PW OKNO Wydział Elektryczny		Przedmiot:	**Grafika Komputerowa i Wizualizacja**
Kierunek:	Informatyka stosowana	Zadanie:	Projekt
Autor:	**Mirosław Rychel**	Temat:	**Kamera wirtualna**
Data:	10 września 2016	Etap: 1 i 2	Obraz rzutowany z obcinaniem krawędzi, oraz z ukrywaniem elementów zasłoniętych

Sprawozdanie

1. Cel i temat projektu:

   Celem projektu było napisanie programu komputerowego, w dowolnej technologii i języku programowania, realizującego tworzenie obrazu poprzez wirtualną kamerę**. Program realizuje obie wersje zadania: zarówno przycinanie krawędzi wystających poza obiektyw jak i algorytm zamalowywania powierzchni zasłoniętych. Wybór wersji odbywa się po ustawieniu odpowiedniego CheckBox'a w menu.**

2. Założenia:
   a. Prosta scena zawierająca kilka prostopadłościanów (analogicznie do widoku budynków w terenie). Elementy sceny nie stykają się i nie przenikają.
   b. Wszystkie warianty położenia kamery powinny generować poprawny obraz.
   c. Program musi uruchamiać się w standardowym systemie, bez instalacji dodatkowych bibliotek.
3. Zastosowane rozwiązania:
   a. Program został napisany w języku Java bez zastosowania dodatkowych bibliotek.
   b. Program działa w systemie Windows, ale język programowania Java umożliwia skompilowanie go dla innych systemów.
   c. Parametry kamery są ustawiane przy pomocy elementów menu.
   d. Zdefiniowane zostały podstawowe ruchy kamery:
      i. Przesunięcia w trzech osiach.
      ii. Obroty wokół trzech osi.
      iii. Powiększanie obrazu w trzech kierunkach.
      iv. Pochylanie obrazu.
      v. Zoom: przybliżanie i oddalanie (realizowane poprzez zmianę odległości kamery od rzutni, przy zachowaniu wymiarów rzutni)
      vi. Home: zawsze powrót do pozycji startowej
   e. Definicje rozmiaru okna, obszaru widzialnego, położenia i rozmiarów prostopadłościanów, wielkości skoku translacji i obrotów, odległości kamery od rzutni znajdują się w kodzie programu (klasa Algorytumy.java).

f. Przyjęto zasadę, że rzutnia ma wielkość zdefiniowanego obszaru widzialnego i znajduje się na płaszczyźnie z = 0 (lewoskrętny układ obserwatora położonego na osi z w punkcie z = -d).
g. Program nie wyświetla obiektów, które przecinają płaszczyznę rzutni.

4. Opis działania programu:

a. Utworzenie tablicy dynamicznej na bieżące współrzędne wierzchołków i przypisanie do niej wartości początkowych. Tablica ta określa zawsze wyjściowe, początkowe położenie obiektów. Po każdej zmianie parametrów dane te są przeliczane w celu wyświetlenia aktualnego obrazu.
b. Utworzenie obrazu startowego na podstawie początkowych parametrów.
c. Utworzenie obramowania okna oraz menu.
d. Oczekiwanie na ustawienie wartości parametrów w menu.
e. Przeliczenie współrzędnych wszystkich wierzchołków przy użyciu:
    i. Dla translacji: dodania lub odjęcia skoku do odpowiedniej współrzędnej
    ii. Dla obrotów: odpowiednie mnożenie i normalizacja uprzednio zdefiniowanych macierzy obrotów (transponowanych) z wektorem położenia wierzchołka
    iii. Dla zoom: zmniejszenie lub zwiększenie odległości obserwatora od rzutni (dostępny zakres d>0)
    iv. Dla „Home" przepisanie danych startowych
f. Wyczyszczenie okna
g. Utworzenie nowego obrazu

5. Opis struktury programu:

Klasa **Kamera** w pakiecie **interfejs_uzytkownika** zawiera elementy pakietu Java Swing niezbędne do wyświetlenia okna. Następnie tworzone jest menu oraz instancja podklasy **Obiektyw**, która zajmuje się narysowaniem obiektów z uzyskanej kolekcji.

