć w przesyłaniu informacji o modelach 3d między różnymi aplikacjami. Wspiera różne modele oświetlenia, teksturowanie i cieniowanie dostępne w przeglądarkach internetowych w czasie rzeczywistym.

Równolegle, w roku 1995, FutureWave Software wydało program FutureSplash Animator będący edytorem grafiki wektorowej. Program, napisany przez Jonathan'a Gaya, tworzył animacje przy wykorzystaniu grafiki wektorowej. Umożliwiał on projektowanie witryn WWW z prostymi animacjami o dobrej jakości, przy ich niewielkim rozmiarze plików [8,9]. FutureSplash został później nabyty przez firmę Macromedia, która zmieniła jego nazwę na **Flash**. System ten składał się z dwóch części: odtwarzacza (Macromedia Flash Player) oraz edytora grafiki i animacji (Macromedia Flash). Macromedia udostępniała Flash Playera jako darmową wtyczkę do przeglądarek internetowych. W rezultacie w 2005 roku Macromedia Flash Player był najpopularniejszą sieciową technologią multimedialną. Następnie, Macromedia zostało wykupione przez Adobe Systems. Do wad odtwarzacza należy zaliczyć m. in. konieczność odinstalowania starszych wtyczek przed zainstalowaniem uaktualnienia w niektórych przeglądarkach czy zbieranie danych o użytkownikach.

Realizację funkcji interaktywnych w środowisku wykonawczym Adobe Flash Player umożliwia język **Action Script**. Początkowo był on projektowany pod kątem kontrolowania prostych dwuwymiarowych animacji utworzonych w środowisku Adobe Flash. Aktualnie dodano funkcjonalności pozwalające na tworzenie gier i aplikacji internetowych. Jest to obiektowy język programowania, którego model zdarzeń oparty jest na specyfikacji Document Object Model (DOM). Jego kod wykonywany jest na maszynie wirtualnej wchodzącej w skład środowiska Flash Player [10]. Dzięki integracji ze środowiskiem ułatwione jest sprawne debugowanie kodu [11].

W styczniu 2008 roku opublikowany został projekt roboczy języka **HTML5**. Standard wprowadził nowe narzędzia poprawiające wydajność przez optymalizację wykorzystania zasobów komputera [12]. Jednym z tych narzędzi jest obiekt XMLHttpRequest pozwalający na asynchroniczną zmianę części strony. Pojawiły się także nowe znaczniki pozwalające na rysowanie grafik oraz odtwarzanie dźwięku i klipów wideo. Przeglądarka stała się multimedialną platformą o wydajności porównywalnej z programami macierzystymi [13]. Jak wspomniano wyżej, jeden z tych znaczników - element Canvas może być użyty do rysowania grafik. Wykonywane jest to za pomocą skryptów.

Wspomniane skrypty pisane są w języku **JavaScript** (JS), który aktualnie jest przyjmowany za jeden z języków (HTML, CSS i JavaScript), które znać musi każdy web developer [14]. JS został opracowany w 1995 roku przez Brendan'a Eicha dla firmy Netscape Corporation [15]. Jest to język zorientowany obiektowo, który został zaprojektowany tak aby mógł on być mieszany z kodem HTML i jest wykorzystywany do dynamicznego aktualizowania stron WWW. Zapewnia to interakcję poprzez reagowanie na zdarzenia wywołane przez użytkownika strony. Ponadto kod jest wykonywany przez przeglądarkę po stronie klienta dzięki czemu nie zachodzi konieczność odświeżania strony WWW [16].

W roku 2006 odbyły się pierwsze demonstracje **WebGL** (Web Graphics Library), będącego interfejsem programistycznym aplikacji (API) języka JS. WebGL jest wieloplatformowy i nie wymaga instalacji żadnych wtyczek a jedynie przeglądarki internetowej [4,17,18]. Został on oparty na OpenGL ES 2.0 i może być wykorzystany do pracy z elementem Canvas. Umożliwia on renderowanie interaktywnej grafiki 3d wspieranej sprzętowo [13,19]. Nie jest on jednak narzędziem wygodnym z uwagi na jego niskopoziomowy charakter. Z tego względu wykorzystywane są biblioteki JS ułatwiające pracę z WebGL.

Najpopularniejszą z tych bibliotek jest stworzona przez Ricardo Cabello (Mr.doob) **Three.js**. Umożliwia ona łatwy dostęp do API WebGL i oferuje zestaw obiektów wykorzystywanych do programowania grafiki 3d [13]. Biblioteka ta jest bogata w funkcje użyteczne przy tworzeniu grafiki 3d jak na przykład funkcje matematyczne wykonujące obliczenia m.in. na wektorach i macierzach, zawiera loadery plików oraz posiada mechanizmy interakcji. Ponadto jest ona szybka, stosunkowo prosta do nauczenia i ma ona obszerną dokumentację wraz z szeregiem przydatnych przykładów [20].

4. Projekt systemu.

Jak zostało określone w rozdziale 1, celami niniejszej pracy dyplomowej jest opracowanie algorytmu tworzącego przekroje obiektów 3d oraz algorytmu poprawiającego kolejność wierzchołków w ścianach tych obiektów. Ponieważ jednak oba algorytmy należy zaimplementować w ramach tworzonego systemu informatycznego, warto jest określić wymagania stawiane przed samym systemem oraz dostępne dla użytkowników funkcjonalności.

4.1. Wymagania aplikacji.

Do **wymagań funkcjonalnych** należy zaliczyć dostępność następujących funkcjonalności:

- wyświetlanie modeli obiektów 3d o poprawnej kolejności wierzchołków w ścianach dla plików w formatach tekstowych (txt i obj) oraz binarnym (g3d),
- wyświetlanie modeli obiektów 3d o losowej kolejności ścian, po ewentualnej ich korekcie, dla plików w formatach tekstowych (txt i obj),
- logowanie do systemu,
- wylogowanie z systemu,
- zapisywanie skorygowanego modelu na komputerze klienta,
- dla zalogowanego użytkownika - możliwość zapisania obiektu na serwerze,
- dla administratora możliwość wyświetlania listy użytkowników oraz ich dodawania, edycji i usuwania.

Ponadto należy dodać także, że strona ma:

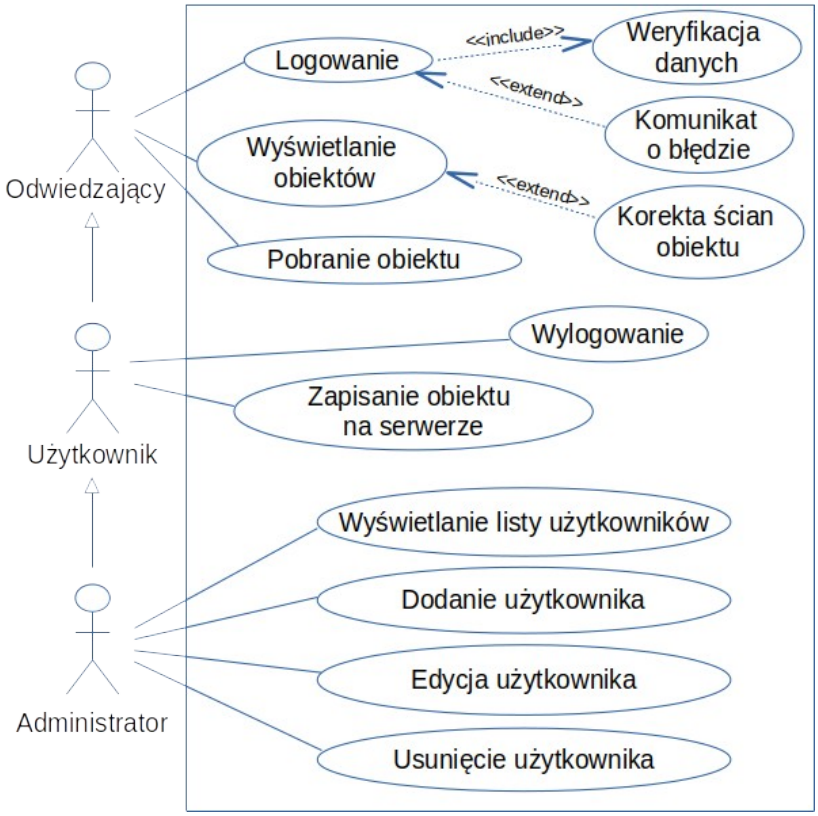
- obsługiwać graficzny interfejs użytkownika,
- pozwalać na dowolne obracanie, przybliżanie i oddalanie wyświetlanego modelu 3d,
- umożliwiać zmianę koloru wyświetlanego obiektu,
- umożliwiać wykonywanie przekrojów pionowych obiektów 3d.

Do **wymagań niefunkcjonalnych** należy zaliczyć:

- system od strony serwera ma być napisany w języku PHP,
- od strony klienta, system ma wykorzystywać język JavaScript wraz z biblioteką Three.js,
- system działa poprawnie w najpopularniejszych przeglądarkach internetowych,
- system jest dostępny 24h/365dni na całym świecie,
- system ma używać pliku tekstowego jako bazy danych użytkowników.

