# **WOJSKOWA AKADEMIA TECHNICZNA**

im. Jarosława Dąbrowskiego

**WYDZIAŁ MECHATRONIKI I LOTNICTWA**



**PRACA DYPLOMOWA**

|  |  |
| --- | --- |
| Temat pracy: | Projekt wyświetlacza wielofunkcyjnego |
|  | MFD do symulatora samolotu odrzutowego F16 |

|  |
| --- |
| Dawid Mateusz Kwiecień |
|  |

(stopień, imiona i nazwisko dyplomanta)

|  |
| --- |
| Lotnictwo i kosmonautyka |
|  |

(kierunek studiów)

|  |
| --- |
| Awionika |
|  |

(specjalność)

|  |
| --- |
| ppłk dr inż. Mariusz Ważny |
|  |

(tytuł/stopień, imię i nazwisko kierownika pracy)

**WARSZAWA 2015**

SPIS TREŚCI

**Indeks skrótów** ............................................................................................................. 8

**Wykaz oznaczeń** ........................................................................................................... 9

**WPROWADZENIE** ..................................................................................................... 10

**1. PRZEGLĄD I ANALIZA STOSOWANYCH ROZWIĄZAŃ SYSTEMÓW ZOBRAZOWANIA INFORMACJI NA WSPÓŁCZESNYCH WOJSKOWYCH STATKACH POWIETRZNYCH**.......................................... 12  
**1.1.Historia i rozwój systemów zobrazowani informacji**………..…………….. 12

**1.2. Cechy współczesnych systemów zobrazowania informacji**……….............. 16

**1.3**. **Analiza wybranych przyrządów i wskaźników**.............................................. 19

**1.3.1. Tradycyjne wskaźniki mechaniczne i elektromechaniczne**……….... 19

**1.3.2.** **Wskaźniki zintegrowane i dyspozycyjne**…………………............... 21

**1.3.3.** **Wskaźniki elektroniczne**..................................................................... 22

**1.4.** **Rola oprogramowania w systemach zobrazowania informacji**.................... 28

**1.5.** **Tendencje i kierunki rozwojowe systemów zobrazowania informacji** ........ 28

**2. ANALIZA BUDOWY I ZASADY DZIAŁANIA SYSTEMU ZOBRAZOWANIA INFORMACJI ORAZ RODZAJU WYŚWIETLANYCH INFORMACJI NA WIELOFUNKCYJNYCH WYŚWIETLACZACH MFD SAMOLOTU F-16**………………………………………………………………… 30

**2.1.** **Ogólna koncepcja budowy i pracy systemu**…. .………………..................... 30

**2.2. Profile misji i różnice w działaniu systemów**…………………………….... 32

**2.2. Wyświetlacze MFD** ........................................................................................... 33

**2.2.1. Menu HSD**…………………………………………………………....... 35

**2.2.2. Menu SMS**…………………………………………………………....... 37

**2.2.3. Menu TFR**…………………………………………………………....... 42

**2.2.4. Menu DTE**………………………………………………………........... 43

**2.2.5. Menu FLCS**………………………………………………………......... 44

**2.2.6. Menu FLIR**…………………………………………………………..... 45

**2.2.7. Menu WPN**…………………………………………………………...... 46

**2.2.8. Menu TGP**…………………………………………………………....... 47

**2.2.9. Menu HAD**…………………………………………………………....... 47

**2.2.10. Menu BLANK**……………………………………………………....... 48

**2.2.12. Menu FCR**…………………………………………………………….. 49

**2.2.13. Menu RESET MENU**………………………………………………… 52

**3. OPRACOWANIE KONCEPCJI PRACY URZĄDZENIA MFD Z WYKORZYSTANIEM ISTNIEJĄCEGO STANOWISKA SYMULATORA SAMOLOTU ODRZUTOWEGO**………………………………...………….…54

**3.1.** **Przyjęte założenia** ……………………………………………………………. 54

**3.2.** **Opis stanowiska symulatora**………………………………………………… 55

**3.3.** **Realizacja ekranu MFD**. …………………………………...……………....... 59

**3.3.** **Techniki transmisji danych i współdzielenia zasobów w systemie symulator – wskaźnik MFD**…. .………………………………………………….………....... 61

**3.3.** **Możliwości rozbudowy stanowiska i wskaźników MFD .**……………........ 63

**4. PROJEKT I WYKONANIE OPROGRAMOWANIA SYMULUJĄCEGO DZIAŁANIE MFD ORAZ JEGO INTEGRACJA ZE STANOWISKIEM SYMULATORA SAMOLOTU F-16**………………………………...………….66

**4.1 Założenia projektowe i wymagania stawiane aplikacji**……………………. 67

**4.2 Opis środowiska i wykorzystanych narzędzi**………………………………… 72

**4.2.1. Microsoft Visual Studio 2013**………………………………………….. 72

**4.2.2. Gimp 2**………………………………………………………………… 73

**4.2.3. Biblioteka OpenGL**…………………………………………………… 74

**4.2.4. Biblioteka Glew**…………………………….……………………........... 74

**4.2.5 Biblioteka Glm**………………………………………………………...... 75

**4.2.6. Biblioteka SOIL**………………………………………………………… 75

**4.2.7. Doxygen**………………………………………………………………… 76

**4.3. Aplikacja odbierająca i renderująca dane**………………………………… 76

**4.3.1. Punkt wejściowy i dyrektywy preprocesora**………………………… 78

**4.3.2. Główna pętla aplikacji i klasa okna**………………………………… 82

**4.3.3. Zarządzanie teksturami i geometrią**………………………………….. 90

**4.34. Obsługa sterowania użytkownika**.…………………………………….. 93

**4.3.5. Klasa parametrów lotu i wątek transmisji sieciowej** ……………....... 97

**4.3.5. Klasy okien dialogowych**……………………………………………... 102

**4.3.6. Zasoby aplikacji**……………………………………………………..... 104

**4.3.7. Programy cieniujące**………………………………………………..... 106

**4.4. Aplikacja serwerowa**…………………………………………………...….... 108

**5. WNIOSKI WYNIKAJĄCE Z REALIZACJI ZADANIA**……………………. 112

**6. BIBLIOGRAFIA**………………………………………………………………… 114

**7. ZAŁĄCZNIKI**…………………………………………………………………….. 115

**6.5. Nośnik CD**…………………………………………………………………… 115

Indeks skrótów

|  |  |  |
| --- | --- | --- |
| **Skrót** | **Znaczenie obcojęzyczne** | **Znaczenie polskie** |
| ADI | *Attitude Indicator* | Sztuczny horyzont |
| CRT | *Catode Ray Tube* | Lampa elektro-promieniowa |
| EADI | *Electronic AD* | Elektroniczny ADI |
| EHSI | *Electronic Horizontal Situation Indicator* | Elektroniczny HSI |
| FOV | *Field of View* | Pole widzenia |
| HDD | *Head-Down Display* | Wskaźniki poniżej linii wzroku |
| HLD | *Head-Level Display* | Wskaźniki na linii wzroku |
| HMD | *Helmet Mounted Display* | Wskaźnik nahełmowy |
| HSI | *Horizontal Situation Indicator* | Wskaźnik sytuacji horyzontalnej |
| HUD | *Head-Up Display* | Wskaźnik przezierny |
| ILS | *Instrumental Landing System* | System automatycznego lądowania |
| LCD | *Lyquid Crystal Display* | Ekran ciekłokrystaliczny |
| LED | *Light-emitting Diode* | Dioda elektroluminescencyjna |
| LPM |  | Lewy przycisk myszy |
| MFD | *Multi Funcion Display* | Ekran wielofunkcyjny |
| PDP | *Plasma Display Panel* | Ekran plazmowy |
| RGB | *Red-Green-Blue* | Czerwony-zielony-niebieski |
| RPM |  | Prawy przycisk myszy |
| SP |  | Statek powietrzny |
| TAS | *True Air Speed* | Prędkość Rzeczywista |
| TFT | *Thin Film Transistor* | Matryca sterowana tranzystorami |
| UV | *Ultra Violet* | Promieniowanie ultra fioletowe |

Wykaz oznaczeń

**WPROWADZENIE**

Nowoczesne wyposażenie pokładowe wraz z systemami zobrazowania informacji odgrywa ważną i doniosłą rolę w rozwoju techniki lotniczej. Obecnie coraz większą skuteczność na polu walki osiąga się poprzez wprowadzenie nowoczesnej cyfrowej techniki analizy i obróbki informacji oraz poprzez wykorzystanie komputerów do przedstawienia pilotowi-operatorowi w jak najbardziej ergonomiczny sposób wypracowanej podpowiedzi. Rosnąca moc obliczeniowa komputerów pozwala na dużą integrację przetwarzania informacji z wielu podsystemów pokładowych. Zadanie wypracowania najważniejszych informacji i przedstawienia ich użytkownikowi spoczywa na sprzęcie, który wykazuje się cechą adaptacyjności, tzn. sam musi dostosować swoje działanie do aktualnych warunków pracy, tak aby w jak największym stopniu odciążyć z tego obowiązku pilota. Równocześnie rozwijana jest technika audio-wizualna. Coraz powszechniej wypierane są tradycyjne wskaźniki analogowe na rzecz zintegrowanych i wielofunkcyjnych wskaźników w postaci paneli cyfrowych, gdzie możliwe jest zobrazowanie dowolnej informacji w zależności od warunków pracy układu lub upodobań użytkownika. Wszystko to składa się na część sprzętową opisywaną w literaturze jako „hardware”.

Rozwój sprzętu pociąga za sobą konieczność dostarczenia odpowiedniego oprogramowania (ang. „software”), które będzie odznaczało się cechą niezawodności, będzie w jak największym stopniu wykorzystywało moc obliczeniową sprzętu, spełniało wymagania postawione przez użytkownika oraz czy będzie „elastyczne” co do warunków w jakich pracuje. Zapotrzebowanie na odpowiednie oprogramowanie powoduje ciągły rozwój techniczny urządzeń. Wraz z rozwojem postępują możliwości symulowania pracy urządzenia lub systemu w sztucznie wykreowanych warunkach. Stworzenie realistycznych symulatorów pełni dwojaką rolę. Pozwala zarówno na doskonalenie umiejętności obsługi poszczególnych systemów przez użytkownika,   
a także dostarcza bezcennych informacji o tym czego oczekuje użytkownik i jakie informacje należy mu przekazywać. Problem symulacji poszczególnych systemów   
i podsystemów niekiedy wymaga większych nakładów sił i środków niż sam system zabudowany na statku powietrznym. Jednak korzyści jakie płyną z dostarczenia takich symulatorów są niekwestionowane.

