AKADEMIA GÓRNICZO-HUTNICZA
im. St. Staszica w Krakowie
EAIiIB, Katedra Automatyki
Laboratorium Biocybernetyki

Przedmiot:	**Wieloprocesorowe Systemy Wizyjne.**			
			Pr13wsw509	
Temat projektu:				
	Interaktywna wizualizacja sceny 3D w przeglądarce internetowej			
Wykonali:	Ziemowit	Rachwał	ziemoracho@gmail.com	
	Oskar	Kuligowski	okuligowski@gmail.com	
Stopień, rok i kier.	Magisterskie	Rok I	Automatyka i Robotyka	
Rok akademicki	2013/2014	Semestr zimowy		
Prowadzący	dr inż.	Mirosław	Jabłoński	

wersja 1.0

Kraków, styczeń, 2014.

Spis treści:

1. Abstrakt

Celem projektu było stworzenie strony internetowej z zaimplementowaną wizualizacją obrazu z kamery rozszerzonego o interaktywne, wirtualne elementy. Zgodnie z założeniami użyto technologii HTML 5 [1], języka JavaScript [2] i biblioteki pozwalającej na wizualizację scen 3D – WebGL [3].

Gotowy program pozwala na wskazanie zdjęcia z kamery oraz zdefiniowania pewnych parametrów z nim związanych, następnie wprowadzenia wielu, opisanych w dalszych rozdziałach, ustawień i ostatecznie przeprowadzenia wizualizacji.

Słowa kluczowe: WebGL, HTML 5, JavaScript, rzeczywistość rozszerzona, kamera, scena 3D, pixel shader, GLSL.

2. Wstęp

Założeniem pracy było stworzenie systemu wizualizacji obrazu uzupełnionego o wirtualne dane, zwanego również rzeczywistością rozszerzoną. Tego rodzaju systemy stają się coraz bardziej popularne, gdyż pozwalają przedstawiać istotne informacje w najbardziej dla człowieka przyswajalny sposób, a bardzo szybko rozwijająca się technika zwiększa możliwości ich zastosowania. Bardzo dobrym przykładem systemu przedstawionego powyżej jest Google Glass.

Aby odpowiednio podejść do problemu należało zapoznać się ze specyfikacją zarówno technologii HTML 5, języka JavaScript, jak i API biblioteki WebGL (Web Graphics Library). Za pomocą tych narzędzi możliwe było zbudowanie strony internetowej oraz aplikacji wykonującej obliczenia niezbędne to wykonania wizualizacji. Zarówno HTML 5 i WebGL są technologiami szybko się rozwijającymi, posiadającymi pełne wsparcie ze strony twórców. Wszystkie największe przeglądarki gwarantują ich obsługiwanie. To czyni projekt łatwym w uruchomieniu na różnych środowiskach, również mobilnych. Innym atutem technologii WebGL jest wsparcie ze strony karty graficznej. Możliwe jest wykorzystanie jej do wykonywania obliczeń, których wykonanie na procesorze zajęłoby więcej czasu.

W ostatecznej formie projektu zawarto panel operatorski stworzony w języku HTML 5, przy pomocy którego możliwe jest badanie poprawności kalibracji kamery. Panel ten umożliwia wczytanie zdjęcia z wykonanego konkretną kamerą oraz skonfigurowanie jego parametrów. Następnie, wprowadzenie danych dotyczących punktu orientacji w przestrzeni trójwymiarowej oraz zdefiniowanie punktów charakterystycznych obiektów będących fragmentami sceny. Druga część projektu, czyli logika pisana w języku JavaScript, po analizie wprowadzonych danych w każdej klatce rysuje na zdefiniowanym fragmencie ekranu kolejne klatki animacji. Do tego używane są tak zwane pixel shadery, czyli programy pisane w języku GLSL, opisane w rozdziale 3.

Jednym z założeń projektu było uwzględnienie strumienia wideo wysyłanego z urządzenia zdalnego jako podkład do wizualizacji sceny, zamiast zdjęć. Nie zostało to jednak zaimplementowane w ostatecznej wersji projektu.