4.2. Diagram przypadków użycia.

Wymagania funkcjonalne wygodnie jest przedstawić w postaci diagramu przypadków użycia. Wraz z naniesionymi aktorami i przypadkami użycia pozwalają one na analizę systemu informatycznego zmniejszając ryzyko wystąpienia błędów implementacji. Diagram odpowiadający wymaganiom postawionym w rozdziale 4.1 został przedstawiony na rysunku 6.



Rys. 6. Diagram przypadków użycia systemu.

5. Projekty algorytmów.

Przed rozpoczęciem kodowania opracowano algorytmy rozwiązujące główne cele niniejszej pracy. Ich zasadnicze koncepcje zostały przedstawione wraz z ilustracjami oraz pseudokodami w kolejnych dwóch podrozdziałach.

5.1. Algorytm tworzący przekrój pionowy.

Utworzenie przekroju pionowego obiektu 3d sprowadzono do najprostszego możliwego przypadku, tj. przyjęto, że przekrój będzie utworzony przez przecięcie obiektu pojedynczą płaszczyzną prostopadłą do osi X. Przy założeniu poprawnej kolejności wierzchołków w modelu, procedura tworzenia przekroju jest następująca. Wczytujemy ściany i wierzchołki obiektu. Następnie obliczane są wektory normalne. Wszystkie te wielkości muszą być przechowane w pamięci komputera w celu późniejszego, selektywnego dodania ich do geometrii. Przez geometrię należy tutaj rozumieć obiekt Three.js przechowujący między innymi współrzędne wierzchołków, ściany i wektory normalne. Przy dodawaniu ścian i wierzchołków do geometrii, wielkości te nie są dodawane osobno, ale dodawane są odpowiednie ściany wraz z ich wierzchołkami i wektorem normalnym. Różnica w kodzie jest przedstawiona odpowiednio na listingu 1 i 2. Należy zauważyć, że w przypadku kodu przedstawionego jako drugi istnieje możliwość dodania lub pominięcia ściany, na przykład ze względu na wartość współrzędnych wierzchołków (Listing 2, linia 12). Dlatego w niniejszej pracy użyto analogicznego kodu.

W algorytmie otrzymujemy od użytkownika parametr x_{gr} będący pozycją "płaszczyzny obcięcia". Jest to wartość, gdzie należy wykonać przekrój. Wywołujemy zatem funkcję przedstawioną na listingu 2 i wykonujemy iteracje po wszystkich ścianach (trójkątach) obiektu. Sprawdzamy wartości współrzędnych x ich wierzchołków (linia 12). Przyjmijmy, że obcinamy obiekt powyżej zadanej wartości x_{gr}. Należy wówczas rozróżnić następujące sytuacje, które przedstawiono na rysunku 7. Pierwszą, gdzie wszystkie wierzchołki mają współrzędną $x \leq x_{gr}$ (rys. 7 - pole z numerem 1). Drugą, gdzie część wierzchołków ma współrzędne x położone przed „płaszczyzną” ($x \leq x_{gr}$), a część za ($x > x_{gr}$) (rys. 7 – pola nr 2 i 3). Trzecią (ostatnią), gdy wszystkie wierzchołki leżą za płaszczyzną obcięcia (rys. 7. - pole oznaczone numerem 4).

```
1    var material = new THREE.MeshLambertMaterial();
2    var geometry = new THREE.Geometry();
3    for(i=0; i<wierzcholki.length;i++){
4        geometry.vertices.push(new THREE.Vector3(
5            wierzcholki[i][0],
6            wierzcholki[i][1],
7            wierzcholki[i][2]
8        ));
9    }
10   for(i=0; i<sciany.length;i++){
11       var v1 = sciany[i][0]-1;
12       var v2 = sciany[i][1]-1;
13       var v3 = sciany[i][2]-1;
14       geometry.faces.push(new THREE.Face3(v1, v2, v3));
15   }
16   geometry.computeFaceNormals();
17   scene.add(new THREE.Mesh( geometry, material));
```

Listing 1. Załączanie wierzchołków i ścian do geometrii. Wszystkie wierzchołki i ściany są dodawane do geometrii.

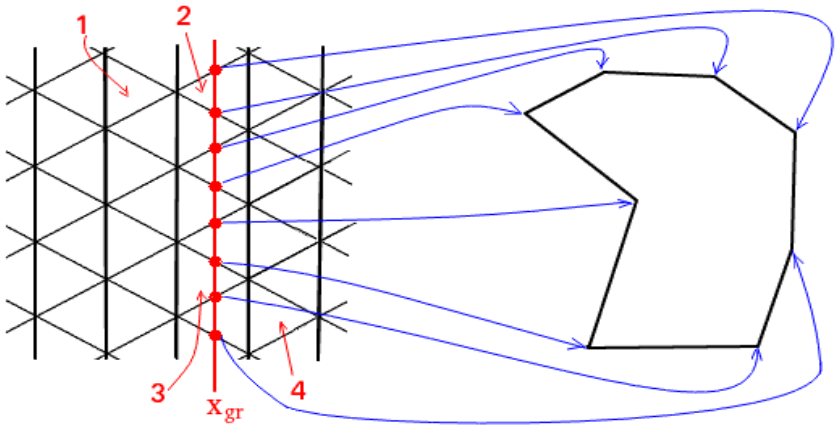
```
1    var material = new THREE.MeshLambertMaterial();
2    var calaGeometria = new THREE.Geometry();
3    for(var i=0;i<sciany.length;i++){
4
5        var n1 = sciany[i][0]-1;
6        var n2 = sciany[i][1]-1;
7        var n3 = sciany[i][2]-1;
8        var v1 = wierzcholki[n1];
9        var v2 = wierzcholki[n2];
10       var v3 = wierzcholki[n3];
11
12       if((v1[0]<=xgr)&&(v2[0]<=xgr)&&(v3[0]<=xgr)){
13           var geometry = new THREE.Geometry();
14           geometry.vertices.push(new THREE.Vector3(v1[0], v1[1], v1[2]));
15           geometry.vertices.push(new THREE.Vector3(v2[0], v2[1], v2[2]));
16           geometry.vertices.push(new THREE.Vector3(v3[0], v3[1], v3[2]));
17           var normal = new THREE.Vector3(wektN[0], wektN[1], wektN[2]);
18           geometry.faces.push(new THREE.Face3( 0, 1, 2, normal));
19           var obiekt = new THREE.Mesh( geometry, material );
20           calaGeometria.merge(obiekt.geometry, obiekt.matrix);
21       }
20   }
21   var calyObiekt = new THREE.Mesh( calaGeometria, material);
22   scene.add( calyObiekt);
```

Listing 2. Załączanie ścian do geometrii. W pętli for brane są kolejne ściany (`sciany[i]`). Na ich podstawie pobierane są wartości współrzędnych wierzchołków (`v1`, `v2`, `v3`). Jeśli współrzędne x wierzchołków są mniejsze bądź równe x_{gr} to ściana jest dodawana do geometrii (linia 12).

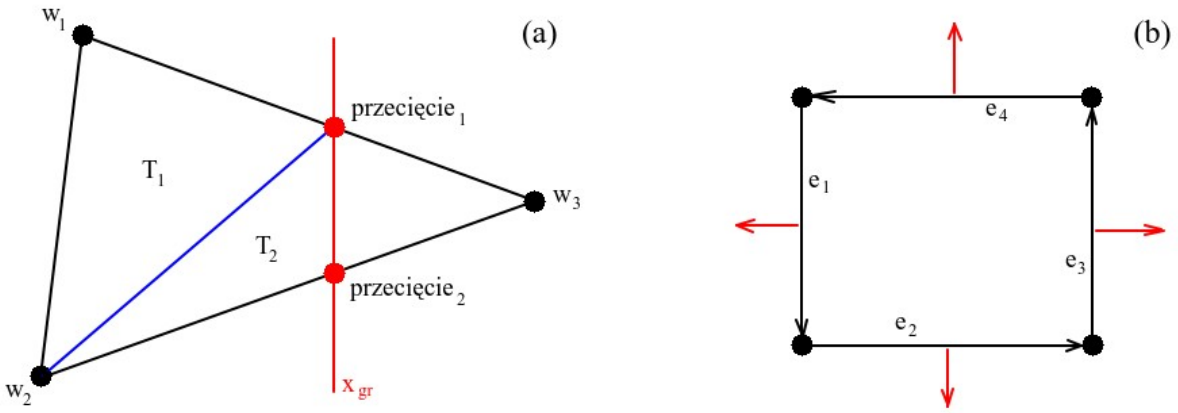
Procedura postępowania jest więc następująca:

1. Jeśli wszystkie wierzchołki ściany mają mniejszą lub równą wartość współrzędnej x od x_{gr}, to dodajemy ścianę do geometrii.
2. Jeśli dwa wierzchołki mają wartość $x > x_{gr}$, to obliczane są punkty przecięcia krawędzi trójkąta z płaszczyzną (x_{gr}) i zastępują one odpowiednie, oryginalne wierzchołki trójkąta. Pozostałe parametry ściany pozostają bez zmian. Tak zmodyfikowaną ścianę dodajemy do geometrii.
3. Jeśli jeden wierzchołek ma wartość x spełniającą $x > x_{gr}$, to obliczane są punkty przecięcia krawędzi ściany (trójkąta) z płaszczyzną. Następnie do geometrii dodawane są dwa trójkąty. Przykładowo, niech $x_3 > x_{gr}$, wówczas pierwszy trójkąt może zawierać dwa wierzchołki oryginalnego trójkąta oraz jeden z punktów przecięcia: T_1 (w_1, w_2, $przeciecie_1$), gdzie przyjęto, że $przecięcie_1$ to punkt przecięcia odcinka w_1w_3 z płaszczyzną obcięcia. Z kolei drugi trójkąt T_2 zawiera w takim przypadku wierzchołki w_2, $przeciecie_2$ i $przecięcie_1$. Sytuację przedstawiono na rysunku 8a.
4. Dla pozostałych przypadków, gdzie wszystkie wierzchołki mają wartość współrzędnych większą od x_{gr}, nie trzeba robić nic. Ściana nie jest dodawana do geometrii, tzn. jest ona w całości obcinana przez „płaszczyznę ścięcia”.