Celem tej pracy jest przeanalizowanie działania wskaźnika wielofunkcyjnego samolotu F-16 oraz zaprojektowanie aplikacji symulującej jego pracę, zintegrowanej   
z symulatorem samolotu F-16. Aplikacja ta stanowić będzie pewien zarys problemu, poruszając jedynie najważniejsze kwestie symulacyjne. Oprogramowanie działać będzie na platformie Microsoft Windows. Do jego działania wykorzystano bibliotekę OpenGL. Wykonanie symulacji pracy wskaźnika wielofunkcyjnego MFD w takiej formie zostanie połączone z istniejącym stanowiskiem symulatora przy pomocy interfejsu sieciowego TCP/IP. Dane, które zostaną wysłane poprzez sieć, zbierać będzie niezależna aplikacja działająca na komputerze symulatora. Interakcja programu   
z użytkownikiem nastąpi poprzez dołączone do komputera z zainstalowaną aplikacją MFD urządzenie wskazujące (np. mysz optyczną).

Praca składa się z pięciu rozdziałów, które stanowią rozwinięcie problematyki zobrazowania informacji oraz opis działania wskaźnika MFD, stanowiska symulatora,   
a także opis zaprojektowanej aplikacji. W rozdziale pierwszym wprowadzono czytelnika w zagadnienia dotyczące wyposażenia pokładowego współczesnych bojowych statków powietrznych. Opisano rozwój systemów zobrazowania informacji na przestrzeni lat oraz przedstawiono ideę działania podstawowych przyrządów, które można znaleźć na wyposażeniu SP. W rozdziale drugim opisano budowę systemu zobrazowania informacji w samolocie F16 oraz przedstawiono jaki rodzaj informacji można znaleźć na wielofunkcyjnych wyświetlaczach MFD. Dokładnie opisano każdą z poszczególnych dostępnych podstron menu głównego. W trzecim rozdziale opisano koncepcję współpracy urządzenia MFD z istniejącym stanowiskiem symulatora, opisano to stanowisko, a także poprowadzono rozważania na temat przyszłościowych modernizacji symulatora. W rozdziale czwartym stanowiącym podstawę tej pracy opisano działanie aplikacji wraz z objaśnieniem elementów kluczowych w jej kodzie źródłowym, opisano wykorzystane biblioteki a także zastosowane techniki pozyskiwania obrazu. Rozdział piąty stanowi podsumowanie pracy oraz wskazanie przewidywanych kierunków rozwoju w zakresie dziedziny techniki zobrazowania informacji.

* 1. PRZEGLĄD WYPOSAŻENIA POKŁADOWEGO WSPÓŁCZESNYCH BOJOWYCH STATKÓW POWIETRZNYCH

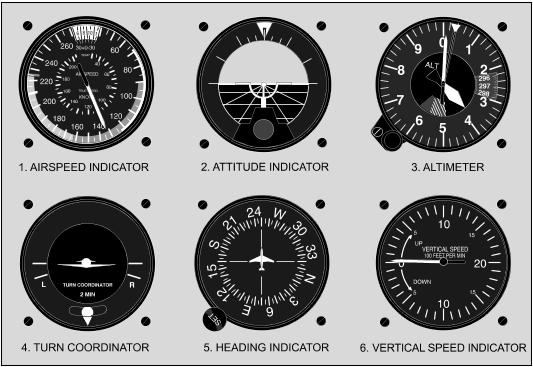
W rozdziale przybliżono zasadę działania współczesnych systemów zobrazowania informacji wykorzystywanych na wojskowych statkach powietrznych. Zdefiniowano pojęcie informacji, omówiono jaką rolę w systemie spełnia pilot, wskazano towarzyszące pilotowi ograniczenia percepcyjne i psychofizyczne takie jak czas reakcji na zaobserwowane zjawiska oraz czas potrzebny do wypracowania właściwych decyzji. Następnie przedstawiono wymagania stawiane współczesnym systemom zobrazowania informacji. Wyjaśniono jakimi cechami powinien odznaczać się wskaźnik wchodzący   
w skład systemu oraz w jaki sposób informacja powinna być dostarczana pilotowi.   
W dalszej części rozdziału scharakteryzowano konkretne rodzaje przyrządów poprzez omówienie ich budowy i podanie przykładów zastosowań na współczesnych bojowych statkach powietrznych. Krótko scharakteryzowano rolę jaką spełnia w systemie oprogramowanie i wymieniono najnowsze biblioteki graficzne obsługujące sprzęt służący do generowania obrazu w czasie rzeczywistym. Na koniec przedstawiono możliwości i kierunki rozwoju w zakresie technologii zobrazowania obrazu i grafiki komputerowej.

* 1. **Historia i rozwój systemów zobrazowania informacji**

Od początku istnienia lotnictwa, od kiedy człowiek wzniósł się w powietrze, aby móc prawidłowo i bezpiecznie wykonywać operacje lotnicze, potrzebował stałego dostępu do najistotniejszych danych i parametrów lotu. Były nimi np. aktualna pozycja statku powietrznego względem horyzontu, wysokość czy kierunek, w którym odbywał się lot. Pionierzy lotniczy odbierali informacje o tych parametrach głównie poprzez obserwację otoczenia statku powietrznego i samego płatowca. W tamtym okresie powodzenie prowadzenia operacji lotniczych zależało od warunków pogodowych -   
w przypadku braku odpowiedniej widoczności nie wykonywano lotu. Od lat 20-tych XX wieku, do obserwacji parametrów lotu, zaczęto powszechnie stosować urządzenia w postaci busol i kompasów magnetycznych. Rewolucja w lotnictwie nastąpiła pod koniec lat 20-tych ubiegłego stulecia, kiedy to niemiecki wynalazca Paul Kollsmann wynalazł pierwszy wysokościomierz barometryczny. W 1929 roku Jimmi Doolittle zasłynął z tego, że jako pierwszy wykonał w całości lot wg wskazań przyrządów, tzw. lot IFR (Instrumental Flight Rules). Kolejnym okresem, jaki nastał w erze rozwoju lotnictwa, był okres II wojny światowej. Nastąpił w nim dynamiczny skok w przemyśle lotniczym. Pojawienie się szybkich i zwrotnych maszyn myśliwskich wymagało precyzyjnej kontroli aktualnych parametrów lotu. Ponadto pewną część operacji stanowiły loty nocne, w których niezbędne były odpowiednie wskazania położenia SP. Najpopularniejszym rozwiązaniem w tamtym okresie okazała się koncepcja zastosowana w lotnictwie RAF. Na zobrazowanie informacji składało się sześć podstawowych przyrządów aerometrycznych, nazywanych także podstawową szóstką:

- wysokościomierz,  
 - prędkościomierz,  
 - zakrętomierz i chyłomierz poprzeczny,  
 - wariometr,  
 - sztuczny horyzont,  
 - wskaźnik kursu.

Na rysunku 1.1 przedstawiono rozmieszczenie podstawowej szóstki przyrządów lotniczych. W górnym rzędzie znajduje się prędkościomierz, dalej sztuczny horyzont oraz wysokościomierz barometryczny. W dolnym rzędzie zakrętomierz i chyłomierz poprzeczny, wskaźnik kursu oraz wariometr. Dzięki takiemu rozmieszczeniu przyrządów na tablicy pilot mógł efektywnie skupić swoją uwagę i podjąć efektywne działanie. Sposób w jaki pilot przenosił wzrok z poszczególnych wskaźników, zależał od aktualnie wykonywanego manewru albo od upodobań pilota.

Rys. 1.1 Podstawowa szóstka przyrządów lotniczych

Rozwiązanie takie okazało się niezwykle popularne i przyjęło się na długie lata powojenne. Na przełomie lat 60 i 70 popularna stała się technika kineskopowa. Urządzenie w postaci lampy kineskopowej, tj ekranu wielofunkcyjnego zaimplementowano w systemie awioniki samolotu F-111D. Ekran integrował wiele cennych informacji w jednym miejscu. Było to szczególnie istotne w przypadku dużych maszyn transportowych, w których przed zastosowaniem lampy, liczba wskaźników   
i przyrządów była znaczna. Wraz z upowszechnieniem lamp kineskopowych, coraz częściej zaczęto używać pojęcia wprowadzonego dzięki NASA - „Glass cockpit”. Jego idea polegała na wypieraniu analogowej techniki prezentacji danych w postaci wskaźników igłowych na rzecz coraz większych, lżejszych i liczniejszych wskaźników monitorowych. Technikę „Glass cockpit” zastosowano m. in. samolocie McDonell Douglas MD-80, produkowanego od 1979 roku. Kolejną rewolucją w rozwoju techniki zobrazowania informacji było wynalezienie monitorów ciekłokrystalicznych LCD, wykorzystywanych jako wskaźniki informacyjne od końca lat 90 w wielu maszynach wojskowych, cywilnych oraz promach kosmicznych. Choć w początkowej fazie rozwoju monitory LCD nie były pozbawione wad (np. były podatne na nadmierne oświetlenie słoneczne, co utrudniało prawidłowy odczyt wskazań), to zdecydowanie przewyższały tradycyjne wskaźniki kineskopowe jeśli chodzi o wymiary, masę i ilość zajmowanego miejsca. Wskaźnik tego typu po raz pierwszy został zamontowany m. in. w samolocie F-16 w wersji block 30. Obok wskaźników CRT i LCD rozwijały się niestandardowe techniki prezentacji danych. Najpopularniejszą z nich, która rozprzestrzeniła się nie tylko na przemysł lotniczy, stały się wskaźniki przezierne, nazywane wskaźnikami typu HUD (Head-up display). Idea ich działania polega na wyświetlaniu obrazu na przeźroczystej powierzchni, tak aby możliwa była jednoczesna obserwacja wyświetlanych danych oraz sytuacji za wskaźnikiem. Dlatego też najczęściej montowano i montuje się tego typu wskaźniki na wysokości wzroku pilota, tak aby mógł prowadzić nieustanną obserwację sytuacji przed statkiem powietrznym przy jednoczesnym dostępie do wskazywanych przez HUD parametrów lotu. Wykorzystuje go wiele maszyn bojowych jak choćby wspomniany F-16. Poniekąd   
z idei wskaźników przeziernych wyewoluował pomysł umieszczenia wskazań na hełmie pilota, tak aby operator statku powietrznego miał dostęp do danych w każdym czasie niezależnie od kierunku linii wzroku, przy jednoczesnej ciągłej kontroli i obserwacji otoczenia. Rozwiązanie to jest znane jako HMD (Helmet-mounted display)   
i powszechnie stosowane w nowoczesnych samolotach bojowych np. F-35 Lighting II.

Z powyższego wynika, że systemy zobrazowania informacji przeszły długą drogę ewolucji na przestrzeni rozwoju przemysłu lotniczego. Wpływ na ich rozwój miały różne dziedziny techniczne począwszy od techniki cyfrowej, układów scalonych aż do rozwoju grafiki komputerowej. Obecnie coraz częściej tradycyjna technika analogowa wypierana jest na rzecz nowych wielko monitorowych zintegrowanych wskaźników.   
W coraz większym stopniu znaczenie mają preferencje pilota co do sposobu prezentacji danych oraz aktualne warunki pracy. To pilot, wykorzystując elastyczność zintegrowanych monitorów wielofunkcyjnych, sam dostosowuje przyrząd i decyduje   
o tym w jaki sposób chce, aby dane zostały mu prezentowane. Bardzo duże znaczenie   
w terminologii lotniczej nabrała wielozadaniowość. Współczesny samolot bojowy musi odznaczać się czymś więcej niż tylko dobrymi osiągami. Musi przede wszystkim sprostać różnym misjom bojowym, przechwytującym albo wywalczającym przewagę powietrzną. Ta różnorodność zadań niesie za sobą konieczność różnego sposobu prezentowania informacji, które powinny spełniać projektowane systemy.