3. Analiza złożoności i estymacja zapotrzebowania na zasoby

Analiza zapotrzebowania na zasoby sprowadza się do uwzględnienia poniższych aspektów problemu:

a) Złożoność związana z istotą problemu

W przypadku przetwarzania grafik 3D każdy obiekt ukazany na ekranie zbudowany jest w rzeczywistości z trójkątów. Natomiast każdy trójkąt posiada 3 punkty, z których każdy posiada 3 współrzędne. Rzeczywiste wyliczenie kolorów pikseli na ekranie sprowadza się do wyliczenia współrzędnych wierzchołków wszystkich obiektów, a następnie obliczenie koloru zawartości pomiędzy tymi wierzchołkami – faktycznej treści figury. Do tego celu służą programy pisane w osobnym języku, GLSL (OpenGL Shading Language). Programy te dzielą się na 2 typy i są to:

- Vertex Shader – program liczący położenie wierzchołków na ekranie
- Fragment Shader – program liczący kolor poszczególnych pikseli znajdujących się pomiędzy wierzchołkami

Złożoność obliczeniowa operacji przeprowadzanych w tych programach wzrasta wraz z każdym dodanym wierzchołkiem liniowo. W związku z tym, problem sprowadza się do ilości wierzchołków na scenie. W przypadku opisywanego projektu i rysowanych obiektów, nie jest ona na tyle duża, by nawet na stosunkowo słabym sprzęcie oko człowieka było w stanie zauważyć różnicę.

b) Zastosowanie algorytmów zwiększających wydajność

Możliwe jest zastosowanie technik zwiększających wydajność samego algorytmu liczenia położenia wierzchołków i kolorów pikseli. Jednym z takich algorytmów jest indeksacja wierzchołków. Algorytm ten został zastosowany, a jego opis znajduje się w dalszych rozdziałach.

c) Wycieki pamięci i złożoność obliczeniowa związana z technikami programistycznymi

Programowanie w języku JavaScript jest dla osób będących zwolennikami programowania obiektowego problemowe, gdyż sam język jest skryptowy. Z tego faktu, jak i z ewentualnych braków umiejętności programistycznych, mogą wynikać nie tylko wycieki pamięci ale także złe rozwiązania powodujące osłabienie wydajności algorytmu. Jednakże z powodu już wymienionego wcześniej (dość mała komplikacja problemu) błędy te, jeżeli występują, najprawdopodobniej nie wpływają na wydajność w sposób znaczący.

4. Koncepcja proponowanego rozwiązania

Rozwiązanie, jak i działanie aplikacji, zostało podzielone na następujące elementy:

a) Wygenerowanie struktury HTML

Na tym etapie tworzone są elementy strony internetowej będącej panelem operatorskim użytkownika aplikacji. Elementy te są statycznie zdefiniowane w głównym pliku aplikacji (index.html) zgodnie ze standardem języka HTML5. Główne panel zawiera canvas („podkład" który używany jest przez bibliotekę WebGL do generowania grafiki), wszystkie pola i przyciski dla użytkownika.

Osoba operująca najpierw wprowadza obrazek i jego ustawienia, następnie definiuje parametry kamery (parametry związane z ogniskową, wartości środkowego piksela, zakres widzialności), parametry puntu odniesienia (umiejscowienie kamery i jej rotacja), a następnie określa rozmiar i położenie wirtualnego prostopadłościanu oraz rozmiar punktów wierzchołkowych.

b) Wstępne stworzenie modelu 3D

Po zaakceptowaniu wprowadzonych danych, zdefiniowanych w poprzednim podpunkcie, tworzony zostaje wirtualny model 3D. Parametry obiektów (położenie wierzchołków ścian, rotacja i translacja kamery oraz inne parametry kamery) zostają przekazane do głównego obiektu przechowującego dane na temat sceny. W tym konkretnym przypadku słowo „obiekt" jest oczywiście rozumiane w sposób, w jaki obiektowość prezentuje język JavaScript. Przy użyciu przekazanych parametrów tworzone są wszystkie obiekty potrzebne do wizualizacji, a także liczone są specjalne macierze modelu-widoku oraz perspektywy. Ostatecznie, na ekranie rysowane są wierzchołki zdefiniowanego prostopadłościanu w utworzonej przy pomocy wszystkich ustawień scenie. Na tym etapie można wstępnie ocenić poprawność kalibracji kamery, gdyż linie łączące wierzchołki prostopadłościanu są (lub nie) prostopadłe lub równoległe do ścian otoczenia, jeżeli położenie wierzchołków zostało zdefiniowane w prawidłowy sposób.