Dla przykładu, warunek pierwszy pokazano na Listingu 2. Pozostałe warunki testuje się analogicznie.



Rys. 7. Ściany obiektu są „odcinane” płaszczyzną odcięcia (reprezentowaną przez x_{gr}). Przedstawiono cztery rodzaje ścian. Pierwszy, gdy ściana jest dodawana do geometrii. Drugi, gdy ściana jest dodawana, przy czym zamienione zostają jej dwa wierzchołki. Trzeci, gdy ściana po odcięciu nie stanowi już trójkąta i musi być podzielona na dwa trójkąty. Czwarty rodzaj ścian jest poza x_{gr} i nie jest on dodawany do geometrii ani rysowany. Po stronie prawej narysowano figurę zamkniętą utworzoną przez punkty obiektu, którego przekrój jest robiony.



Rys. 8. a) Jedna z możliwych sytuacji w trakcie tworzenia przekroju trójkąta. Oryginalna ściana (trójkąt) musi być zastąpiony dwoma trójkątami T_1 i T_2. b) Uporządkowanie punktów odcinków e_1 – e_4 na podstawie zwrotu iloczynu wektorowego ich wektorów normalnych i różnicy ich współrzędnych. Wszystkie takie iloczyny odcinków przedstawionych na rysunku są zwrócone w kierunku „nad kartkę".

Punkt przecięcia krawędzi ściany $[x, y, z]$ z płaszczyzną znajdujemy dzięki równaniom prostej w przestrzeni [21]:

$$\begin{cases} x = x_0 + t(x_1 - x_0) \\ y = y_0 + t(y_1 - y_0) \\ z = z_0 + t(z_1 - z_0) \end{cases} \quad dla \quad t \in \mathbb{R}. \tag{1}$$

Przy czym należy pamiętać, że zachodzi $x = x_{gr}$. Należy zauważyć stosunkową prostotę rozszerzenia tego jednowymiarowego przykładu na dowolnie zadaną płaszczyznę. W celu późniejszego zamknięcia obiektu, którego przekrój tworzymy, dla każdego „przecinanego" trójkąta należy zapamiętać wyznaczone punkty przecięcia. Wygodnie jest to zrobić parami, wraz z wektorem normalnym obcinanego trójkąta w postaci obiektu. Trzeba zaznaczyć, że w przypadku równości obu współrzędnych przecięć (np. jeśli x_{gr} jest równe współrzędnej x wierzchołka w_3 na rys. 8) takiego obiektu nie należy pamiętać.

Po wykonaniu powyższej procedury dla wszystkich trójkątów, należy zamknąć przecięty obiekt 3d odpowiednim wielokątem utworzonym z zapamiętanych wcześniej punktów przecięć. Wyjątek od powyższego otrzymamy dla obiektów 3d, które nie stanowią obiektów zamkniętych, jak np. płaszczyzna. Takich obiektów nie należy zamykać. Wspomniane punkty, tworzące odcinki e (od ang. edge) powinny stanowić łamaną zwyczajną zamkniętą (rys. 7) lub ich skończony zbiór. Aby utworzyć takie łamane, punkty w odcinkach powinny być odpowiednio ułożone. Ponadto, same odcinki powinny zachowywać pewien porządek. W celu osiągnięcia uporządkowania wewnątrz odcinków można wykorzystać zwrot

iloczynu wektorowego „wektora normalnego odcinka” i różnicy jego współrzędnych. Wspomniany „wektor normalny odcinka jest równy wektorowi normalnemu zapamiętanemu podczas tworzenia przekroju odpowiedniego trójkąta. Zwrot wspomnianego iloczynu wektorowego musi być taki sam dla całego zbioru odcinków (rys. 8b). W przypadku niewłaściwego zwrotu iloczynu należy zamienić kolejność punktów będących końcami odcinka. Zapewnia to uporządkowanie punktów w wierzchołkach. Jeżeli założymy, że mamy tabelę odcinków o uporządkowanych wierzchołkach, to uporządkowanie kolejności odcinków można osiągnąć w sposób przedstawiony na Listingu 3 (listing funkcji `znajdzLamane(odcinki)`):

1. Tworzymy tabelę sąsiadów odcinków zawierającą indeks sąsiada danego odcinka (linie 4-10). Ponieważ początkowo tabela sąsiadów była wypełniona wartością „-1”, to brak sąsiada spowoduje pozostanie tej wartości w tabeli.
2. Jeśli `sąsiedziOdcinków` zawiera wartość „-1” (linia 12), to znaczy, że w obiekcie, którego przekrój jest robiony występują niedomknięte kształty (odcinki tworzą co najmniej jedną łamaną otwartą). Wówczas funkcja kończy działanie i zwraca `null` (obiekt nie jest zamykany).
3. W przeciwnym wypadku (linia 14) poczynając od odcinka, gdzie jeszcze nie byłeś (`zrobioneOdcinki.indexOf(0)`) (linia 19) idź do sąsiada, aż dojdziesz tam skąd zaczynałeś (`wskaznik!=indeks`). Zaznaczaj odcinki, które przeszedłeś i dodawaj je do k-tej łamanej.
4. Kontynuuj krok 3 tak długo póki są jeszcze odcinki, które nie zostały przypisane do łamanej (`while(zrobioneOdcinki zawiera 0)`) (linia 18).
5. Zwróć wszystkie znalezione łamane.

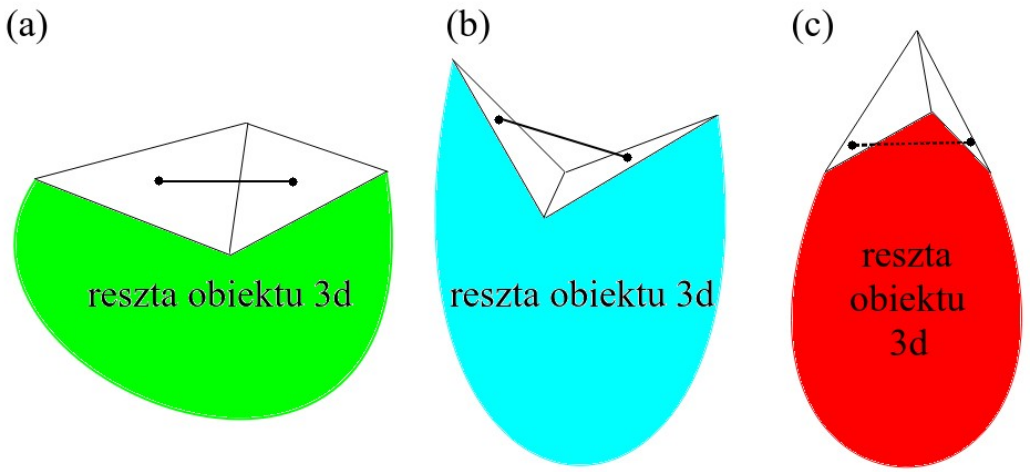
Zbiór łamanych należy triangulować, po czym powstałe trójkąty należy dodać do geometrii. Triangulacja jest co prawda opisana w literaturze, np. w [22] jednakże jest to proces nietrywialny. Na szczęście biblioteka Three.js umożliwia wyświetlenie wielokąta na podstawie zadanego zbioru uporządkowanych punktów [23]. Ponieważ przedstawiony wyżej algorytm ma na celu właśnie takie uporządkowanie należy jedynie utworzyć tablice zawierające `punkty[i] = new THREE.Vector2(punktLamanej).` Następnie należy dodać owe punkty do kształtu (`new THREE.Shape(punkty)`). Ów kształt trzeba jeszcze odpowiednio obrócić i przesunąć tak, aby leżał na płaszczyźnie ścięcia.

```
1     var lamane=[];
2     var sasiedziOdcinkow = new Array(odcinki.length).fill(-1);
3
4     for(var i=0; i<odcinki.length; i++){
5         for(var j=0; j<odcinki.length; j++){
6             if(odcinki[i].v2==odcinki[j].v1){
7                 sasiedziOdcinkow[i]=j;
8             }
9         }
10    }
11
12    if(sasiedziOdcinkow zawiara -1)){
13        return null;
14    } else {
15        var zrobioneOdcinki = new Array(odcinki.length).fill(0);
16
17        var k=0;
18        while(zrobioneOdcinki zawiera 0){
19            var indeks = zrobioneOdcinki.indexOf(0);
20
21            var doZrobienia =[];
22            var wskaznik = sasiedziOdcinkow[indeks];
23            doZrobienia.push(odcinki[indeks]);
24            zrobioneOdcinki[indeks]=1;
25
26            while(wskaznik!=indeks){
27                doZrobienia.push(odcinki[wskaznik]);
28                zrobioneOdcinki[wskaznik]=1;
29                wskaznik = sasiedziOdcinkow[wskaznik];
30            }
31            lamane[k]=doZrobienia;
32            k++;
33        }
34    }
35    return lamane;
```

Listing 3. Ciało funkcji `znajdzLamane(odcinki)`. Linie 4-10 wyznaczają sąsiadów odcinków. Linie 12-13 jeśli jakiś odcinek nie ma sąsiada to przerwij działanie funkcji zwracając `null`. Linie 18-33 są odpowiedzialne za wyszukanie łamanych zamkniętych wśród odcinków.