* 1. **Cechy systemów zobrazowania informacji**

System zobrazowania informacji, podobnie jak każdy inny system dopuszczony do użytkowania na SP, musi odznaczać się specyficznymi właściwościami oraz spełniać wymagania niezawodności i należytej pracy. W dużym stopniu o efektywności działania systemu decyduje to, w jaki sposób zachodzi interakcja między nim, a użytkownikiem docelowym. Relację tę opisują takie dziedziny nauki jak ergonomia, psychologia czy czynnik ludzki. W zagadnieniach dotyczących systemów zobrazowania informacji dużą część zainteresowań poświęca się ograniczeniom człowieka w odbieraniu, przetwarzaniu i wypracowywaniu informacji. Analizując te ograniczenia można jednoznacznie wykazać czym powinien cechować się system.

Na wstępie należy wyjaśnić jak człowiek odbiera, interpretuje i wypracowuje informację w przypadku systemów zobrazowania informacji w lotnictwie. Człowiek posiada siedem zmysłów. Nie wszystkich używa w jednakowym stopniu. W przypadku pilotażu najbardziej istotnym jest wzrok, nieco mniej słuch i zmysł równowagi. Ludzkie oko jest czułe na światło z zakresu fal od 480 [nm] do 780 [nm]. Zakres ten nazywany jest promieniowaniem optycznym. Duże znaczenie w odbiorze bodźców wzrokowych ma kontrast oświetlenia. Gdy kolory są do siebie podobne ludzkie oko ma kłopot   
z szybką identyfikacją symboli i elementów. Dlatego dobór kolorystyczny symboliki   
w przypadku zobrazowania informacji powinien być wybrany tak, aby wykorzystywać najważniejsze kolory odróżniające się od tła. W powszechnym stopniu przyjęto   
w lotnictwie oznaczać kolorem czerwonym tylko najpilniejsze i zagrażające dalszemu lotowi sytuacje awaryjne, kolorem bursztynowym informacje ostrzegawcze, które są niebezpieczne, jednak pozwalają na kontynuowanie lotu, natomiast innymi kolorami jak np. niebieskim czy białym każde inne czynniki niewymagające podjęcia natychmiastowych działań. Pilot obserwując wskazanie jest w stanie zareagować   
w sposób intuicyjny, gdyż specjalnie dobrana symbolika i kolorystyka umożliwia mu takie działanie bez szczegółowego zagłębiania się w naturę i istotę problemu. Analizując interakcję pilot – maszyna, można zestawić zbiór wymagań jakie powinien spełniać system zobrazowania informacji, aby odbiór danych przez człowieka był jak najbardziej efektywny. Do najważniejszych kryteriów wpływających na powyższą relację należą:

- liczba wskaźników  
 - sposób kodowania informacji  
 - poziom sygnału  
 - klarowność informacji  
 - rozmieszczenie fizyczne wskaźników  
 - ciągłość wskazań  
- stopień obciążenia pamięciowego załogi

Liczba wskaźników umieszczonych w kokpicie powinna być minimalna. Wynika to z ograniczeń pamięciowych jakie posiada człowiek oraz jego podzielności uwagi. Rozproszone na wiele przyrządów wskazania są trudne do precyzyjnego określenia   
w danej chwili. Czas, jaki należy poświęcić na poprawne zaobserwowanie, przeanalizowanie i wypracowanie decyzji dla danego wskazania, wydłuża się drastycznie dla większej liczby przyrządów. Dlatego też w lotnictwie daje się zaobserwować trend coraz większej integracji informacji, która może pochodzić   
z różnych źródeł i podsystemów. Cecha ta jest charakterystyczna dla współczesnych myśliwców i maszyn bojowych. Starsze analogowe wskaźniki, występujące nierzadko w dużej liczbie, zmieniane są na rzecz nowoczesnych monitorów na których pilot ma podgląd bieżącej sytuacji w powietrzu i wszystkich interesujących go informacji, a co najważniejsze dostępnych w jednym miejscu.

Kodowanie informacji wiąże się z możliwościami psychofizycznymi pilota operatora. Sposób w jaki informacja zostanie przedstawiona (kodowanie) odgrywa bardzo istotną rolę. Na podstawie badań dobierany jest kształt oraz rozmiary tarcz, skale podziałek, wielkość symboli na wielofunkcyjnych ekranach oraz liczba elementów dodatkowych. Łatwiejsze i bardziej intuicyjne za sprawą ciągłej widoczności całej podziałki, wydają się być wskaźniki o tarczach okrągłych. Okrągłe tarcze implementowane są także we wskaźnikach ekranowych.

Poziom sygnału odgrywa znaczną rolę jako cecha determinująca oddziaływanie wskazywanej informacji na receptory człowieka. Powinien on być dobrany odpowiednio, tj. powinien przekraczać próg czułości wykrycia przez użytkownika,   
a jednocześnie nie być zbyt silny i intensywny, aby niepotrzebnie nie odciągać uwagi lub wręcz przeszkadzać w prowadzeniu operacji pilotażu. To uwarunkowuje wielkości tarcz wskaźników analogowych, wymiary podziałki, długości i grubości igieł, podświetlenie przyrządu lub w przypadku wskaźników ekranowych, jasność monitora, nasycenie i inne właściwości związane z siłą oddziaływania na receptory człowieka.

Klarowność informacji polega na tym, że informacja powinna w sposób ściśle precyzyjny i dyrektywny określać ciąg następnych czynności, jakie musi podjąć użytkownik. Rozwiązanie takie skraca czas potrzebny na wypracowanie decyzji odciążając z tego obowiązku użytkownika. Przykładem może być wskaźnik do systemu lądowania ILS, gdzie użytkownik za pomocą symboliki ruchomych belek na bieżąco informowany jest o niezbędnych manewrach zapewniających właściwe podejście do lądowania.

Fizyczne rozmieszczenie wskaźników i przyrządów zależy od funkcji pełnionej przez przyrząd. Najbardziej istotne wskazania wymagają ciągłości obserwacji przez pilota i powinny znajdować się w strefie najszybszego kontaktu wzrokowego, tak aby pilot mógł w możliwie krótkim czasie zorientować się o swojej sytuacji przestrzenno-położeniowej. Nieco mniej istotne wskazania jak temperatury zespołu napędowego lub ciśnienia w poszczególnych instalacjach mogą być usytuowane w mniej widocznym miejscu, gdyż dostęp do nich nie jest niezbędny w trakcie trwania całego lotu. Najnowszymi rozwiązaniami są wskaźniki przezierne i nahełmowe w których pilot nie musi w ogóle odrywać wzroku z obserwowanego punktu otoczenia SP mając jednocześnie na bieżąco podgląd parametrów lotu.

Inną ważną cechą systemów zobrazowania informacji jest ciągłość wskazań. Jest to zagadnienie bardzo złożone, obejmujące szereg metod z zakresu probabilistyki   
i prawdopodobieństwa, mających na celu przewidywanie niektórych istotnych parametrów co bardzo ułatwia pracę pilota. Przykładem takiego rozwiązania są systemy wspomagające prowadzenie walki powietrznej, które muszą stosować metody probabilistyczne aby w sposób ciągły śledzić zmieniający swoje położenie cel. Ciągłość wskazań odnosi się również do niezawodności systemu i poprawności jego wskazań.

Załoga SP musi zostać odpowiednio przeszkolona i posiadać kwalifikacje uprawniające ją do bezpiecznego prowadzenia działań operacyjnych. Wiąże się to   
z dużą ilością niezbędnej dokumentacji, którą należy przyswoić. Celem tej charakterystyki systemu zobrazowania informacji jest częściowe lub całkowite odciążenie załogi od konieczności podejmowania decyzji na podstawie pamięci. W tym celu stosowane są wskaźniki posiadające np. zaznaczone wartości parametrów granicznych w postaci kolorowych skali i symboliki.

Z powyższych zależności wynika, że system zobrazowania informacji powinien spełniać szereg właściwości i kryteriów, dzięki którym zostaje zaprojektowany   
i wdrożony do użytkowania. W dużym stopniu wpływ na jego postać mają bezpośrednie badania laboratoryjne, które określają jakie ściśle właściwości są najkorzystniejsze,   
a jakich należy się przy projektowaniu takich systemów wystrzegać.

* 1. **Analiza wybranych przyrządów i wskaźników**

Współczesne trendy wśród maszyn bojowych wykazują tendencję do zmiany tradycyjnych mechanicznych lub mechaniczno-elektrycznych wskaźników na rzecz rozwiązań bardziej nowoczesnych w postaci wskaźników przeziernych, nahełmowych lub ekranowych. Jednak stale w użyciu znajdują się myśliwce bojowe na których stosowane są oba wymienione rodzaje przyrządów, głównie ze względów bezpieczeństwa i niezawodności. Przykładem jest samolot wielozadaniowy F-16   
w którym, oprócz nowoczesnej techniki zobrazowania informacji można znaleźć tradycyjne przyrządy jak prędkościomierz, wysokościomierz, wskaźnik kursu albo przepływomierz. W dalszej części podrozdziału przeanalizowano podstawowe rodzaje przyrządów lotniczych z którymi można zetknąć się na współczesnych wojskowych statkach powietrznych.

### 1.3.1 Tradycyjne wskaźniki mechaniczne i elektromechaniczne

Wskaźniki mechaniczne i elektromechaniczne przez lata były podstawą   
w funkcjonowaniu systemów zobrazowania informacji. Ich popularność wynika w dużym stopniu ze stosunkowo prostej konstrukcji, wysokiego wskaźnika niezawodności oraz prostego procesu technologicznego. Mimo, że jest to rozwiązanie dość stare to nadal wykorzystywane na współczesnych wojskowych SP.

Wskaźniki tego typu posiadają tor pomiarowy w skład którego wchodzą elementy takie jak czujnik, zespół przełożeń, tor przesyłowy i element zobrazowania wskazania. Ponadto w przypadku wskaźników elektromechanicznych do powyższych elementów dochodzi przetwornik mechanoelektryczny zamieniający bezpośrednio mierzoną wielkość fizyczną na jej elektryczny ekwiwalent. Przykładami takiego typu wskaźników są wysokościomierze barometryczne, prędkościomierze, wskaźniki temperatury, zakrętomierze i chyłomierze poprzeczne itp. Rysunek 1.2 przedstawia wygląd wskaźnika wysokościomierza barometrycznego.