c) Wprowadzenie danych dotyczących symulacji

Po ocenie sceny, użytkownik wprowadza dane dotyczące obiektu, którego celem będzie poruszanie się wewnątrz prostopadłościanu. Dane te, to rozmiar oraz położenie.

d) Uruchomienie symulacji i interakcja

Ostatecznym krokiem jest uruchomienie symulacji. Parametry obiektu (sfery) poruszającego się wewnątrz prostopadłościanu zostają przekazane do głównej klasy, a następnie tworzony jest sam obiekt. Zaczyna się poruszać według początkowego wektora prędkości, a po dotarciu do ściany prostopadłościanu „odbija się", czyli odwraca zwrot wektora prędkości w kierunku, w którym spotkał się ze ścianą. Ma to na celu zaprezentowanie poruszającego się przedmiotu w stworzonej

przestrzeni, w celu dokładnej oceny poprawności kalibracji i doboru parametrów. W czasie symulacji użytkownik może dodawać wektory prędkości do poruszającej się sfery.

5. Symulacja i Testowanie

Modelowanie i symulacja

Testowanie podzielono na dwie części. Testowanie części statycznej oraz testowanie części dynamicznej. Testując część statyczną zwracano uwagę na poprawność miejsca wyświetlania zdefiniowanych punktów w utworzonej przestrzeni 3D. W tym przypadku stwierdzono konieczność umiejscawiania kamery obserwatora „w środku" zdefiniowanego prostopadłościanu, to jest – pomiędzy jego wierzchołkami. W innym przypadku występowało prawdopodobieństwo nałożenia się obiektów i zmniejszenia czytelności obrazu.

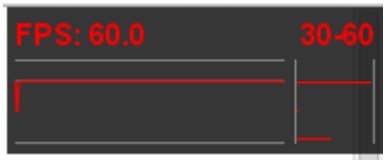
Część dynamiczną przetestowano pod kątem ilości wyświetlanych klatek na sekundę. Wyniki testów prezentują się w tabeli nr 1.

Tabela 1 – Wyniki testów animacji

Platforma	Procesor	RAM	Karta graficzna	Ilość klatek na sekundę
PC	Intel Core i5-2410M @2.3 GHz (2 cores)	4 GB	NVIDIA GeForce GT 520M	56-60
PC	Intel Core i7-2670QM @2.2 GHz (4 cores)	8 GB	NVIDIA GeForce GT 555M	~60
Samsung Galaxy S4	Exynos 5 Octa @1.6 GHz (4 cores)	2 GB	PowerVR SGX544MP3	45-60