Klasa **Algorytmy** z pakietu o tej samej nazwie dostarcza klasie **Kamera** trzy kolekcje: parametrów trójwymiarowych, parametrów jednowymiarowych oraz elementów (linii lub płaszczyzn) do narysowania na ekranie. **Algorytmy** dziedziczą wszystkie klasy z pakietu **narzędzia**. Są to kolejne stopnie abstrakcji elementów obrazu: **Punkt3D, Punkty3D, Odcinek, Odcinki, Plaszczyzna, Plaszczyzny**. Są to niewielkie struktury i z punktu widzenia samego programu można ich było nie wydzielać, ale dzięki temu program jest trochę bardziej przejrzysty i jest przygotowany do ewentualnej rozbudowy. Klasa zawiera także realizację czterech algorytmów: przekształceń 3D, rzutowania obrazu, przycinania krawędzi wystających poza ekran oraz zasłaniania powierzchni. W zasadzie można było wydzielić same algorytmy od parametrów i struktur przechowujących dane, ale nie poprawiłoby to czytelności programu, bo algorytmy są tylko cztery, za to skomplikowało by komunikację, gdyż parametry, zastosowane algorytmy oraz przetwarzane dane są ze sobą ściśle powiązane.

Zastosowałem następujące usprawnienia oraz algorytmy:

a. Dane o obrazie są reprezentowane w postaci jednej tabeli zawierającej jedynie współrzędne wierzchołków. Inne struktury, linie i sześciany są wskaźnikami do tej tabeli, co pozwala uniknąć redundancji danych.
b. Przy każdym odświeżeniu obrazu dane są pobierane z tabeli początkowej i przeliczane z użyciem liczb zmiennoprzecinkowych. Pozwoliło to uniknąć nakładania się błędów oraz błędów zaokrągleń.
c. Pierwszym, naturalnym podejściem do problemu transformacji obrazu 3D jest zastosowanie macierzy. Nie jestem jednak przekonany do stosowania algorytmów klasy O(n3) w celu wymnażania kolejnych macierzy przekształceń. Zastosowałem przeliczone już wzory. Dodatkowo w przypadku obrotów macierz przeliczana jest tylko raz, gdy zmienią się parametry. Następnie przy wyliczaniu kolejnych wierzchołków wykorzystywane są już gotowe „stałe". Ogólnie złożoność obliczeniowa transformacji to O(n) gdzie n to liczba wierzchołków znajdujących się w wygenerowanej przestrzeni.
d. Struktura programu nie ogranicza go jedynie do rysowania odcinków prostych i do sześcianów złożonych z prostokątów. Nic nie stoi na przeszkodzie, żeby program operował dowolnymi odcinkami złożonymi oraz figurami przestrzennymi złożonymi z dowolnych płaszczyzn.
e. Przy realizacji dwóch głównych funkcji programu zastosowałem **algorytmy Cohena-Sutherlanda** (przycinania krawędzi) oraz **algorytm malarski** zamalowywania powierzchni niewidocznych, z wyliczaniem położenia środków ciężkości jako wyznacznika odległości od widza. Nie są to algorytmy oferujące najdokładniejsze wyniki, ale za to proste i przejrzyste w implementacji.