  
Rys 1.2 Wskaźnik wysokościomierza barometrycznego

Wskaźniki podobne do zaprezentowanego stosowane są na wielu samolotach zarówno wojskowych jak i cywilnych. Cechą charakterystyczną dla tego typu wskaźników jest to, że bezpośrednio obrazują ściśle określony rodzaj informacji. Najczęstszym rozwiązaniem jest tarcza z naniesioną podziałką o kształcie okrągłym,   
z wewnętrznymi znacznikami w postaci cyfr. Wartość parametru wskazuje igła lub więcej igieł różnej długości, które wykonują pełny ruch obrotowy bez ustalonego zakresu. Przyrządy mechaniczne jak i mechanoelektryczne wyposażone są często   
w dodatkowe pokrętła umożliwiające ich kalibrację i dostrojenie do danych warunków otoczenia. W wysokościomierzu barometrycznym częstym elementem dostrajającym jest pokrętło ustawienia ciśnienia odniesienia powiązane ze swoją podziałką obrazującą ustawioną wartość ciśnienia np. 780 [mmHG] lub 1027 [hPa].

Innym typem wskaźników mechaniczno-elektrycznych są wskaźniki symboliczne. Są to różnego rodzaju oznaczenia w postaci symboli wyświetlanych w momencie zaistnienia określonego stanu. Przykładami są lampki ostrzegawcze i informacyjne np. wysuniętego podwozia, zaciągniętego hamulca parkingowego lub informujące   
o podłączonym lotniskowym źródle zasilania energią elektryczną.

### 1.3.2. Wskaźniki zintegrowane i dyspozycyjne

Obok wskaźników mechanicznych i elektroniczno-mechanicznych inną grupę stanowią wskaźniki zintegrowane i dyspozycyjne. Geneza ich powstania związana jest ze sposobem ewolucji przeprowadzania operacji lotniczych. W szczególności na wojskowych statkach powietrznych pilot, aby prawidłowo wykonać zadanie musi korzystać z różnych źródeł informacji. Ta różnorodność objawia się zwiększającą się ilością wskaźników i przyrządów, a co za tym idzie wydłużonym czasem reagowania pilota – operatora. Aby wyeliminować ten problem wprowadzono integrację wskazań dla kilku parametrów lotu na jednym wskaźniku dzięki czemu do wykonania zadania nie jest konieczne przenoszenie uwagi na kolejne przyrządy. Wskaźniki dyspozycyjne umożliwiają zobrazowanie nakazanej wartości parametru. Nie są biernymi wskazaniami – umożliwiają manipulację tymi wartościami.

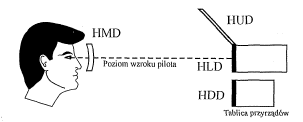
W skład tej grupy przyrządów wchodzą wskaźniki położenia przestrzennego ADI, sytuacji horyzontalnej HSI, wskaźnik kontroli zespołu napędowego, elektroniczny EADI oraz elektroniczny EHSI. Rysunek 1.3 przedstawia widok kabiny samolotu Su-27. W centralnej części panelu znajduje się wskaźnik położenia przestrzennego   
i sytuacji horyzontalnej. Za nimi znajdują się znaczniki umożliwiające nastawienie pożądanego kierunku kursu drogi oraz znaczniki ruchomych belek mające na celu zobrazowanie właściwego podejścia do lądowania, stąd wskaźnik ten można kwalifikować również jako dyspozycyjny. Innymi samolotami bojowymi wykorzystującymi tego typu zobrazowanie informacji są F-16, Mig-29 lub F-15 Eagle.

  
Rys. 1.3 Widok panelu w kabinie samolotu Su-27

### 1.3.3. Wskaźniki elektroniczne

Dynamiczny rozwój elektroniki od lat 80 XX wieku przyczynił się w dużym stopniu do coraz powszechniejszego stosowania wskaźników elektronicznych. Początkowo, trudne technologicznie do wykonania i często zawodne elementy elektroniczne, obecnie są tanie, łatwe do wykonania i bardziej niezawodne dzięki wysokiej klasy komputerom i algorytmom zabezpieczającym działanie tego typu urządzeń. Ponadto wskaźniki elektroniczne pozwalają na olbrzymią integrację różnego rodzaju parametrów i wskazań w jednym miejscu co ma bardzo istotne znaczenie   
z punktu widzenia użytkownika. Większość systemów należących do tej kategorii cechuje mała masa, łatwość obsługi i serwisowania, możliwość stosowania różnych standardów komunikacji pomiędzy poszczególnymi urządzeniami oraz możliwość dostosowywania sposobu pracy i wskazania przez pilota nawet w trakcie użytkowania. Wśród wskaźników elektronicznych można wyróżnić:

- wskaźniki przezierne typu HUD (Head Up Display) znajdujące się minimalnie powyżej poziomu wzroku pilota, umożliwiając jednoczesną obserwację przestrzeni przed SP oraz parametrów wskazywanych przez wskaźnik,  
 - wskaźniki kolimowane HLD (Head Level Display) znajdujące się na wysokości wzroku pilota,  
 - wskaźniki monitorowe HDD (Head Down Display) znajdujące się poniżej wzroku najczęściej w postaci ekranów wielofunkcyjnych MFD,  
- wskaźniki HMD (Helmet Mounted Display) będące hełmowymi wskaźnikami lub celownikami dzięki którym pilot może w dowolnym momencie.

  
Rys 1.4 rodzaje wskaźników

Wskaźnik HUD (Rys. 1.4) umiejscowiony jest najczęściej w górnej części tablicy przyrządów. Składa się z lampy elektropromieniowej, układu optycznego w postaci soczewek i luster oraz przeźroczystego ekranu. Wiązka generowana przez lampę zostaje skierowana przy użyciu układu optycznego na ekran co umożliwia zogniskowanie promieni świetlnych w nieskończoności i uzyskanie równoległej wiązki. Najczęściej wskaźniki HUD składają się z modułu zasadniczego (tzw. głowicy urządzenia), panelu sterującego zawierającego szereg przycisków i przełączników, pozostałych urządzeń jak zasilacz, wbudowane układy diagnostyczne i inne. Głowica zawiera dodatkowe elementy w postaci pokręteł umiejscowione przy ekranie przeźroczystym, umożliwiające dostosowanie jego położenia do preferencji użytkownika.

Najważniejszymi parametrami HUD są pole widzenia (FOV) oraz zdolność do przepuszczania światła. Ponadto ważnym parametrem jest kąt widzenia dostępny dla pilota bez konieczności wykonania ruchu głową (IFOV). Kąt ten zmniejsza się wraz ze zwiększaniem się dystansu od układu optycznego, natomiast ulega zwiększeniu wraz ze wzrostem średnicy soczewek lub wiązki z lampy CRT. Z tego też powodu odległość pomiędzy ekranem, a pilotem powinna być jak najmniejsza. Parametr ten (odległość) nie może być jednak swobodnie zmieniany gdyż wpływ na niego mają inne uwarunkowania jak np. bezpieczeństwo awaryjnego opuszczania SP.

Najważniejsze parametry, które zobrazowane są na wskaźniku HUD to podstawowe parametry lotu jak wysokość, prędkość TAS, kurs, kąt pochylenia samolotu oraz wysokość radiowa. Ponadto obok standardowych wskazań bardzo często na HUD wyświetlane są ostrzeżenia o sytuacjach awaryjnych, elementy wspomagające prowadzenie działań bojowych czy wskazania dotyczące stanu samolotu.

Niewątpliwie wskaźniki przezierne niosą ze sobą olbrzymie korzyści i zalety wynikające z ich stosowania. Pilot może skupić się na wykonywanych czynnościach i sytuacji w powietrzu mając jednocześnie stały podgląd niezbędnych danych i informacji. Zwiększa to bezpieczeństwo lotu oraz wpływa na efektywność pracy wykonywanej przez pilota.

Wskaźniki kolimowane HLD pełnią bardzo podobną funkcję do wskaźników HUD. Znajdują się w optymalnym położeniu z punktu widzenia pilota umożliwiając bardzo krótki czas przenoszenia wzroku z innych przyrządów. Miejsce ich położenia w kabinie znajduje się tuż pod wskaźnikiem HUD co umożliwia błyskawiczne przenoszenie wzroku z jednego przyrządu na drugi bez konieczności akomodacji oka. Głowa pilota może znajdować się nadal w tym samym położeniu, a odczyt potrzebnej informacji następuje w wyniku ruchu gałek ocznych, a nie głowy. Z tego też powodu wskaźniki HLD odgrywają istotną rolę np. podczas prowadzenia działań bojowych, gdzie ułamki sekund mają bardzo duże znaczenie. W tego typu wyświetlaczach zobrazowane są najczęściej: aktualna sytuacja prowadzonych działań bojowych, mapa i zobrazowanie pozycyjne lub dane radarowe. Przykład zastosowania tego typu wskazań to samolot Rafale.

Wskaźniki monitorowe są kolejną grupą wskaźników elektronicznych. Stały się bardzo popularne, szczególnie na wojskowych samolotach bojowych wypierając tradycyjne wskaźniki. Najczęściej miejscem ich rozmieszczenia są tablice przyrządowe. Pociąga to za sobą pewne skutki, np. w przypadku przenoszenia wzroku z innych wskaźników znajdujących się na linii wzroku oko musi przejść proces akomodacji co wiążę się z pewnym czasem.

Wskaźniki HDD cechuje niewielka masa i rozmiary w porównaniu do tradycyjnych przyrządów. Są bezkonkurencyjne jeśli chodzi o integrację informacji i sposób jej prezentacji. Dzięki stosowanym komputerom graficznym (generatory symboli) możliwe jest w zasadzie dowolne zobrazowanie informacji. Często wskaźniki ekranowe występują w formie ekranów wielofunkcyjnych MFD (Multi Funcion Diplay), na których pilot ma do dyspozycji dane nawigacyjne, sterowanie systemem uzbrojenia, podgląd sytuacji radarowej, podglądy z kamer termowizyjnych i inne specyficzne wskazania. W większości przypadków możliwe jest dobranie sposobu prezentacji symboli i parametrów, a także zaprogramowanie skrótów do najczęstszych zakładek poszczególnych podstron menu głównego. Na rysunku 1.5 pokazano wygląd kokpitu nowoczesnego wojskowego samolotu F-22 Raptor. Widać na nim dwa ciekłokrystaliczne ekrany spełniające przytoczone funkcje.