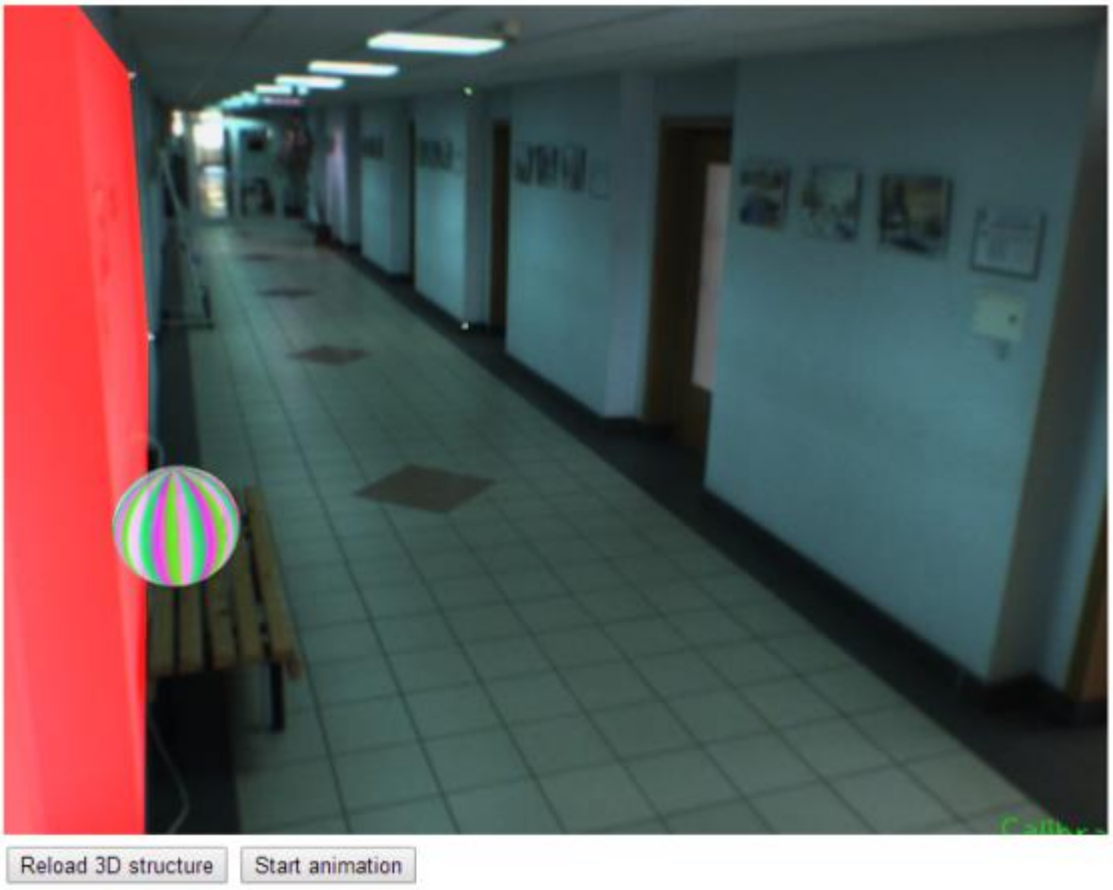
W przypadku wszystkich platform podczas animacji występowały, z małą częstotliwością, „przycięcia". Można to rozumieć jako nagłe zatrzymanie się obrazu na parędziesiąt milisekund (na pewno zauważalną dla oka ilość czasu).

Testy przeprowadzono przy pomocy przeglądarki Google Chrome. Odczyt klatek na sekundę został dokonany przy użyciu wbudowanej funkcjonalności przeglądarki. Przykładowy odczyt widnieje na rysunku nr 1.



Rysunek 2 - Odczyt klatek na sekundę

Dla utworzenia przykładowej animacji wykorzystano dane zaczerpnięte z projektu o bardzo podobnej treści [4]. Użyto zdjęcia z kamery, uzupełniono obliczone parametry kamery i zdefiniowano punkty charakterystyczne. W wyniku tych działań otrzymano scenę 3D prezentującą się na rysunku nr 2.



Rysunek 3- Przykładowa animacja

6. Rezultaty i wnioski

Wydajność działania aplikacji okazała się duża. Ilość klatek na sekundę jest bardzo zadowalająca, jednak istnieją wymienione wcześniej „przycięcia”. Z powodu faktu, iż zakładana ilość operacji związana z poziomem komplikacji problemu, powinna być niewielka, podejrzewa się, iż problem leży w implementacji. Pomimo zastosowania wymienionego w dziale nr 3 algorytmu indeksacji, prawdopodobnie faktyczna ilość obliczeń jest wielokrotnie większa niż w przypadku optymalnego podejścia. Inną możliwością jest występowanie wycieków pamięci powodujących

przepełnienie buforów i charakterystyczne „przycięcie” w momencie ingerencji systemu operacyjnego. Są to jednak jedynie domysły, gdyż nie przeprowadzono badań mających na celu określenie przyczyny tego problemu.

7. Podsumowanie

Powstała aplikacja częściowo spełniła założenia projektantów. Początkowa koncepcja opierała się na zrealizowaniu aplikacji uzupełniającej rzeczywisty obraz z kamery wirtualnymi obiektami w scenie 3D. Nie zrealizowano jednak tej funkcjonalności. Jedynym elementem rzeczywistości jest zdjęcie, będące tłem dla sceny.