6. Procedura tworzenia obrazu

a. Sprawdzenie, czy obiekt nie przecina rzutni, a jeśli przecina, to pominięcie go przy wyświetlaniu
b. Rysowanie po kolei wszystkich zadeklarowanych wcześniej prostopadłościanów
c. Rzutowanie wszystkich wierzchołków prostopadłościanu
d. Sprawdzanie i oznaczenie położenia każdego wierzchołka na rzutni według algorytmu Cohena-Sutherlanda
e. Rysowanie 3 krawędzi wychodzących z jednego wierzchołka, powtarzane 4 razy w celu utworzenia obrazu całego prostopadłościanu
    i. Podczas rysowania każdej krawędzi analizowane jest położenie wierzchołków i ewentualne odpowiednie obcięcie krawędzi jedno lub dwustronne lub jej odrzucenie
    ii. Dla odróżnienia w trakcie testów krawędzie ścian przedniej, tylnej oraz boczne łączące (przypisanie nomenklatury na podstawie pozycji startowej) są rysowane przy użyciu różnych kolorów
f. Program posiada również procedurę rysowania prostopadłościanów bez obcinania (niewykorzystywaną w obecnej wersji programu).

7. Ocena uzyskanego efektu

Zakładana funkcjonalność została zrealizowana poprawnie. Podczas różnorodnych testów wszystkie obrazy były wyświetlane zgodnie z wyobrażeniem. Scenę można oglądać z każdej strony. Obejście jej o kąt pełny daje obraz zgodny z obrazem startowym.

Niestety w programie między innymi następujące zagadnienia wymagają jeszcze poprawy:

- Program działa poprawnie jedynie dla małych kątów obrotu. Sprawdzenia wymagałoby prawdopodobnie zachowanie programu gdy współrzędne przyjmują wartości ujemne i poprawność obliczeń odległości punktów dla tych wartości. Szczególnie mocno widać to właśnie przy obrotach. Warto byłoby też sprawdzić jeszcze raz poprawność wyliczania macierzy obrotów.

- Algorytmy niepoprawnie przetwarzają obraz gdy „wchodzi" się do wnętrza jakiejś bryły, przycinanie krawędzi niedokładnie ustala punkt w którym należy obciąć krawędź, algorytm malarski nie radzi sobie z powierzchniami równoległymi do płaszczyzny XZ. Być może dobre efekty dałoby zastosowanie algorytmu przycinania do frustrum.

- Od strony zastosowania Javy program wymaga uzupełnienia o obsługę klawiszy w celu zmiany parametrów, o umożliwienie zmiany rozmiarów okna programu, dobrze byłoby też dodać menu w którym można byłoby sterować skokiem zmiany wartości parametrów oraz ich wartościami domyślnymi i granicznymi. Obecnie jest to ustalone na stałe w kodzie programu.

Do mocnych stron programu należą:

- Przejrzysta struktura, zastosowane zostały proste algorytmy i udało się ich nie zagmatwać i nie pogrążyć wśród zbędnych obliczeń. Gdyby przeprowadzić testy programu pod kątem obciążenia jestem przekonany, że działałby dość wydajnie.

- Istnieją możliwości rozbudowy i uzupełniania struktury bez potrzeby budowania wszystkiego od początku. Dzięki zastosowaniu języka Java i związanej z nim obiektowości można bez trudu przystosować kod do użycia innych algorytmów i platform sprzętowych.

8. Pisząc program korzystałem między innymi z następujących stron:

http://mst.mimuw.edu.pl/lecture.php?lecture=gk1&part=Ch3
http://eduinf.waw.pl/inf/utils/002_roz/2008_07.php
http://www.asawicki.info/productions/artykuly/Zaawansowana_kamera_3D.php5
http://www.math.uni.lodz.pl/~marekbad/files/grafika/cg_exam/Wyklad_04_Rzutowanie.pdf
https://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula
http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV0405/REDSTONE/AxisAngleRotation.html
https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
https://en.wikipedia.org/wiki/3D_projection
http://www.procto.biz/software-technology/courseware-for-hidden-linesurface-removal
https://sites.google.com/site/glennmurray/Home/rotation-matrices-and-formulas
http://inside.mines.edu/fs_home/gmurray/ArbitraryAxisRotation/
http://www.mat.uniroma2.it/~picard/SMC/didattica/materiali_did/Java/Java_3D/Java_3D_Programming.pdf