  
Rys 1.5 Ekrany MFD samolotu F-22 Raptor

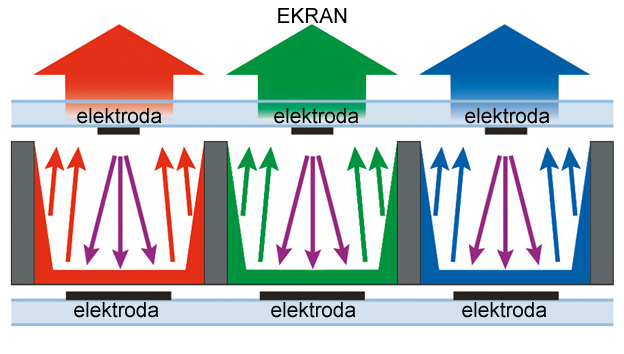
Najbardziej istotną częścią wskaźników HDD jest matryca, z której wykonano ekran. Ze względu na zastosowaną technologię ekrany te możemy podzielić na:

- z lampą kineskopową CRT,  
- ciekłokrystaliczne LCD,  
- plazmowe PDP,  
- diody elektroluminescencyjne LED.

Najstarszą grupę stanowią ekrany oparte na tradycyjnej technologii lampy kineskopowej CRT. Mają dużą masę i wymiary. Metoda ich działania polega na generowaniu wiązki elektronów, która skierowana polem elektromagnetycznym uderza w matrycę pokrytą luminoforem. Tego typu urządzenia pracują w trybie monochromatycznym tj. odzwierciedlają jedynie jeden kolor wraz z jego stopniem nasycenia (intensywności). Częściej spotykane są ekrany o trzech działkach elektronowych w których każda wiązka odpowiada za nasycenie koloru czerwonego, zielonego i niebieskiego (RGB). Matryca w takim przypadku nie jest pokrywana   
w całości substancją luminoforu lecz stanowi zbiór punktów dla każdego rodzaju wiązki. Wrażenie świecenia wynika z ciągłego odświeżania obrazu. Wiązka przemieszcza się po matrycy w sposób swobodny lub zdeterminowany (metoda rastrowa i liniowa).

Nowocześniejszą odmianą wskaźników ekranowych stanowią matryce ciekłokrystaliczne LCD. Technika generowania obrazu polega na przepuszczaniu światła przez warstwę ciekłych kryształów, które zmieniają jego kierunek doprowadzając do całkowitego lub częściowego przepuszczania światła lub jego braku. Matryce z tego gatunku można podzielić na pasywne zawierające elektrody na podłożach elementów ciekłokrystalicznych lub aktywne, które posiadają cienkowarstwowe tranzystory przełączające (TFT). Matryce tego typu umożliwiają uzyskiwanie dużych rozdzielczości obrazu i dobrych kątów widzenia.

Zupełnie inaczej działają matryce plazmowe, które do uzyskania efektu świecenia wykorzystują efekt działania lampy fluorescencyjnej. W przeciwieństwie do pozostałych typów wyświetlaczy same generują fotony (światło). Panele składają się   
z bardzo wielu komórek wypełnionych gazami szlachetnymi do których podłączone są elektrody zasilające. Warstwa powierzchowna komórek pokryta jest luminoforem   
w kolorze czerwieni, zieleni lub niebieskim. Po doprowadzeniu napięcia do elektrod następuje wyładowanie w gazie, co generuje promieniowanie UV. Na skutek tego substancja – luminofor zaczyna emitować fotony. Rysnek1.6przedstawia zasadę działania tego typu ekranów

  
Rys 1.6 Ekran plazmowy

Wskaźniki zobrazowania na hełmach HMD mogą występować w różnej postaci. Spotykanym rozwiązaniem są matryce diodowe albo lampy elektropromieniowe wraz z zamontowanymi ekranami półprzepuszczalnymi. Układ generowania obrazu przesyła sygnał do specjalnie przygotowanego okularu, gdzie pilot widzi zobrazowanie na tle otoczenia. Często tylko jedno oko uczestniczy w projekcji obrazu, natomiast drugie pozostaje w interakcji z otoczeniem bez udziału wskaźnika. Przy dłuższych okresach użytkowania może to powodować zmęczenie wzroku. Dlatego konstrukcje wskaźników stereoskopowych jak również dwuokularowych są intensywnie badane i rozwijane. Ważnym zagadnieniem w przypadku tego typu wskaźników jest mechanizm ciągłego śledzenia położenia głowy pilota. Najczęściej odbywa się to poprzez sieć czujników rozmieszczonych w kabinie odbierających sygnały nadawane z hełmu. Sposoby realizacji tego zagadnienia:

- magnetyczne,  
 - optoelektroniczne,  
 - akustyczne,  
 - przez rozpoznawanie obrazów (pattern recognition).

Wskaźnik HMD oprócz powszechnego użycia we współczesnych bojowych SP znalazł również zastosowanie w modernizacji śmigłowca bojowego W-3WA.

* 1. **Rola oprogramowania w systemie zobrazowania informacji**

Coraz nowsze systemy zobrazowania informacji wymagają stosowania komputerów dużej mocy obliczeniowej, które są w stanie generować obrazy w czasie rzeczywistym. Istotną rolę odgrywa tutaj nie tylko sprzęt ale również oprogramowanie. Najpopularniejszymi językami do zastosowań aplikacji lotniczych są języki rodziny   
C oraz ADA. W przypadku wskaźników elektronicznych generowanie grafiki następuje dzięki wydajnym bibliotekom graficznym jak np. OpenGL, które udostępniają szereg funkcji i odciążają programistów z konieczności operowania na niskopoziomowym języku maszynowym. Wydajne i nowoczesne komputery tego typu posiadają specjalny dedykowany system kompatybilny z większością udostępnianych API. Oprogramowanie musi spełniać podobnie jak sprzęt szereg wymagań niezawodnościowych ujętych w normach takich jak DO-178B/ED-12B, czy DO-248B/ED-94B. W dokumentach znajduje się szereg definicji o kodzie źródłowym   
i obiektowym, wskazane działania programistyczne, krytyczne błędy oraz odpowiednie i zalecane procedury testowania aplikacji.

* 1. **Tendencje i kierunki rozwojowe systemów zobrazowania informacji**

Na przestrzeni kilkudziesięciu lat daje się zaobserwować pewien trend i kierunek, w którym zmierzają systemy zobrazowania informacji, szczególnie w przypadku zastosowań wojskowych. Następuje coraz większa integracja wskazań, a same urządzenia dzięki postępowi technologicznemu stają się lżejsze i mają mniejsze rozmiary. Bardzo popularne stało się określenie „Glass Cockpit” odnoszące się do powszechnego stosowania wielofunkcyjnych wskaźników ekranowych stopniowo wypierających standardowe mechaniczne i elektryczno-mechaniczne przyrządy. Dzięki temu pilot ma do dyspozycji w jednym miejscu wiele informacji pochodzących   
z różnych systemów np. uzbrojenia czy nawigacyjnych. Nie musi przy tym przenosić wzroku na poszczególne przyrządy i oszczędza czas. Coraz większe znaczenie nabierają nahełmowe wskaźniki w postaci dwuokularowych urządzeń montowanych na hełmy lub w postaci gogli. W coraz większym zakresie to pilot decyduje o tym jak informacja ma zostać mu przekazana i sam dostosowuje tryby pracy wskaźnika. Rysunek 1.7 przedstawia wnętrze kokpitu najnowocześniejszego myśliwca F-35. Widać na nim, że tradycyjny panel wskaźników został niemal całkowicie wyparty przez ekrany, na których pilot w każdej chwili może otrzymać wybraną przez siebie informację.

 *Rys. 1.7 Futurystyczne wnętrze kokpitu myśliwca F-35*

* 1. ANALIZA BUDOWY I ZASADY DZIAŁANIA SYSTEMU ZOBRAZOWANIA INFORMACJI ORAZ RODZAJU WYŚWIETLANYCH INFORMACJI NA WIELOFUNKCYJNYCH WYŚWIETLACZACH MFD SAMOLOTU F16

W poniższym rozdziale omówiono szereg podstawowych zagadnień dotyczących budowy i zasady działania systemu zobrazowania informacji samolotu odrzutowego F16. Wyjaśniono jakie komponenty wchodzą w skład systemu oraz ich znaczenie dla pilota. Przedstawiono również rodzaje interfejsów służących do przesyłania danych pomiędzy urządzeniami wchodzącymi w skład tego systemu, jego architekturę oraz główną ideę działania profilów misji samolotu, tzw. „Master Mode”, w których to funkcjonowanie poszczególnych podsystemów oraz rodzaj informacji dostarczanej pilotowi zmienia się w zależności od aktywnego profilu. W drugiej części tego rozdziału omówiono działanie wskaźników wielofunkcyjnych MFD. Przedstawiono wyczerpujący opis poszczególnych stron wchodzących w skład menu głównego wskaźnika, wyjaśniono ich związek z innymi komponentami różnych systemów zabudowanych na samolocie oraz opisano zastosowaną symbolikę. Celem rozdziału jest szczegółowe zaznajomienie się z zasadą funkcjonowania całego systemu zobrazowania informacji na samolocie F16 oraz poznanie funkcjonowania wskaźników MFD.

## **2.1 Ogólna koncepcja budowy systemu zobrazowania informacji samolotu F16**

System zobrazowania informacji samolotu odrzutowego F16 składa się z kilku odrębnych komponentów takich jak urządzenia wskaźników i przyrządów umieszczone w kabinie, komputery sterujące pracą podsystemów, kontrolujące przesyłanie informacji pomiędzy poszczególnymi urządzeniami, generatory symboli oraz interfejs. Wskaźniki i przyrządy pokładowe zastosowane w samolocie wielozadaniowym F16 występują w formie tradycyjnych mechaniczno-elektrycznych wskaźników takich jak wskaźnik kursu, położenia przestrzennego, natężenia przepływu paliwa czy prędkości obrotowej wału silnika. Znalazły one zastosowanie głównie dzięki swojej wysokiej niezawodności i prostoty konstrukcji. W samolocie F16 pełnią one jednak marginalną rolę – wykorzystywane są jedynie w sytuacjach awaryjnych oraz w trakcie procedur rozruchu silnika, procedur diagnostyczno-naprawczych i innych czynności zajmujących niewielki odsetek czasu w odniesieniu do trwania całej misji. Podobną funkcję pełnią wskaźniki w formie lampek sygnalizacyjnych informujących pilota o zaistniałych awariach oraz ostrzegające go w przypadku nie wykonania jakiejś czynności. Niemal cała działalność operacyjna pilota opiera się na bardziej zaawansowanych technicznie przyrządach jak wskaźnik HUD oraz ekrany wielofunkcyjne MFD. Wskaźnik HUD znajduje się na wysokości oczu pilota i pełni najważniejszą rolę podczas prowadzenia działań bojowych. Pilot ma możliwość szybkiego podglądu sytuacji przestrzennej w jakiej znajduje się płatowiec tj. wysokości barometrycznej, prędkości IAS, wysokości mierzonej radiowysokościomierzem, ułożenia samolotu względem horyzontu, kątów pochylenia i przechylenia, aktualnego i zadanego kursu, wartości przyspieszeń liniowych, komunikatów w przypadku zaistnienia sytuacji awaryjnych oraz szeregu innych informacji zależnych od aktualnie wybranego profilu „Master Mode”. Wskaźnik HUD pełni bardzo ważną rolę gdyż pozwala informować pilota o sytuacjach niebezpiecznych bez konieczności odrywania wzroku skierowanego do przodu wzdłuż osi podłużnej samolotu i obserwacji przestrzeni przed płatowcem. Więcej informacji o wskaźnikach przeziernych typu HUD zawiera rozdział I. Innymi ważnymi urządzeniami wchodzącymi w skład systemu zobrazowania informacji samolotu F16, powszechnie wykorzystywanymi podczas trwania misji są wskaźniki wielofunkcyjne MFD. Zbudowane są w postaci ramki z przyciskami OBS oraz ekranu. Pilot może wchodzić w interakcję ze wskaźnikiem poprzez wybór odpowiedniego przycisku. Wskaźnik MFD wyświetla menu główne oraz kilkanaście menu w formie przełączalnych zakładek. Znajdują się na nich informacje nawigacyjne, informacje pochodzące z systemu uzbrojenia, informacje o sytuacji radarowej, podgląd z kamery na podczerwień, menu z ustawieniami i wiele innych, szerzej omówionych w dalszej części tego rozdziału. Wskaźniki MFD różnią się rodzajem wyświetlanej informacji w zależności od tego jaki jest ustawiony profil Master Mode. W niektórych przypadkach różnice są nieznaczne np. rozmieszczenie elementów i symboli, ale w niektórych sytuacjach traci się funkcjonalność poszczególnych podsystemów. Pilot ma duży wpływ na charakter wyświetlanej informacji na wskaźnikach MFD dzięki możliwości zaprogramowania 3 zakładek i przypisania ich do jednego z trzech klawiszy OBS, co w efekcie daje możliwość szybkiego przełączania się pomiędzy wybranymi zakładkami.