Dodatkowym atutem jest panel pozwalający na wprowadzenie wielu parametrów kamery oraz punktów w przestrzeni. Dzięki temu możliwe jest przeprowadzenie analizy dokładności dokonanej kalibracji kamery oraz wyliczonych parametrów. Ilość ustawień oraz przycisków w panelu operatorskim jest bardzo duża. Stanowi to problem stricte praktyczny, jednak nie użyto wczytywania z pliku. Spowodowane jest to chęcią stworzenia panelu, w którym użytkownik może na bieżąco korygować swoje ustawienia w łatwy sposób.

Wydajność operacji liczących animacje nie jest idealna. Pomimo dużej ilości klatek na sekundę występuje problem opisany w poprzednich działach. Błędem projektantów przede wszystkim był brak rozpoznania problemu i ewentualnego dojścia do jego rozwiązania.

8. Literatura

Przykład technika odwoływania się do literatury: *„W [1] można znaleźć szczegółowe informacje dotyczące pisania publikacji i sposobów prowadzenia badań naukowych”*.

[1] http://www.w3schools.com/html/

[2] http://www.w3schools.com/js/

[3] http://learningwebgl.com/blog/

[4] Pękala Ł., Podolski Ł., “Wizualizacja scen 3D z wykorzystaniem WebGL”, Akademia Górniczo-Hutnicza, Kraków 2013

[5] http://www.chadvernon.com/blog/resources/directx9/vertex-and-index-buffers/

9. DODATEK A: Szczegółowy opis zadania

Specyfikacja projektu

Celem projektu jest wizualizacja scen 3D dla potrzeb systemu monitoringu z wykorzystaniem języka JavaScript oraz API WebGL. Oczekiwanym rezultatem jest zbiór stron internetowych prezentujących trójwymiarowe sceny odwzorowujące monitorowane obiekty wraz z naniesionymi dodatkowymi informacjami: alarmy, obszary strefy dozoru, trajektorie ruchu obiektów (przechodniów, pojazdów). Należy opracować protokół komunikacyjny umożliwiający przesyłanie sygnału wizyjnego z kamery oraz dodatkowych informacji z modułów detekcji celem ich prezentacji. Interaktywna strona powinna umożliwiać przeprowadzenie kalibracji kamery oraz zmianę parametrów wizualizacji przy pomocy myszki i klawiatury jak również niwelacji perspektywy wybranych obszarów obrazu. W zakres projektu wchodzi przetestowanie aplikacji na różnych platformach – również mobilnych.

Szczegółowy opis realizowanych algorytmów przetwarzania danych.

W kontekście istotnych dla technologii algorytmów zaimplementowano następujące metody będące praktycznie szkieletem całego programu:

- onReload3DStructureButton()

  Jest to pierwsza najważniejsza metoda odczytująca wszelkie parametry ze statycznej strony html, tworząca główny obiekt programu i przesyłająca niezbędne argumenty.

- onStartAnimationButton()

  Funkcja przekazująca dane dotyczące animacji i uruchamiająca ją.

- initGL()

  Metoda uruchamiająca środowisko WebGL. W razie braku obsługi WebGL na danej platformie, metoda ta komunikuje o tym użytkownika.

- init3DScene()

  Funkcja ta służy do stworzenia wszelkich obiektów (ścian, sfer) posiadając dane odebrane od użytkownika.

- init3DSceneBuffers()

  Metoda ta inicjalizuje bufory położenia wierzchołków i ich indeksów, a także koloru. W rzeczywistości każdy element mający być narysowanym na ekranie musi posiadać konkretne

bufory zainicjalizowane przez specjalne funkcje WebGL. Dzięki temu wykonanie koniecznych obliczeń jest przeprowadzone za pomocą programów zwanych shader'ami.

- initShaders()

W tym miejscu następuje stworzenie programów wspomnianych w poprzednim podpunkcie. W przypadku tej aplikacji głównym celem shader'ów (a właściwie vertex shader'a) było wyliczenie współrzędnych wierzchołków. Operacja ta w istocie polega na obliczeniu następującego działania:

$$position = perspectiveMatrix * modelViewMatrix * vertexPosition$$

Z tego równania wynika, iż prawdziwa pozycja wierzchołka jest zależna od względnej pozycji i przemnożona przez 2 macierze opisane w kolejnym podpunkcie.