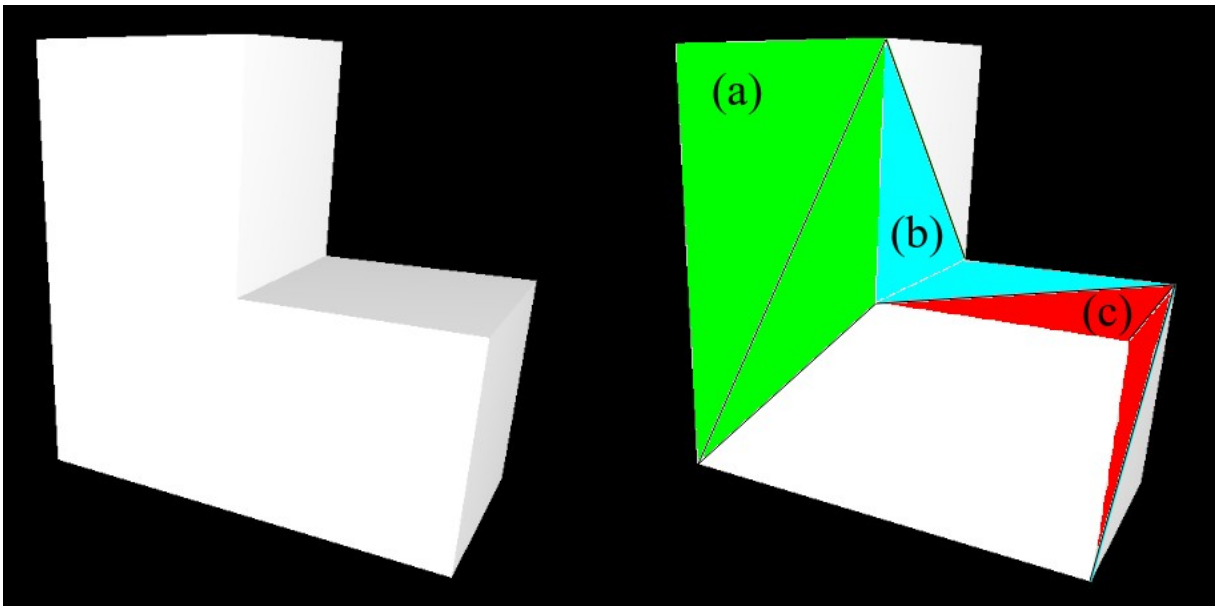
5.2. Algorytm poprawiający kolejność wierzchołków w ścianach.

W celu opracowania algorytmu poprawiającego orientację ścian obiektów należy analizować lokalne kształty obiektów 3d. Niech obiekt 3d składa się z płaskich ścian. Wówczas, jeśli weźmiemy dwie dowolne, sąsiadujące ze sobą ściany obiektu, to gdy połączymy odcinkiem owe ściany okazuje się, że punkty odcinka (za wyjątkiem punktów skrajnych) mogą leżeć jedynie na ścianach (rys. 9a), poza obiektem (rys. 9b) lub w środku obiektu (rys. 9c). Odpowiada to stwierdzeniu, że obiekt 3d może być jedynie lokalnie płaski, wklęsły lub wypukły.



Rys. 9. Obiekty 3d lokalnie (a) płaskie – odcinek łączący dwie ściany leży na tych ścianach, (b) wklęsłe – odcinek łączący ściany leży poza obiektem, (c) wypukłe – odcinek łączący ściany leży wewnątrz obiektu.

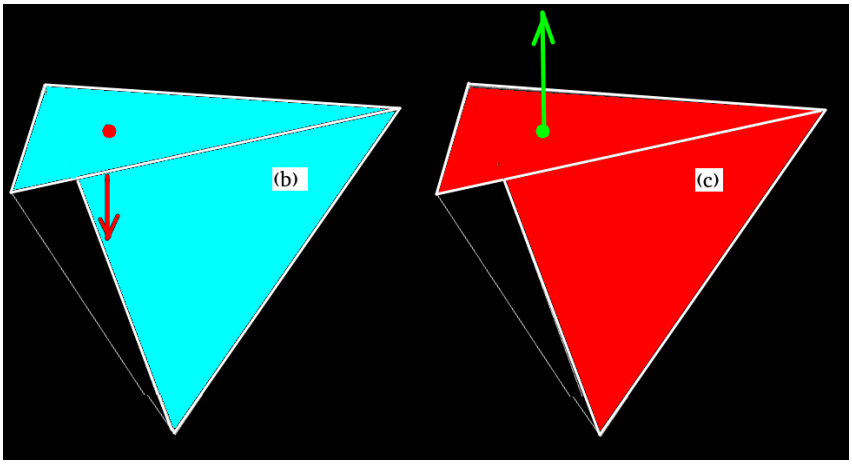
Lokalne kształty obiektów 3d odpowiadają więc geometrycznym relacjom pomiędzy ich pojedynczymi ścianami. Na rysunku 10 przedstawiono obiekt przypominający kształtem dużą literę L. Po stronie prawej, dla analizowanego obiektu oznaczono trzy pary sąsiadujących ścian. Ściany oznaczone kolorem zielonym oraz literą (a) leżą na tej samej płaszczyźnie i w związku z tym posiadają równoległe wektory normalne o takim samym zwrocie. Pozostałe oznaczone pary ścian nie leżą na tych samych płaszczyznach i zostaną przeanalizowane poniżej. Na podstawie lokalnych kształtów obiektów 3d, należy zauważyć, że wyróżnione pary odpowiadają jedynym trzem możliwym stosunkom, między sąsiadującymi ścianami. Innymi słowy, jeżeli pary ścian dla danego obiektu ze sobą sąsiadują, to jest możliwa tylko jedna z tych trzech konfiguracji.



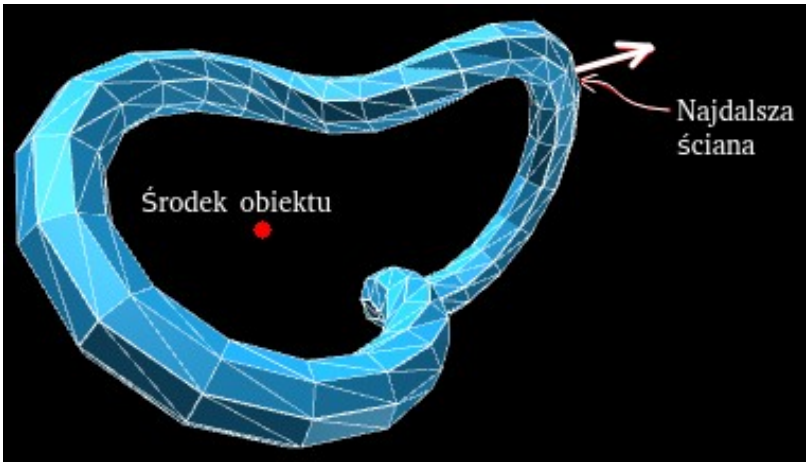
Rys. 10. Obiekt 3d w kształcie litery L. Po stronie prawej, na rysunku zostały oznaczone trzy możliwe wzajemne orientacje sąsiadujących ścian.

Pary (b) i (c) wraz z jednym wektorem normalnym na parę przedstawiono na rysunku 11. Przyjmijmy, że wektor umieszczony na rysunku jest wektorem znanym. Natomiast wyznaczyć należy wektor, którego nie umieszczono na rysunku. Trzeba zauważyć, że pary ścian zarówno (b) jak i (c) tworzą pewien hipotetyczny czworościan (stąd dorysowana krawędź na rys. 11). Zachodzi również zależność: jeżeli znany wektor normalny jest skierowany do wewnątrz czworościanu (przypadek b), to drugi – poszukiwany wektor normalny też musi mieć zwrot do środka. Jeżeli natomiast znany wektor jest skierowany na zewnątrz czworościanu (przypadek c), to drugi wektor również musi mieć zwrot na zewnątrz. Ponieważ wszystkie pary ścian spełniają jedną z trzech wspomnianych zależności, to jesteśmy w stanie wyznaczyć wszystkie wektory normalne dowolnego obiektu 3d przy znajomości jednego wektora normalnego obiektu.