## 2.2. Profile misji i różnice w działaniu systemów

Statek powietrzny F16 ze względu na cechę wielozadaniowości może prowadzić operację w kilku możliwych wariantach. Poszczególne warianty nazywane także trybami pracy (Master Mode) mogą zostać ustawione w trakcie przygotowania do misji. Dostępnymi profilami misji są:

- A-A – operacje powietrze – powietrze (ang. „Air to Air” )   
 - A-G – operacje powietrze –ziemia (ang. „Air to Ground”)  
 - NAV – operacje nawigacyjne (domyślnie ustawiane gdy nie wybrano innych)  
 - DGFT – operacje walki manewrowej (ang. „Dogfight”)  
 - MSR OVRD – operacje typu kombinowane (ang. „Missile Override”)  
 - Emergency Jettison – awaryjny zrzut uzbrojenia  
 - Selective Jettison – awaryjny selektywny zrzut uzbrojenia

Wybór poszczególnych trybów Master Mode wpływa znacząco na awionikę płatowca jak również wygląd i rodzaj wyświetlanych informacji na wskaźnikach. Każdy z trybów (za wyjątkiem S/J) może zostać zaprogramowany przed misją i następnie załadowany do komputera za pomocą DTC. Również pilot ma możliwość manualnego przełączenia pomiędzy poszczególnymi profilami wedle własnego uznania, poprzez wybór odpowiedniego przycisku w systemie HOTAS.

Tryb NAV jest domyślnym trybem nawigacyjnym dostępnym zawsze gdy żaden inny tryb nie został wybrany. Zapewnia on informacje nawigacyjne o punktach trasy oraz pozycji SP. Tryb NAV może zostać wybrany manualnie w przypadku aktywnego innego trybu np. AA lub AG poprzez naciśnięcie przycisku na konsoli ICP.

Tryb A-A zapewnia wsparcie dla działań operacyjnych typu powietrze-powietrze wykorzystując pociski średniego i krótkiego zasięgu oraz działko. Dostępny jest po wybraniu z konsoli ICP przycisku A-A .

Tryb MSL OVRD pozwala pilotowi na gwałtowne przekonfigurowanie systemu uzbrojenia pod kątem zdolności operacyjnych A-A z dowolnego innego trybu Master Mode. W trybie tym możliwe są dowolne kombinacje uzbrojenia co stanowi jego zaletę. Wybór trybu MSL OVRD następuje z poziomu drążka kontroli zespołu napędowego, poprzez wybór pozycji MSL OVRD z przełącznika DGFT/MSL OVRD.

Tryb DGFT jest bardzo zbliżony do trybu MSL OVRD i również pozwala na szeroki wariant uzbrojenia A-A. Różnice pomiędzy tymi dwoma trybami tkwią w zastosowanej wizualizacji na ekranach oraz domyślym ustawieniu radaru (CRM dla MSL OVRD, ACM dla DGFT). Wybór tego trybu możliwy jest po wybraniu pozycji DGFT na przełączniku DGFT/MSL OVRD.

Tryb A-G pozwala na wybór zasobów uzbrojenia typu powietrze – ziemia i wspomaganie prowadzenia działań bojowych wymierzonych w cele naziemne. Wybór trybu A-G następuje z pozycji konsoli ICP poprzez naciśnięcie przycisku A-G.

Tryb Selective-Jettison pozwala na bezpieczne zrzucenie uzbrojenia będącego w stanie nieuzbrojonym, tzn. pozbycia się wybranego rodzaju podwieszenia bez ryzyka zadziałania środka bojowego. Podobnie działa Emergency-Jettison lecz w tym przypadku zrzucany jest nie wybrany selektywnie środek bojowy lecz wszystkie możliwe do zrzutu podwieszenia.

## 2.2. Wyświetlacze MFD

Wyświetlacze wielofunkcyjne MFD występują w formie dwóch kolorowych ciekłokrystalicznych ekranów LCD o rozmiarze 4x4” wraz z ramką zawierającą 20 przycisków do wyboru poszczególnych plansz (OBS) menu oraz cztery funkcyjne trójstanowe przyciski do regulacji parametrów wyświetlania (jak kontrast, natężenie jasności wyświetlanych symboli oraz tekstu). Trzy środkowe przyciski rozmieszczone w dolnej części ramki pozwalają uzyskać bezpośredni dostęp do wybranych przez pilota plansz.

Informacja jaka jest prezentowana pilotowi zależy m. in. od trybu pracy w jakim znajduje się system (Air to Ground, NAV, Air to Air itp.) oraz aktualnie wybranej planszy menu. Wybór poszczególnych menu składa się z wyjścia do menu głównego MFD (poprzez naciśnięcie przycisku obok nazwy aktualnie wyświetlanej planszy znajdującej się w dolnej części ekranu) oraz wyboru żądanego menu poprzez naciśnięcie przycisku znajdującego się obok nazwy planszy. Możliwy jest również bezpośredni dostęp do trzech wybranych plansz poprzez wciśnięcie jednego z trzech przycisków w dolnej części ekranu (OBS #12-14).

Z uwagi na możliwość sterowania w systemie HOTAS, ekrany, a właściwie poszczególne plansze mogą znajdować się w tzw. stanie SOI (Sensor of Interest). Wówczas pilot ma możliwość wyboru dostępnych na wyświetlaczu opcji za pomocą przycisków funkcyjnych znajdujących się na joysticku, nie integrując w działanie innych urządzeń.

Charakterystyczną rolę w wyświetlaczach MFD pełni przycisk OBS #15, który niezależnie od aktualnie wybranej planszy ma oznaczenie SWAP i służy do zamiany wyświetlanego kontekstu pomiędzy lewym i prawym MFD.

 *Rys. 2.1 Menu główne wskaźnika MFD*

### 2.2.1. Menu HSD

Menu HSD służy do wyświetlania informacji nawigacyjnych. Pilot ma do dyspozycji podgląd aktualnego położenia samolotu z pozycji lotu ptaka wraz z informacjami o zasięgu i kierunku promieniowania anteny radaru (tzw. stożek radaru), pierścieniach stanowiących okręgi o stałej średnicy (odległości), pozycji kursora radarowego, systemu nawigacyjnego INS wraz z punktami trasy (tzw. waypointy), dodatkowymi liniami zaprogramowanymi przed wylotem, naniesionymi zagrożeniami w postaci punktów pozycyjnych (np. obrony przeciwlotniczej, zgromadzonych oddziałów itp.) wraz z zaprogramowanymi przed wylotem ich zasięgami działania w postaci okręgów. Kolory okręgów stanowiących zasięg zagrożenia zmieniają się po wkroczeniu w jego obręb przez statek powietrzny.

Przyciski OBS #19 oraz #20 służą do zwiększania/zmniejszania zasięgu widoku HSD. Wówczas po zwiększeniu zasięgu widoku pilot widzi większą część przestrzeni otaczającej samolot i może podejmować odpowiednie manewry z dużym wyprzedzeniem czasowym.

Przycisk OBS #1 podpisany DEP służy do przełączania widoku między widokiem cofniętym (DEEP) gdzie symbol SP oraz pierścieni znajdują się poniżej środka ekranu (na wysokości ok. 1/3 wysokości ekranu), a widokiem centralnym (CEN) gdzie symbole są wyśrodkowane względem ekranu.

Przycisk OBS #2 umożliwia przełączanie pomiędzy widokiem DCPL (Decoupled) oraz CPL (Coupled), który powiązuje zasięgi widoku HSD oraz zasięg stożka radaru. W tym trybie oba zasięgi są odpowiednio dopasowane oraz brak jest dostępnych ustawień zasięgu widoku za pomocą przycisków OBS #19 i #20.

Przycisk OBS #5 umożliwia wejście w menu CNTL (Control) w którym dostępny jest szereg ustawień wyświetlania m. in.:

-FCR: wyświetla stożek radarowy oraz kursor radaru;  
 - PRE: wyświetla zaprogramowane waypointy oraz zagrożenia;  
 - AIFF: wyświetla odpowiedź na zapytanie swój – obcy;  
 - Line 1 -4: wyświetla linie DTC;  
 - Rings: wyświetla pierścienie otaczające symbol SP;

- ADLINK: wyświetla informacje Air to Air pomiędzy sojusznikami;  
 - GDLINK: wyświetla informacje Air to Ground pomiędzy sojusznikami;  
 - NAV 1,2,3: wyświetla plan lotu INS;

Wyjście z menu CTRL możliwe jest po ponownym naciśnięciu klawisza OBS #5.

Przycisk OBS #7 podpisany jako FZ umożliwia „zamrożenie” widoku otoczenia SP . W tym trybie pracy to symbol SP porusza się względem nieruchomego świata. Ponowne wciśnięcie OBS #7 powoduje powrót do normalnego trybu wyświetlania (nieruchomy SP oraz poruszające się otoczenie).

Poniżej przedstawiono widok planszy HSD (Rys 2.2a) oraz jego menu CTRL (Rys 2.2b).