- prepareCamera()

Jest to metoda przygotowująca dwie najważniejsze macierze programu, będące właściwie kluczowymi elementami. Macierze te, oprócz położenia wierzchołków obiektów, w całości charakteryzują wygląd sceny 3D. Są to macierz modelu-widoku oraz macierz perspektywy. Po pobraniu od użytkownika istotnych danych macierze te tworzone są według wzoru:

$$modelViewMatrix = \begin{bmatrix} rotXX & rotXY & rotXZ & transX \\ rotYX & rotYY & rotYZ & transY \\ rotZX & rotZY & rotZZ & transZ \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Gdzie:

$rotIJ$ – odpowiednie rotacje sceny 3D wzdłuż odpowiednich osi I, J

$transI$ – odpowiednia translacja sceny 3D wzdłuż osi I

$$perspectiveMatrix = \begin{bmatrix} \frac{2}{x_2 - x_1} & 0 & 0 & -\frac{x_2 + x_1}{x_2 - x_1} \\ 0 & \frac{2}{y_2 - y_1} & 0 & -\frac{y_2 + y_2}{y_2 - y_1} \\ 0 & 0 & -\frac{2}{z_2 - z_1} & -\frac{z_2 + z_1}{z_2 - z_1} \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \alpha & 0 & pX & 0 \\ 0 & \beta & pY & 0 \\ 0 & 0 & -(n+f) & n*f \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Gdzie:

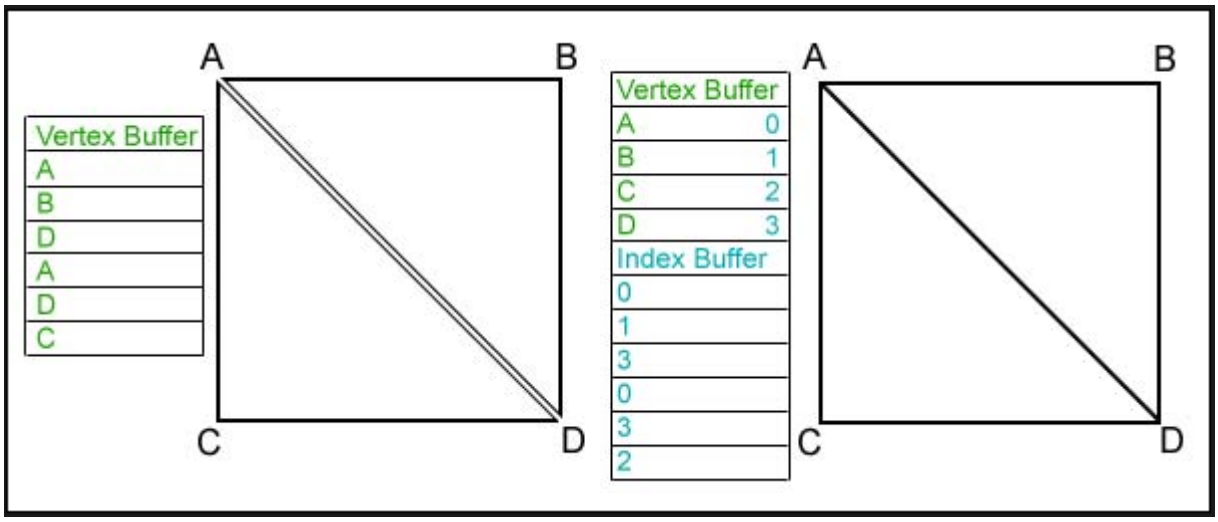
x_i, y_i, z_i- współrzędne zależne bezpośrednio od wielkości wyświetlanego tła oraz narzuconych przez użytkownika ograniczeń widoczności (najbliższy i najdalszy widzialny punkt)
α, β – parametry kamery związane z ogniskowymi w obu osiach.
pX, pY- współrzędne punktu centralnego kamery w pikselach.

n, f- określony przez użytkownika najbliżej i najdalej widzialny punkt.

W rzeczywistości pierwszą z dwóch składających się na macierz perspektywy macierz tworzy się przy pomocy funkcji ortho() podając jej potrzebne argumenty.