W celu wyznaczenia pierwszego wektora normalnego należy zauważyć, że ściana obiektu będąca najdalej oddalona od jego środka ma przeważnie wektor skierowany w kierunku od środka na zewnątrz (rys. 12). To spostrzeżenie wraz z uwagami poczynionymi wyżej wystarczą do wyznaczenia wektorów normalnych (i kolejności wierzchołków w ścianach obiektu) dla większości obiektów 3d. Wspomniana odległość to odległość euklidesowa od środka obiektu do środka ściany. W przypadku, gdyby po wyznaczeniu wszystkich wektorów obiekt był wyświetlany nieprawidłowo należy odwrócić zwrot wszystkich wektorów.



Rys. 11. Czworościany tworzone przez dwie sąsiadujące ściany obiektu. Wektor normalny o zwrocie w kierunku środka czworościanu (przypadek b) implikuje, że czworościan jest „pusty” w środku i stąd drugi wektor normalny też musi być zwrócony do środka czworościanu. Wektor skierowany na zewnątrz (przypadek c) sugeruje, że czworościan jest „pełny” zatem drugi wektor normalny również musi być zwrócony na zewnątrz.



Rys. 12. Pewien obiekt 3d (TorusKnotGeometry z bibioteki Three.js) wraz z naniesionym hipotetycznym środkiem (czerwony punkt) oraz wektorem normalnym ściany najbardziej oddalonej od środka obiektu.

Pseudokod algorytmu poprawiającego kolejność wierzchołków w ścianach został przedstawiony na listingu 4. Przy czym funkcja `wyznaczPierwszyWektorNormalny()` sprowadza się do kodu umieszczonego w listingu 5.

```
1   wczytajObiekt();
2   wyznaczPierwszyWektorNormalny();
3   wyznaczReszteWektorow();
4   poprawSciany();
5   rysuj();
```

Listing 4. Pseudokod schematu poprawiania kolejności wierzchołków w ścianach.

```
1   zerujWektory();
2   wyznaczSrodekObiektu();
3   wyznaczSrodkiScian();
4
5   var najdalsza = wyznaczNajdalszaSciane(0);
6
7   var temp = sciany[0];
8   sciany[0] = sciany[najdalsza];
9   sciany[najdalsza] = temp;
10
11  temp = srodkiScian[0];
12  srodkiScian[0] = srodkiScian[najdalsza];
13  srodkiScian[najdalsza] = temp;
14
15  wektoryNormalne[0] = wyznaczWektorNormalny(0);
16  normujWektor(0);
17  wektoryNormalne[0] = sprawdzZwrotWektora(0);
```

Listing 5. Kod funkcji wyznaczającej pierwszy wektor normalny.

Funkcja `zerojWektory()` inicjuje tablicę wektoryNormalne zawierającą w każdej komórce następującą tabelę [0,0,0]. Taki wektor zerowy będzie stanowić informację dla funkcji `wyznaczReszteWektorow()` (Listing 6), że wektor ten nie został jeszcze wyznaczony. Funkcja `wyznaczSrodekObiektu()` wyznacza środek obiektu, którego ściany są analizowane. Następnie wyznaczane są środki ścian. `wyznaczNajdalszaSciane(var calkowita)` wyznacza numer ściany, której środek leży najdalej od środka obiektu (w sensie odległości euklidesowej) dla zbioru {`calkowita`, `calkowita+1`, …, `ilosc.scian`}. Kolejnym krokiem algorytmu jest zamienienie kolejności ścian i ich środków. Wykonywane jest to w celu oddzielenia ścian, dla których wyznaczono już wektory normalne od pozostałych. Dla najdalszej ściany (obecnie ściany o numerze 0) zostaje wyznaczony wektor normalny. Funkcja `normujWektor(var calkowita)` normalizuje wektor normalny o numerze równym `calkowita`, po czym `sprawdzZwrotWektora(var nr)` ustawia zwrot zadanego wektora przeciwnie do środka obiektu.

```
1   var wyznaczonychSasiadow = new Array(sciany.length).fill(0);
2   var i,j;
3   var iloscWyznaczonych = 1;
4
5   while(iloscWyznaczonych<sciany.length){
6       var flaga=0;
7
8       for(i=0; i<sciany.length; i++){
9           while(wyznaczonychSasiadow[i]>2){
10              i++;
11          }
12
13          if(!(wektoryNormalne[i]==[0,0,0])){
14              for(j=iloscWyznaczonych; j<sciany.length; j++){
15                  if(sciany[i] == sciany[j]){
16                  }else{
17
18                      if(dwaWierzcholkiTakieSame(sciany[i],sciany[j])){
19                          if(wektoryNormalne[j]==[0,0,0]){
20                              wyznaczDrugiWektor(i, j);
21                              zamienKolejnosc(j, iloscWyznaczonych);
22                              iloscWyznaczonych++;
23                              wyznaczonychSasiadow[i]++;
24                              flaga =1;
25                          }
26                      }
27                  }
28              }
29          }
30      }
31      if(flaga==0){
32
33          var najdalsza = wyznaczNajdalszaSciane(iloscWyznaczonych);
34          wektoryNormalne[najdalsza] = wyznaczWektorNormalny(najdalsza);
35          normujWektor(najdalsza);
36          wektoryNormalne[najdalsza] = sprawdzZwrotWektora(najdalsza);
37          zamienKolejnosc(najdalsza, iloscWyznaczonych);
38          iloscWyznaczonych++;
39      }
40  }
```

Listing 6. Kod funkcji `wyznaczReszteWektorow()`.

Funkcja `wyznaczReszteWektorow()` została przedstawiona na Listingu 6. Rozpoczyna się ona od deklaracji wypełnionej zerami tablicy `wyznaczonychSasiadow` oraz zmiennej `iloscWyznaczonych`, której wartość wynosi 1, ponieważ wyznaczono już pierwszy wektor normalny. Następnie, w linijce 5 rozpoczyna się pętla, która będzie wykonywana, aż wyznaczone zostaną wszystkie wektory normalne. W niniejszej pętli tworzona jest zmienna pomocnicza `flaga`, która na początku przyjmuje wartość 0. W linii 8 rozpoczynamy iteracje po wszystkich ścianach, przy czym jeśli dla danej ściany wyznaczono już wektory normalne wszystkich trzech sąsiadów, to ściana taka jest pomijana. Jeśli wektor normalny nie został jeszcze wyznaczony, to jest on równy wektorowi zerowemu ([0, 0, 0]). Dlatego w linii 13 widzimy, że jeśli i-ty wektor normalny jest niezerowy (został już wyznaczony), to iterujemy po ścianach z niewyznaczonymi wektorami normalnymi i szukamy ścian, które graniczą (mają dwa takie same wierzchołki) ze ścianą i. Jeśli owy j-ty wektor normalny takiego sąsiada jest zerowy (nie został jeszcze wyznaczony), to jest on wyznaczany na podstawie rozumowania przedstawionego na początku niniejszego podrozdziału. Następnie, zamieniana jest kolejność ścian, wektorów normalnych i środków ścian w taki sposób, że na początku tych tablic (od 0 do ilości wyznaczonych wektorów) są wielkości wyznaczone, a następnie – na końcu znajdują się wielkości niewyznaczone. Ilość wyznaczonych wektorów jest inkrementowana, dodawana jest informacja o wyznaczonym sąsiedzie ściany, po czym zmienna `flaga` jest ustawiana na 1. Gdyby spośród wszystkich niewyznaczonych ścian nie znaleziono żadnych sąsiadów to zmienna flaga pozostaje równa zeru. Sytuacja taka ma miejsce, jeśli występują rozłączne (bez wspólnych wierzchołków) części obiektu 3d. Wówczas, spośród ścian z niewyznaczonymi wektorami normalnymi wybieramy najdalej położoną od środka i wyznaczamy jej wektor normalny. Normujemy go, po czym ustawiamy jego zwrot na skierowany od środka obiektu na zewnątrz. Zmieniamy kolejność ścian, wektorów normalnych i środków. Zwiększamy `ilośćWyznaczonych` o 1 po czym iteracje zaczynają się od początku. Na podstawie wektorów normalnych poprawiana jest kolejność wierzchołków w ścianach – co jest celem algorytmu.

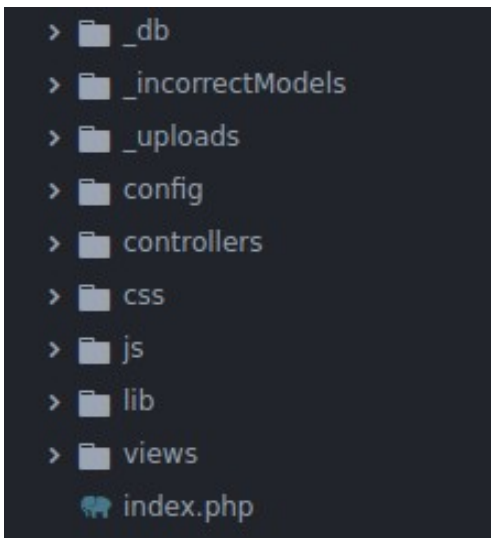
6. Struktura aplikacji.