Rys. 2.2 a – menu główne HSD, b – zakładka CNTL

### 2.2.2. Menu SMS

Menu SMS (Storage Management System) umożliwia dostęp do danych związanych z aktualnym uzbrojeniem zainstalowanym na SP, w postaci wykazu pocisków, zbiorników oraz zasobników podwieszonych na belkach i pylonach. Pilot poprzez system transferu danych (DTC) może zaprogramować jaki rodzaj podwieszenia aktualnie znajduje się na samolocie. Ponadto ma on dostęp do specyficznych dla danej kategorii uzbrojenia właściwości kontrolnych. Informacje wyświetlane w menu SMS ściśle zależą od profilu misji jaki został wybrany tzw. Master Mode.

W profilu o charakterze nawigacyjnym (NAV Master Mode) w menu SMS zostaną wyświetlone zasoby i stan uzbrojenia jakie znajdują się aktualnie na SP w postaci graficznej tekstowej. W miejscu reprezentującym podwieszenie wyświetlana jest nazwa zasobu, a w miejscu przycisku OBS #11 dostępne jest dodatkowe menu S/J (Selective Jettison). Pilot za pomocą tego przycisku może przełączać się pomiędzy główną stroną SMS, a dodatkową S/J, która została szerzej omówiona w dalszej części tego podrozdziału. Widok głównej zakładki SMS w trybie nawigacyjnym przedstawia rysunek 2.3:

  
Rys. 2.3 Widok menu SMS w trybie NAV

W trybie A-G (Air to Ground) główna zakładka SMS wyświetla tylko informacje przeznaczone dla broni typu powietrze-ziemia. Informacja wyświetlana na środku ekranu jest wówczas związana z aktualnym ustawieniem broni dostępnym po wybraniu zakładki CTRL (Control).

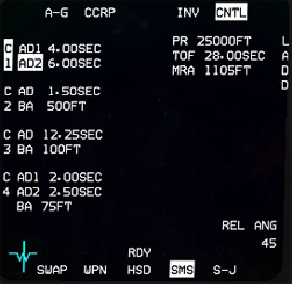
- OBS #1: wyświetla bieżący tryb Master Mode, a po wciśnięciu umożliwia ustawienie trybu broni strefowej (posiada odrębną zakładkę);  
 - OBS #2: jest używany do zmiany trybu dostarczania CCIP-CCRP-DTOS-LADD & MAN. Każdy z trybów posiada własną odrębną zakładkę;  
 - OBS #4: podpisany INV wyświetla zakładkę podwieszonych zasobów;  
 - OBS #5: jest to zakładka CNTL (Control) służąca do ustawień broni;  
 - OBS #6: wyświetla aktualnie aktywną broń A/G, po naciśnięciu przełącza aktywną broń A/G na następną;  
 - OBS #7: jest podpisany tak samo jak bieżący profil dla swobodnego zrzutu uzbrojenia A/G. Dostępne są do zaprogramowania dwa profile (PROF1 oraz PROF2);  
 - OBS #8: wyświetla wybrany tryb zrzucania uzbrojenia: tryb pojedynczy – oznaczony jako SGL umożliwia tradycyjny zrzut konkretnego uzbrojenia, tryb PAIR – umożliwia zrzucanie uzbrojenia parami;  
 - OBS #9:   
 - OBS #10:  
 - OBS #18: wyświetla tryby . Po naciśnięciu przełącza się pomiędzy trybami NOSE, TAIL oraz NSTL (NOSE + TAIL);

Rysunek 2.4 przedstawia widok zakładki SMS dla profilu A-G .

  
Rys. 2.4 Widok menu SMS w trybie A-G

Zakładka kontroli CNTL w trybie A-G, dostępna po naciśnięciu przycisku OBS #5 zawiera informacje na temat ustawień uzbrojenia A-G. Górna część ekranu (OBS #1-5) oraz dolna (OBS #11-15) pozostaje taka sama jak w przypadku głównej zakładki SMS. Zmianie ulegają informacje wyświetlane po lewej i prawej stronie ekranu. Pilot ma do dyspozycji pięć ustawień broni bombardierskiej (C1 do C4 oraz LADD).

- C1 są to kategorie broni przeznaczone do ogólnych zastosowań oraz bronie naprowadzane wiązką lasera. Zapewniają dwa rodzaje opóźnienia jeden dla przedniego zapalnika (NOSE) oraz drugi do tylnego (TAIL). Po naciśnięciu OBS #20 dostępna jest zakładka gdzie obydwa opóźnienia mogą zostać ustawione  
 - C2 jest przeznaczone dla jednostek zawierających wiele mniejszych bomb oraz dla wszystkich bomb detonujących się na określonej wysokości (BA – Burst Altitude). Naciśnięcie przycisku OBS #19 umożliwia dostęp do ustawień opóźnienia oraz wysokości BA.  
 - C3 jest dodatkowym ustawieniem dla bomb typu CBU – umożliwia ustawienie opóźnienia i wysokości eksplozji.  
 - C4 jest przeznaczone dla podwójnych bomb CBU i umożliwia ustawienie parametrów AD1, AD2 oraz BA.  
 - LADD   
 - OBS #10 jest przeznaczone dla ustawienia kąta zwalniania zasobów i używane przez komputer misji do obliczeń



Rys. 2.5 Zakładka CNTL w menu SMS

W przypadku wyboru menu SMS w trybie A-A podstawowa struktura i układ oznaczeń jest zbliżony do trybu A-G:

- OBS #1: wyświetla obecny tryb A-A i po naciśnięciu wybiera podstroję z ustawieniami działka  
- OBS #4: służy do wyboru zakładki z aktualnie zainstalowanym uzbrojeniem  
- OBS #5: jest to zakładka kontrolna (CTNL) dla aktualnie wybranej broni;  
- OBS #6: aktywna broń A-A. Po naciśnięciu przełącza aktywną broń na następną w kolejności;  
- OBS #8: aktywny tylko wtedy kiedy podwieszono broń naprowadzaną na promieniowanie podczerwone. Pozwala na ustawienie WARM lub COOL czujnika zainstalowanego na pocisku;  
- OBS #18: zależy od rodzaju załadowanej broni – dla broni naprowadzanych radarowo ustawia efektywny zasięg i rodzaj celu: LARGE, MEDIUM i SMALL.  
- OBS #19: pozwala na ustawienie trybu odpalania pocisku SLAVE albo BORE. W trybie SLAVE pocisk korzysta z radaru pokładowego (FCR), natomiast po wybraniu BORE po odłączeniu od SP przełącza się na swój własny wbudowany radar.



Rys. 2.5 Widok zakładki głównej menu SMS w trybie A-A

Zakładka Selective Jettison (S/J) dostępna jest we wszystkich trybach Master Mode. Pozwala pilotowi na kontrolowane wystrzelenie uzbrojenia pozostającego w stanie rozbrojonym (nieaktywnym) lub swobodny zrzut. Tylko niektóre rodzaje uzbrojenia dostępne są w tym trybie. Wybór konkretnej broni następuje po naciśnięciu odpowiedniego przycisku OBS znajdującego się obok pożądanej do zrzucenia broni. Wówczas nazwa broni zostaje podkreślona. Ponowne naciśnięcie tego samego przycisku OBS powoduje odrzucenie wyboru danej broni.

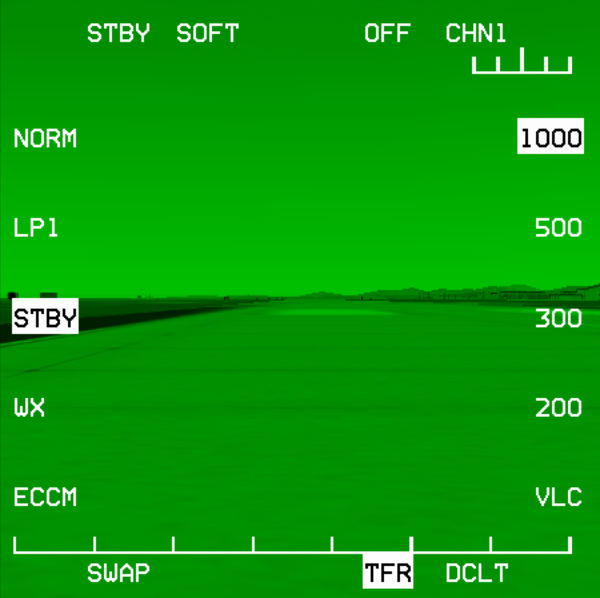
### 2.2.3. Menu TFR

Menu TFR (Terrain Following Radar) może zostać wybrane z menu głównego poprzez naciśnięcie przycisku OBS #17. Związane jest z radarami małego zasięgu (do 36000 ft) skierowanymi przed i w dół SP. Pozwalają one na wykonywanie lotów na bardzo małych wysokościach bez ryzyka zderzenia z terenem. Podsystem ten oferuje możliwość automatycznego poderwania samolotu w przypadku wysokiego prawdopodobieństwa kolizji, co odciąża pilota z konieczności ciągłego monitorowania wysokości i umożliwia skupienie się na innych czynnościach. TFR jest częścią systemu LANTIRN umożliwiającego podgląd przestrzeni przed SP w zakresie fal podczerwonych.

Zakładka TFR oferuje następujące możliwości funkcjonalne:

- OBS #1: wybór jednego z sześciu trybów operacyjnych: NORM, LPI, STBY, WX, ECCM oraz VLC. Najczęściej używanymi są NORM oraz STBY;  
- OBS #2: wybór stopnia agresywności podążania za terenem (Hard/Soft/Smooth);  
- OBS #4: służy do uruchomienia lub zablokowania systemu TFR;   
- OBS #5: bieżący kanał radaru;  
- OBS #6: ustawia wysokość 1000ft nad poziomem terenu ;  
- OBS #7: ustawia wysokość 500ft nad poziomem terenu;  
- OBS #8: ustawia wysokość 300ft nad poziomem terenu;  
- OBS #9: ustawia wysokość 200ft nad poziomem terenu;  
- OBS #10: ustawia bardzo niską wysokość VLC (tylko dla przelotów nad morzem lub bardzo płaskim terenem);  
- OBS #16: tryb kontroli emisji;  
- OBS #17: tryb ustawień pogodowych;  
- OBS #18: wybór trybu STBY;  
- OBS #20: wybór trybu NORM

W przypadku trybu NORM samolot jest automatycznie podrywany do bezpiecznej wysokości, a na wskaźniku HUD pojawia się informacja „OBSTACLE”. W przypadku trybu STBY samolot nie jest automatycznie podrywany, wyświetlana jest jedynie informacja „OBSTACLE” na wskaźniku HUD i wyświetlaczu MFD. Rysunek 2.6 przedstawia widok zakładki TFR

  
Rys. 2.6 Widok menu TFR

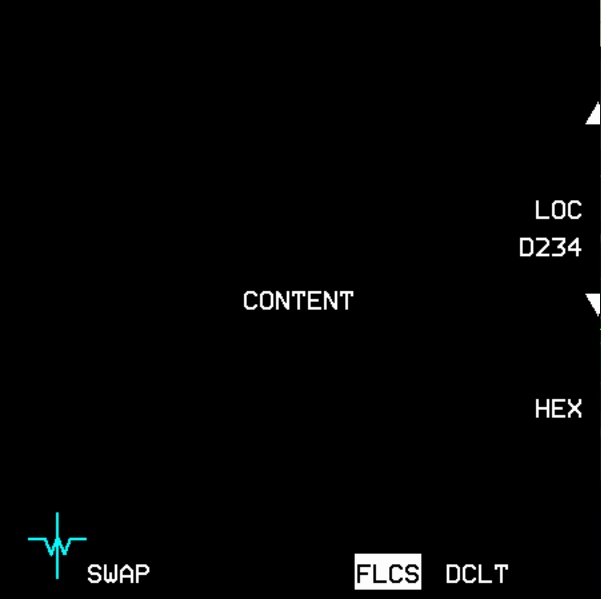
### 2.2.4. Menu DTE

Data Terminal Equipement jest kolejną zakładką dostępną z poziomu menu głównego wskaźnika MFD przy pomocy przycisku OBS #8. Jest używana do zarządzania danymi sporządzanymi podczas planowania misji i przesłania ew. transmisji tych danych do komputera misji MMC. Przesłanie danych następuje po naciśnięciu przycisku OBS #3. Podczas trwania tego procesu poszczególne nazwy systemów (FCR, DLINK, COMM itp.) są podświetlane. Rysunek 2.7 przedstawia wygląd menu DTE.