- animate()

Ostateczna funkcja rysująca obiekty na ekranie. Po przekazaniu do shader'ów wymaganych wartości następuje rysowanie obiektów. Wszelkie obiekty są reprezentowane przez trójkąty, więc rysowanie każdego obiektu sprowadza się do rysowania pewnej ilości trójkątów. Zastosowany algorytm indeksacji zwiększa wydajność tej operacji. Jego działanie polega na stworzeniu macierzy z indeksami kolejnych wierzchołków, aby nie rysować powtarzających się linii. Sens działania algorytmu pokazuje rysunek nr 4.



Rysunek 4 - algorytm indeksacji [5]

Jak widać na rysunku nr 4, algorytm indeksacji pozwala na narysowanie dwóch trójkątów bez przerysowywania lini AD.

10. DODATEK B: Dokumentacja techniczna

Procedura symulacji i testowania:

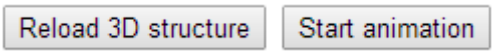
Program posiada w kodzie zaimplementowane wykorzystane biblioteki. Znajdują się w folderze „src/dependencies” i są to:

- gl-matrix-min – biblioteka używana do obliczeń związanych z macierzami i wektorami. Szczególnie przydatnymi funkcjonalnościami są translacje, rotacje i inne transformacje macierzowe.
- webgl-utils – biblioteka zapewniająca współpracę z większością przeglądarek internetowych

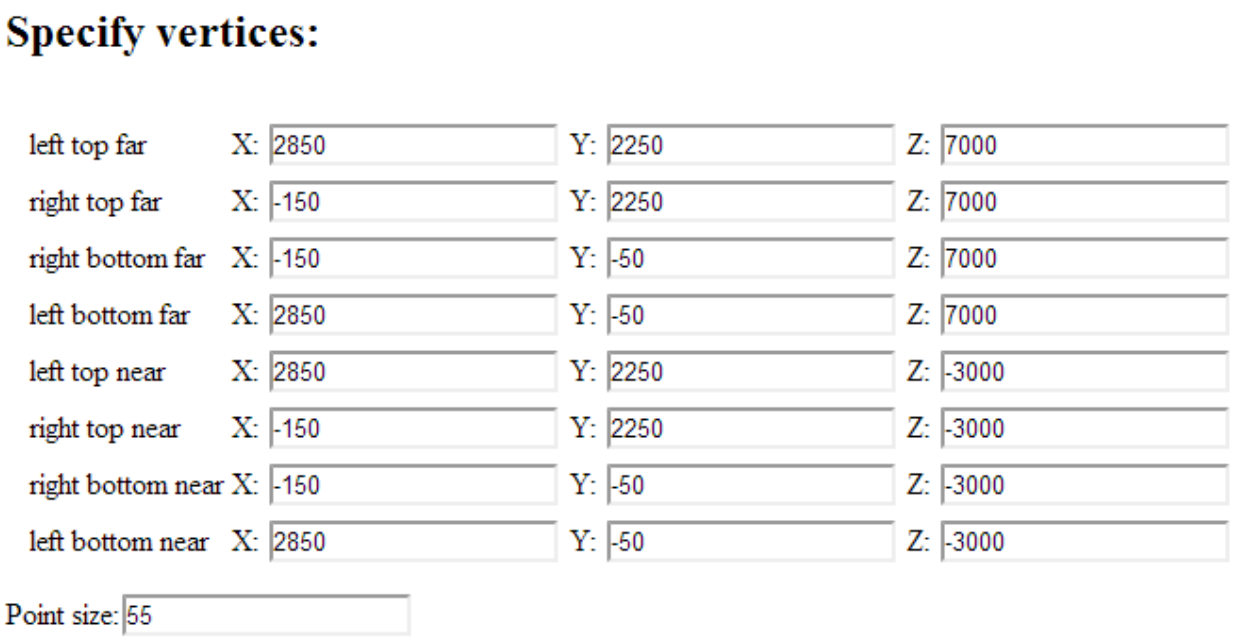
Do uruchomienia niezbędna jest przeglądarka internetowa posiadająca wsparcie ze strony WebGL (należy pamiętać o ewentualnym jego uruchomieniu). Aby włączyć program należy uruchomić plik „index.html” znajdujący się w folderze „src”.