Szkielet aplikacji został napisany w języku PHP. Jej struktura była inspirowana strukturą katalogów frameworku Laravel 4.2 [24]. Została ona przedstawiona na rysunku 13. Jednakże nie wykorzystano w niej silnika szablonowania, natomiast bazę danych np. MySQL zastąpiono plikiem tekstowym.

Kolejne katalogi zawierają:

- bazę danych zawierającą informację o użytkownikach systemu. Najważniejsze z nich, to: login, hasło i rola,
- obiekty 3d o losowej kolejności wierzchołków w ścianach (za wyjątkiem teaopot.obj),
- obiekty o poprawnej kolejności wierzchołków w ścianach,
- pliki konfiguracyjne aplikacji,
- skrypty PHP będące kontrolerami,
- arkusze stylów,
- skrypty JavaScript wraz ze skryptami osób trzecich (w katalogu lib),
- skrypty PHP osób trzecich,
- skrypty PHP będące widokami.

Plik index.php jest kontrolerem głównym.



Rys. 13. Struktura katalogów aplikacji.

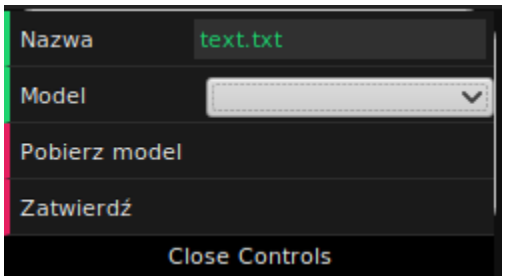
Zgodnie z umieszczonym wcześniej diagramem przypadków użycia (Rys. 6.), użytkownicy aplikacji zostali podzieleni na trzy grupy tj. Odwiedzających, Użytkowników oraz Administratorów. Grupy te mają różne uprawnienia, stąd aplikacja musiała zawierać system logowania. Nawigacja po stronie odbywa się za pomocą menu głównego (Rys. 14). Link „Aplikacja główna" odsyła użytkownika do modułu umożliwiającego tworzenie przekrojów pionowych obiektów 3d. „Korektor modeli" dotyczy modułu poprawiającego kolejność wierzchołków w ścianach obiektu. „Użytkownicy" - link dostępny tylko dla zalogowanego użytkownika posiadającego rolę „admin" prowadzi do listy użytkowników będących w bazie danych (i mogących się logować).

Rys. 14. Menu główne aplikacji (widok dla administratora). Aplikacja główna pozwala na tworzenie przekrojów pionowych obiektów 3d. Korektor modeli pozwala na korekcję wierzchołków w ścianach. Użytkownicy - opcja dostępna tylko dla administratora pozwala na wyświetlenie listy użytkowników w bazie danych.

Pomocnicze menu użytkownika zostało stworzone przy użyciu bibioteki dat.gui.js. Menu to jest różne w zależności od zakładki tj. dla aplikacji głównej oraz dla korektora modeli. W pierwszym przypadku wyświetla ono nazwę wybranego pliku, pozwala na zmianę wyświetlanego modelu, na włączenie i wyłączenie wyświetlania części obiektu, zmianę koloru obiektu oraz na tworzenie przekroju. Należy pamiętać, że możliwa zmiana modelu dotyczy plików będących w katalogu „_uploads" oraz że modele zawarte w tym katalogu mają poprawną kolejność wierzchołków w ścianach. Przykładowy widok menu dla aplikacji głównej przedstawiono na rysunku 15. Widok ten różni się dla różnych modeli, gdyż dla każdego podmodelu dynamicznie tworzone są „katalogi" menu zawierające opcje pokaż i kolor. W przypadku obiektu teapot.obj istnieje tylko jeden podmodel.

Rys. 15. Menu pomocnicze aplikacji głównej.

W przypadku korektora, menu pomocnicze wyświetla nazwę wybranego pliku, pozwala na zmianę wyświetlanego modelu i pobranie poprawionego modelu na komputer klienta. W przypadku zalogowanego użytkownika również na wysłanie modelu na serwer – do katalogu „_uploads” (opcja zatwierdź). Ze względów bezpieczeństwa [25,26] nie pozwolono użytkownikom wczytywać plików z ich komputera. Zamiast tego, utworzony został katalog zawierający modele o losowej kolejności wierzchołków w ścianach („_incorrectModels”). Użytkownicy mogą dokonywać korekty ścian dla modeli będących w tym katalogu. Aplikacja została umieszczona i przetestowana na serwerze http://155.158.112.56/~PracaInzAN/.



Rys. 16. Menu pomocnicze dla korektora modeli.

Jak wspomniano wcześniej, w aplikacji wykorzystano koncepcję kontrolera głównego przedstawionego na wykładach dotyczących programowania stron internetowych autorstwa dra Przemysława Kudłacika [27]. Najważniejsze części jego ciała w postaci pseudokodu przedstawiono na listingu 7. Składa się ono z dwóch switch-y (linia 2 oraz 22). Pierwszego publicznego oraz drugiego, którego akcje są dostępne po zalogowaniu. Sprawdzanie zalogowania jest dokonywane w pliku „check.php” (linia 19). Działanie switch-y jest następujące: na podstawie akcji podanej w adresie URL wybierany jest case, którego kod będzie wykonywany. W kolejnych case-ach tworzony jest odpowiedni obiekt, po czym następuje wywołanie jego metod. W switch-u publicznym należy na końcu każdego case-u zatrzymać dalsze przetwarzanie skryptu komendą `exit()`, aby nie uruchomić check.php. Ponadto, nie może on mieć akcji domyślnej `default`.

```
1     //-------------- akcje publiczne --------------
2     switch ($action){
3         case '':
4             include_once getConf()->root_path.'/ModelCtrl.php';
5             $ctrl = new ModelCtrl();
6             $ctrl->generujIndeks();
7             exit();
8         break;
9         case 'check':
10            include_once getConf()->root_path.'/LoginCtrl.php';
11            $ctrl = new LoginCtrl();
12            $ctrl->doLogin();
13            exit();
14        break;
15        case ...
16    }
17
18    //-------------- sprawdzenie zalogowania --------------
19    include getConf()->root_path.'/check.php';
20
21    //-------------- akcje dostępne po zalogowaniu --------------
22    switch ($action) {
23        case 'logout' :
24            include_once getConf()→root_path.'/LoginCtrl.php';
25            $ctrl = new LoginCtrl();
26            $ctrl→doLogout();
27        break;
28        case …
29    }
```

Listing 7. Pseudokod kontrolera głównego.

7. Implementacja algorytmów i rezultaty.

Przed wykonaniem jakichkolwiek operacji na modelach 3d musi nastąpić ich wczytanie do pamięci komputera. W przypadku plików tekstowych, w celu odczytu plików będących na serwerze wykorzystywany jest obiekt `XMLHttpRequest`. W rezultacie otrzymywana jest linia tekstu, która jest dzielona na poszczególne linijki danych, z których odczytywane są wartości współrzędnych wierzchołków, numery wierzchołków w ścianach i opcjonalnie wektory normalne. Na podstawie tych informacji tworzony jest obiekt 3d. Odczyt plików binarnych g3d wykorzystywanych w geologii jest ściśle związany z ich strukturą. Dlatego w niniejszej pracy, w tym celu skorzystano z gotowego skryptu (g3dReader.js autorstwa dra Krzystofa Wróbla). Skrypt ten tworzy tablicę takich samych modeli, jak w przypadku plików tekstowych.

Kolejną operacją wykonywaną dla wszystkich modeli jest normalizacja współrzędnych wierzchołków. Stanowi ona istotne udogodnienie dla użytkownika, gdyż dzięki niej wyświetlany obiekt jest zawsze porównywalnych rozmiarów. Normalizacja była wykonana za pomocą wyrażeń analogicznych do tych prezentowanych w pracy [28]:

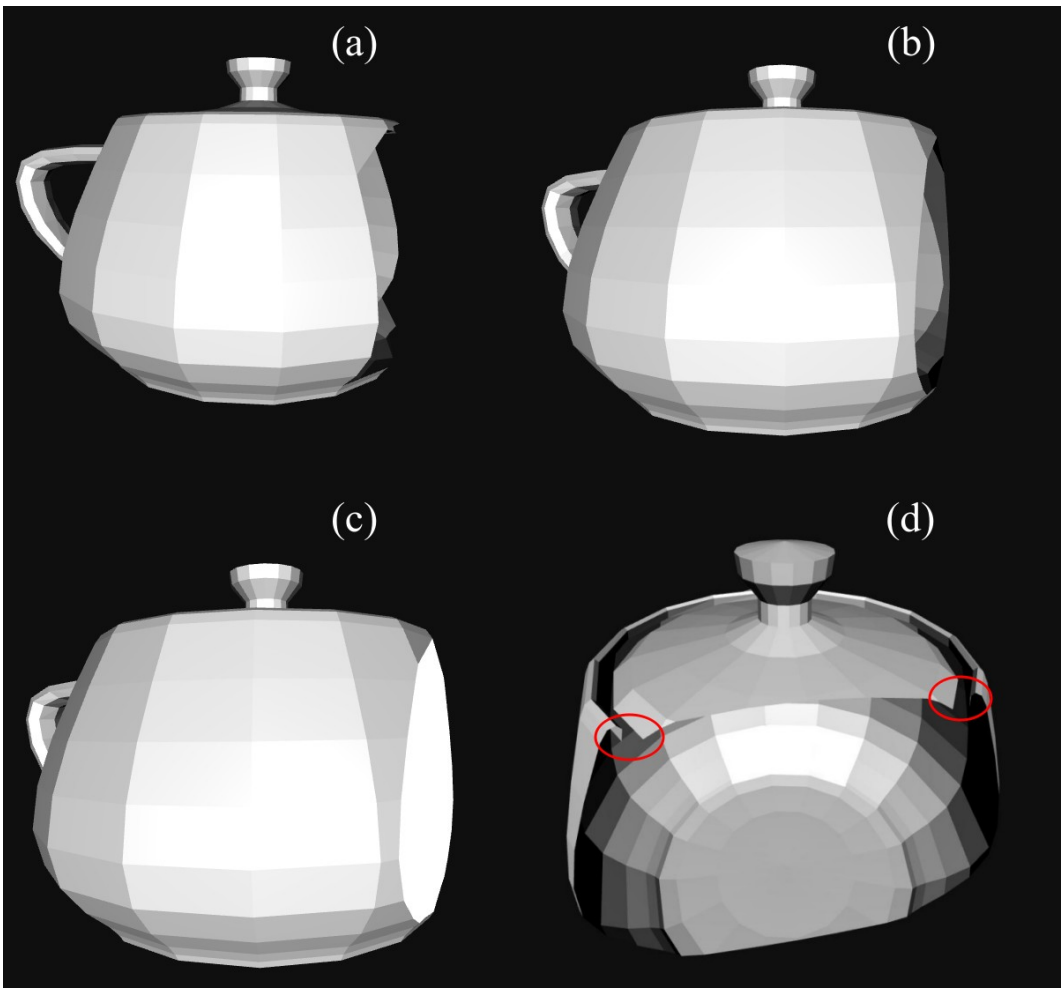
$$\boldsymbol{V}_{nowy} = \frac{2 \cdot proporcje \cdot (\boldsymbol{V} - środek)}{max - min} \tag{2}$$

gdzie **V** to oryginalne współrzędne wierzchołka, *max* i *min* to odpowiednio maksymalne współrzędne wierzchołków obiektu w danym kierunku. *środek* jest wyznaczany przez policzenie średniej między *max* i *min*. Natomiast *proporcje* wyznaczane są przez policzenie różnic *max-min* dla kierunku X, Y i Z a następnie wzięcie maksymalnej z nich i podzielenie przez nią wszystkich tych różnic. Pozwala to zachować proporcje obiektu. Po normalizacji cały obiekt ma współrzędne, we wszystkich kierunkach zawarte w przedziale od -1 do 1.

7.1. Generowanie przekrojów pionowych obiektów 3d.

Po poprawnym wczytaniu obiektu oraz unormowaniu współrzędnych wierzchołków implementowano kod wczytujący poszczególne ściany do geometrii (załączony na listingu 2). Przedstawione procedury zostały poszerzone o warunek logiczny: jeśli współrzędne *x*

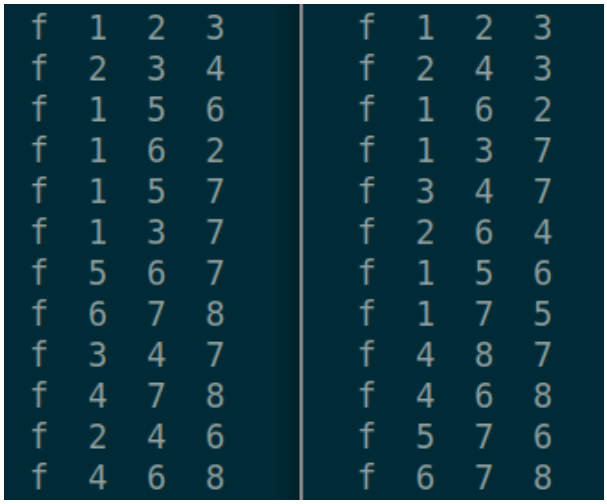
wszystkich wierzchołków są mniejsze od zadanej wartości granicznej, to dany trójkąt zostaje załączony do geometrii (i w rezultacie wyświetlony). Rezultaty przedstawiono na rysunku 17a. Wyraźnie widoczne są „postrzępione" ściany czajnika. Następnie po uwzględnieniu w kodzie skryptu instrukcji warunkowych dodających do geometrii trójkąty powstałe z przecinanych, oryginalnych ścian obiektu osiągnięto efekt przedstawiony na rys. 17b. Widoczna jest gładka krawędź obiektu w miejscu przekroju oraz wnętrze obiektu. Następnym krokiem implementacji algorytmów było dodanie kształtu (`THREE.Shape`) zamykającego przecięty obiekt (rys. 17c). Widoczny jest kształt zamykający przekrój. Jak zaznaczono wcześniej, w przypadku niezamkniętych konturów program nie dodaje kształtu zamykającego obiekt (rys. 17d). Przerwy między pokrywką a resztą czajnika zaznaczono na rysunku.



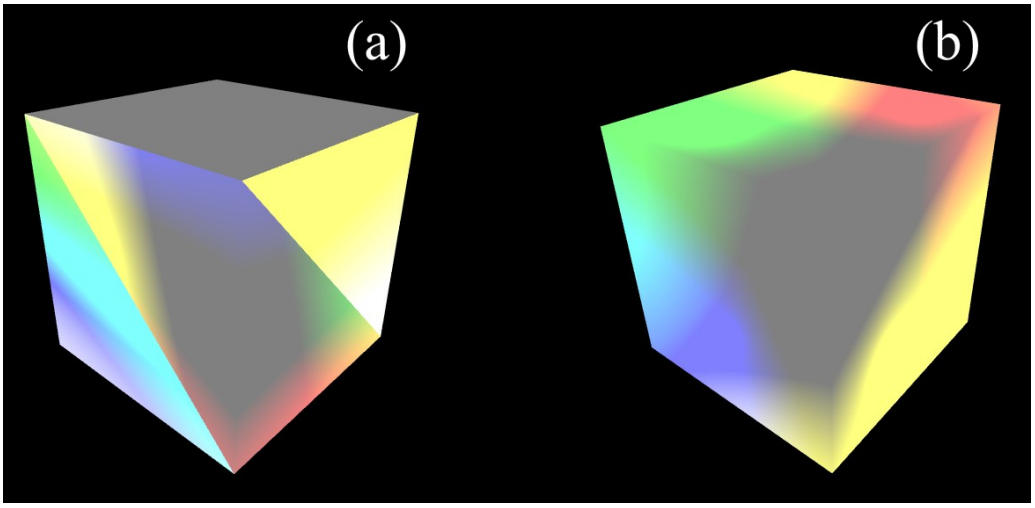
Rys. 17. Kolejne etapy tworzenia przekroju pionowego obiektu „teapot.obj". (a) Przekrój po dodaniu tylko ścian o współrzędnych $x \leq x_{gr}$. (b) Przekrój po dodaniu ścian przecinanych płaszczyzną przekroju. (c) Przekrój jest zamykany. (d) Przecięty czajnik bez zamkniętego konturu nie jest zamykany. Przerwy zaznaczono czerwonymi elipsami.

7.2. Normalizacja wektorów normalnych obiektów.

W celu przetestowania implementacji algorytmu poprawiającego kolejność wierzchołków stworzono kilka modeli obiektów 3d o losowej kolejności wierzchołków w ścianach. Ściany jednego z nich (sześcian.txt) zostały przedstawione po stronie lewej rysunku 18. Cyfry stanowią numery wierzchołków należących do odpowiednich ścian. Po stronie prawej umieszczono wierzchołki po dokonaniu korekty przy użyciu implementacji algorytmu. Zmiana kolejności ścian obiektu jest bez znaczenia dla rezultatów wyświetlania obiektu, natomiast wierzchołki obiektu nie uległy zmianie podczas korekcji. W związku z powyższym należy zauważyć, że ściany o numerach wierzchołków 2 3 4, 1 5 7, 5 6 7, 4 7 8 i 2 4 6 zostały zmienione odpowiednio przez 2 4 3, 1 7 5, 5 7 6, 4 8 7, oraz 2 6 4. Rezultaty niniejszej korekty przedstawiono na rysunku 19. Na rysunku (a) widoczny jest sześcian bez poprawiania kolejności wierzchołków w ścianach. Góra sześcianu jest czarna z powodu niewłaściwej kolejności wierzchołków i w rezultacie zwrotu wektorów normalnych. Ponadto obie pokazane ściany boczne mają niewłaściwe kolory. Na rysunku 19b kolejność wierzchołków w ścianach została skorygowana. Sześcian jest wyświetlany poprawnie.



Rys. 18. Numery wierzchołków występujące w ścianach sześcianu. Kolejność ścian nie ma wpływu na wyświetlanie obiektu. Po lewej stronie przed korektą, po stronie prawej po korekcie. Po bliższym przyjrzeniu można zauważyć, że ściany 2 3 4, 1 5 7, 5 6 7, 4 7 8 i 2 4 6 zostały zmienione podczas korekty.



Rys. 19. Obiekt sześcian.txt dla modelu oświetlenia Phonga przy użyciu czterech świateł. (a) Sześcian został wczytany bez korekty kolejności wierzchołków w ścianach. Widoczne są zarówno ściany z wektorami normalnymi zwróconymi w przeciwnym kierunku niż padające światło, jak i błędne kolory ścian. (b) Po korekcie obiekt jest wyświetlany poprawnie.

8. Podsumowanie.

W ramach rozprawy zaproponowano oraz zaimplementowano algorytmy rozwiązujące następujące problemy:

- tworzenie poprawnie wyświetlanych przekrojów pionowych obiektów 3d,
- korektę wektorów normalnych oraz kolejności wierzchołków w ścianach obiektów 3d.

Mimo osiągnięcia dwóch powyższych celów, oba z powyższych tematów ciągle pozostają otwarte na ulepszenia. Przykładowo szybkość działania algorytmu tworzącego przekroje pionowe pozostawia wiele do życzenia.

W celu przyśpieszenia algorytmu można m.in.:

- zastosować geometrię buforową,
- dla wielu obiektów wczytywanych na scenę przechowywać ich wartości maksymalne i minimalne w celu sprawdzenia czy płaszczyzna przecina obiekt,
- sortować ściany ze względu na wartości ich wierzchołków.

Zaletą algorytmu tworzącego przekroje jest stosunkowa łatwość rozszerzenia go na dowolne płaszczyzny. Wystarczy w warunku wspomnianym przy Listingu 2 sprawdzać, po której stronie płaszczyzny leżą punkty. Może to jednak jeszcze bardziej obniżyć wydajność algorytmu. Problematyczne wydaje się również dodawanie większej ilości płaszczyzn tworzących przekroje. Ponadto w algorytmie tworzenia przekrojów pionowych nie uwzględniono możliwości występowania dziur w obiekcie. Wreszcie ze względu na osobne dodawanie poszczególnych ścian do geometrii, nie da się wykorzystać automatycznego wyznaczania wektorów normalnych do wierzchołków. Wszystkie te problemy mogą prowadzić do konieczności modyfikacji algorytmu w celu jego dalszego rozwijania.

Algorytm korygujący kolejność wierzchołków w ścianach należy poszerzyć o kierowane do użytkownika zapytanie o poprawność wyświetlania podobiektu. W przypadku odpowiedzi negatywnej, należy odwrócić wektory oraz zmienić kolejność wierzchołków.

Literatura.

[1] J. Ganczarski, *OpenGL, Podstawy programowania grafiki 3D*, Helion, 2015.

[2] T. Komura,
Computer Graphics, Lecture 5. Hidden Surface Removal and Rasterization,
[Online] [Dostęp: 27.04.2019]. https://slideplayer.com/slide/9854716/

[3] R. Cabello, *Three.js clipping examples*, [Online] [Dostęp: 27.04.2019].
https://threejs.org/examples/webgl_clipping.html

[4] M. Ciosk, *Projektowanie webowych aplikacji graficznych w technologii WebGL*,
Uniwersytet Śląski w Katowicach, Wydział Informatyki i Nauki o Materiałach,
Zakład Systemów Komputerowych, Sosnowiec, 2018.

[5] J. Grant,
3D Graphics for Game Programming. Chapter I - Modeling in Game Production,
[Online] [Dostęp 24.03.2019]. https://slideplayer.com/slide/8318377/

[6] M. Gajer, *Podręcznik podstaw programowania w języku VRML*, AGH, Kraków, 2012.

[7] Web3D Consortium, *What is X3D?*, [Online][Dostęp: 26.03.2019].
http://www.web3d.org/x3d/what-x3d

[8] IT History Society, *FutureSplash Animator*, [Online][Dostęp: 26.03.2019].
https://www.ithistory.org/db/software/futurewave-software/futuresplash-animator

[9] A. Szatkowska, M. Szatkowski,
Projektowanie grafiki trójwymiarowej w środowisku Flash,
Śląska Wyższa Szkoła Informatyczno-Medyczna, Wydział Informatyki,
Chorzów, 2010.

[10] Adobe Systems, *Poznajemy język ActionScript 3.0*, [Online][Dostęp: 26.03.2019]
https://help.adobe.com/pl_PL/as3/learn/as3_learning.pdf

[11] S. Ryfka, *Implementacja algorytmu Ray Casting w środowisku Flash,*
Śląska Wyższa Szkoła Informatyczno-Medyczna, Wydział Informatyki,
Chorzów, 2010.

[12] Mozilla Developer Center, *HTML5*, [Online][Dostęp: 26.03.2019].
https://developer.mozilla.org/pl/docs/HTML/HTML5

[13] T. Parisi, *Aplikacje 3D. Przewodnik po HTML5, WebGL i CSS3*, Helion, 2015.

[14] W3Schools, *JavaScript Tutorial*, [Online][Dostęp: 27.03.2019]. https://www.w3schools.com/js/

[15] A. Chmielewski, *JavaScript*, Politechnika Białostocka, Wydział Informatyki, 2011. [Online][Dostęp: 27.03.2019]. http://aragorn.pb.bialystok.pl/~achmielewski/wd/ti4.pdf

[16] S. Malik, K. Przewoźnik, *JavaScript*, Encyklopedia Zarządzania [Online][Dostęp: 27.03.2019]. https://mfiles.pl/pl/index.php/JavaScript

[17] Khronos Group, *WebGL - OpenGL ES for the Web*, [Online][Dostęp: 27.03.2019]. https://www.khronos.org/webgl/

[18] Mozilla Developer Center, *WebGL*, [Online][Dostęp: 27.03.2019]. https://developer.mozilla.org/pl/docs/Web/API/WebGL_API

[19] W. Brown, *1.3 - Computer Graphics - A Brief History – LearnWebGL*, [Online][Dostęp: 27.03.2019]. http://learnwebgl.brown37.net/the_big_picture/webgl_history.html

[20] R. Cabello, *three.js - Javascript 3D library*, [Online][Dostęp: 27.03.2019]. https://threejs.org/

[21] M. Grzesiak, *Płaszczyzna i prosta w przestrzeni*, Politechnika Poznańska, Instytut Matematyki, Materiały dla Studentów – Geometria, [Online][Dostęp: 27.03.2019]. http://maciej.grzesiak.pracownik.put.poznan.pl/W-GEO-plaszczyzna-prosta.pdf

[22] J. O'Rourke, *Computational Geometry in C (Second Edition)*, Cambridge University Press, 1998. [Online][Dostęp: 27.03.2019]. http://cs.smith.edu/~jorourke/books/compgeom.html

[23] L. Blue, et. al, *Simple procedurally – generated shapes*, [Online][Dostęp: 27.03.2019]. https://threejs.org/examples/#webgl_geometry_shapes

[24] kernel-panic, Laravel 4.2 directory structure, [Online] [Dostęp 18.04.2019]. http://laravelhowto.blogspot.com/2017/03/laravel-42-directory-structure.html

[25] S. Hanselman, *Back to Basics: When allowing user uploads, don't allow uploads to execute code*, [Online] [Dostęp 18.04.2019]. https://www.hanselman.com/blog/BackToBasicsWhenAllowingUserUploadsDontAllowUploadsToExecuteCode.aspx

[26] I. Data,
*PHP: Running *.jpg as *.php or How to Prevent Execution of User Uploaded Files*,
[Online] [Dostęp 18.04.2019].
https://medium.com/@igordata/php-running-jpg-as-php-or-how-to-prevent-execution-of-user-uploaded-files-6ff021897389

[27] P. Kudłacik,
Języki Programowania Stron Internetowych (cz.1) [PHP] - 6. Kontroler główny,
[Online] [Dostęp 18.04.2019], http://kudlacik.eu/page/prz_jpsi.projects

[28] K. Wróbel, et. al,
Three dimensional image projections and its measurement using the vrml technique,
Journal of medical informatics & technologies 11 (2007),
[Online] [Dostęp 18.04.2019],
https://www.researchgate.net/publication/242599258_THREE_DIMENSIONAL_IMAGE_PROJECTIONS_AND_ITS_MEASUREMENT_USING_THE_VRML_TECHNIQUE

Spis listingów.

Spis rysunków.

Spis użytych skryptów autorstwa osób trzecich.

1 three.js ……Ricardo Cabello
2 dat.gui.min.js …… Jono Brandel, et. al
3 TrackballControls.js ……. Eberhard Graether, et. al
4 binaryReader.js …….. Jonas Raoni Soares da Silva
5 FileSaver.js …….Eli Grey
6 g3dReader.js ……... dr Krzysztof Wróbel
7 helper_functions.php, Messages.class.php …….. dr Przemysław Kudłacik