Po uruchomieniu aplikacji pojawia się panel operatorski. Opis panelu przedstawiają poniższe rysunki:

Rysunek 5 - Właściwy obszar wizualizacji



Rysunek 6 - Przyciski wczytania struktury 3D oraz startu animacji

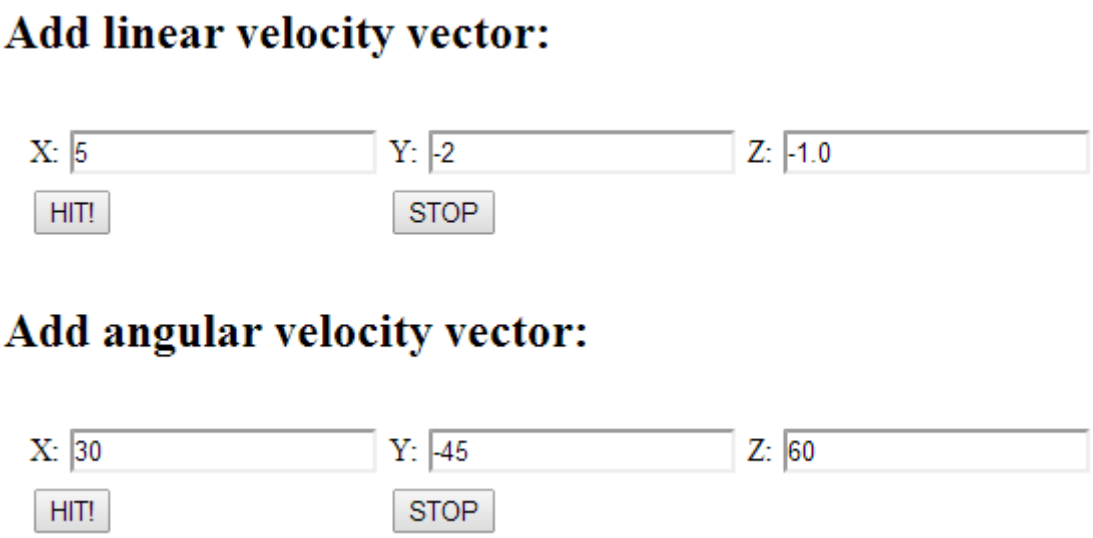


Rysunek 7 - Obszar definiowania punktów prostopadłościanu i określenia wielkości punktów

Rysunek 8 - Obszar wprowadzania ustawień kamery

Rysunek 9 - Obszar wprowadzania ustawień obiektu poruszającego się

Rysunek 10 - Obszar wczytywania tła



Rysunek 11 - Obszar wprowadzania dodatkowych wektorów podczas animacji

Po uruchomieniu pliku głównego należy po kolei:

- Wczytać tło
- Zdefiniować parametry kamery
- Zdefiniować punkty prostopadłościanu oraz rozmiar punktów
- Wczytać strukturę 3D (można pominąć ten krok)
- Zdefiniować właściwości obiektu poruszającego się
- Włączyć animację
- Ewentualnie dodawać wektory prędkości

Wszelkie wartości są domyślnie ustawione według szablonu.

11. DODATEK D: Spis zawartości dołączonego nośnika (płyta CD ROM)

W poszczególnych folderach nośnika, w zależności od rodzaju projektu, znajdują się:

- **SRC** - Kompletne foldery robocze wraz z plikiem uruchomieniowym „index.html”.
  - SRC/dependencies – folder z używanymi bibliotekami
  - SRC/objects – folder przechowujący pliki ze zdefiniowanymi klasami
- **DOC** - aktualna wersja niniejszego raportu w postaci elektronicznej (MS WORD oraz PDF)

12. DODATEK E: Historia zmian

Tabela 1 Historia zmian.

Autor	Data	Opis zmian	Wersja
Ziemowit Rachwał Oskar Kuligowski	2014-01-26	Utworzono pierwszą wersję raportu	1.0

PR13wsw509			**Karta oceny projektu**
Data	Ocena	Podpis	Uwagi