  
Rys. 2.7 Widok menu DTE

2.2.5. Menu FLCS

Celem tej zakładki jest wyświetlenie ustawień dotyczących systemu kontroli sterowania lotem (Flight Control System). Wygląd zakładki przedstawia rysunek 2.8:

  
Rys. 2.8 Widok menu FLCS

### 2.2.6. Menu FLIR

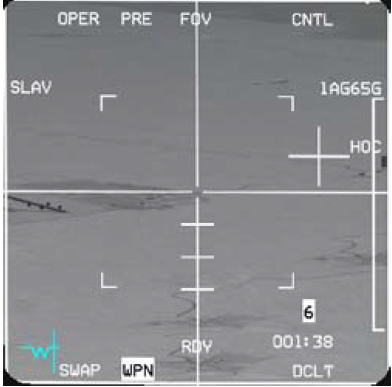
Menu FLIR (Forward Looking InfraRed) dostępne jest poprzez wybór przycisku OBS #16 z poziomu menu głównego wskaźnika MFD. Umożliwia podgląd przestrzeni przed samolotem w zakresie widma fal podczerwonych emitowanych przez różne obiekty. Podobnie jak w przypadku zakładki TFR, FLIR jest dostępny tylko po dołączeniu do samolotu zasobnika nawigacyjnego wyposażenia LANTIRN. Ponieważ zakładka TFR oferuje możliwości podglądu przestrzeni frontowej w zakresie widma podczerwonego, FLIR nie jest używane w nowszych wersjach samolotu F16 (Block 50/52).

  
Rys. 2.9 Menu FLIR – nie używane w wersji B50/52

### 2.2.7. Menu WPN

Menu WPN jest dostępne z poziomu menu głównego ekranu MFD poprzez przycisk OBS #18. Oferuje ono możliwość podglądu przez pilota w zakresie fal podczerwonych z czujników zamieszczonych na pokładzie lub wmontowanych w uzbrojenie. Pilot obserwując cel może „zablokować” określony obiekt i wówczas pocisk będzie nakierowywany na zaznaczony cel. Przykładowymi pociskami dostępnymi z wbudowanymi sensorami są pociski AGM-65 Mavericks i AGM-88 HARM. W tej zakładce dostępne są następujące opcje wyboru:

- OBS #1: status;  
- OBS #2: zmiana trybu PRE-VIS-BORE;  
- OBS #3: ustawia pole widzenia (FOV);  
- OBS #5: dostęp do zakładki kontrolnej (CNTL);  
- OBS #6: wyświetla typ aktualnie wybranej broni;  
- OBS #7: podaje polaryzację;  
- OBS #20: wybiera opcję SLAVE

  
Rys. 2.10 Menu WPN

### 2.2.8. Menu TGP

Menu TGP jest dostępne z poziomu menu głównego MFD za pomocą przycisku OBS #19. Zakładka jest aktywna jedynie wtedy, gdy przenoszony jest zasobnik TGP

  
Rys. 2.11 Menu TGP

### 2.2.9. Menu HAD

Menu HAD wybierane jest z menu głównego MFD przyciskiem OBS #2. Może być wybrane z dowolnym aktywnym trybem Master Mode, ale tylko w trybie A-G posiada zdolność operacyjną. Ponadto, aby móc korzystać z zakładki HAD musi zostać podwieszony zasobnik HTS i załadowany AGM-88. Wybór HAD bez załadowanego AGM-88 będzie wyświetlał zakładkę BLANK. Operacje wykonywane w oparciu o zakładkę HAD są bardzo podobne do operacji SMS, jednak HAD oferuje kilka funkcjonalności zakładki HSD. Za pomocą przycisku OBS #3 podobnie jak w przypadku planszy HSD można zmieniać zasięg lub poprzez przyciski OBS #19 i #20.

Widok oraz zobrazowania dostępne w menu HAD pokazuje rysunek 2.12.

  
Rys. 2.12 Menu HAD

### 2.2.10. Menu BLANK

W jednym z dwóch MFD jest to możliwe aby wyłączyć wyświetlanie na ekranie jakichkolwiek informacji. W tym celu należy wybrać z poziomu menu głównego zakładkę BLANK za pomocą przycisku OBS #1. Wówczas nie wyświetlane są żadne symbole i oznaczenia, pojawia się jedynie napis BLANK informujący o wybranej zakładce. Przyciski OBS #11-15 nadal pełnią podstawową rolę i za ich pomocą możliwy jest dostęp do innych zaprogramowanych zakładek.

### 2.2.11. Menu FCR

Menu FCR jest jedną z bardziej skomplikowanych zakładek, dostępną z poziomu menu głównego za pomocą przycisku OBS #20. Stanowi ona część bardzo rozbudowanego systemu Fire Control Radar. W dużym uproszczeniu zakładka ta przedstawia wszystko to co „widzi” radar. Może on być dostępny w trybie pracy A-A lub A-G w zależności od aktualnego trybu Master Mode.

W trybie A-A FCR jest włączany za pomocą przełącznika na panelu SNSR. Za każdym razem gdy zostaje włączony uruchamia on swoją wbudowaną procedurę testową BIT (Built –in Test), która może potrwać nawet do kilku minut.

Przycisk OBS #1 oznaczony jako CRM (Combined Radar Modes) służy do wyboru trybu pracy radaru i po naciśnięciu wyświetla dostępne tryby pracy: GM (skanowanie ziemi), GMT (wykrywanie poruszających się pojazdów), SEA (przeciw okrętowy), STBY (tryb gotowości) oraz ACM (walka powietrzna).

Przycisk OBS #2: Służy do wyboru związanego z trybem modu.

Przycisk OBS #3 służy do wyboru pola widzenia (FOV) i nie jest dostępny we wszystkich trybach. Pozwala na ustawienie FOV jako NORM lub EXP.

Przycisk OBS #4 zmienia stan radaru na stan gotowości (wówczas wszystkie symbole są usunięte) i podkreślony jest napis OVRD. Ponowne naciśnięcie przycisku OBS #4 powoduje powrót urządzenia do trybu operacyjnego.

Przycisk OBS #5 służy do otwarcia strony kontrolnej radaru (CNTL) umożliwiającej wybór wielu opcji jego trybu pracy. Strona ta jest niemal identyczna w trybie A-A jak również A-G.

Przycisk OBS #6 odpowiada za wybór trybu IDM. Domyślnie jest ustawiony na wartość ASGN (Assign), ale może zostać zmieniony na CONT (Continuous) lub DMD (Demand).

Przyciski OBS #7-10 odnoszą się do sojuszników i za ich pomocą można wybrać sojusznika z którym chcemy dzielić informacje radarowe.

Przycisk OBS #17 określa przestrzeń pionową, którą skanuje antena radaru. Domyślnie antena jest w stanie skanować stożek o kącie wierzchołkowym 4.9⁰. Ustawiając za pomocą OBS #17 wartość 1 antena skanuje taki obszar. Gdy zostanie ustawiona liczba większa niż 1 np. 2, 3 albo 4 obszar ulega odpowiedniemu pionowemu zwielokrotnieniu.

Przycisk OBS #18 służy do ustawienia azymutalnego zakresu skanowania anteny. Możliwe do ustawienia wartości kąta to 60⁰ (A6), 30⁰ (A3) oraz 10⁰ (A1). Zaletą węższego pasma skanowania anteny jest szybkość wykonywania czynności skanowania. Niestety niesie to za sobą konsekwencje możliwości nie wykrycia kontaktu na radarze.

Przyciski OBS #19 oraz OBS #20 służą do zwiększania oraz zmniejszania zasięgu radaru. (wypromieniowywanej mocy).

  
Rys. 2.13 Menu FCR tryb A-A dodatkowy tryb CRM

Tryb pracy radaru włącza się automatycznie po wybraniu trybu A-G w Master Mode. W menu wyboru dodatkowego trybu pracy radaru podobnie jak w trybie AA po lewej stronie znajdują się opcje dla wyboru trybów związanych z profilem AA (CRM oraz ACM), a po prawej stronie do wyboru znajdują się te związane z profilem AG (GM, GMT, SEA, BCN oraz STBY).

Mechanizacja pracy radaru w poszczególnych dodatkowych trybach jest taka sama jednak zmienia się czułość radaru dostosowana do wybranych kategorii celów. Radar renderuje odbijające powierzchnie z różnym wzmocnieniem. Kontakty wykryte podczas skanowania zobrazowane są w postaci białych kropek tak jak pokazano to na rysunku.

Przycisk OBS #1 pokazuje wybrany aktualnie tryb dodatkowy podobnie jak w przypadku modu AA.

Przycisk OBS #2 ustawia ręczny (manualny) lub automatycznie przełączany zasięg. W manualnym trybie za pomocą przycisków OBS #19-20 pilot wybiera pożądany zasięg natomiast w trybie automatycznym zasięg zmienia się gdy pozycja kursora przekroczy następny dostępny zasięg ustawiony dla FCR.

Przycisk OBS #3 jest ustawieniem FOV i przyjmuje cztery wartości NORM, EXP, DBS1 (Doppler Beam Sharpering) oraz DBS2.

Przycisk OBS #4 przełącza FCR w tryb czuwania. Wówczas na MFD wyświetlana jest plansza BLANK, a ponowne naciśnięcie przycisku OBS #4 ponownie uruchamia radar.

Przycisk OBS #5 otwiera menu kontroli radaru, które jest identyczne w trybie AA jak również AG. W zakładce tej możliwe jest ustawienie kanału skanowania radaru, w celu uniknięcia interferencji fal z innym radarem, zmiany intensywności symboli markera, wybór szerokości pasma (WIDE i NARROW), opóźnienia strumienia i inne.

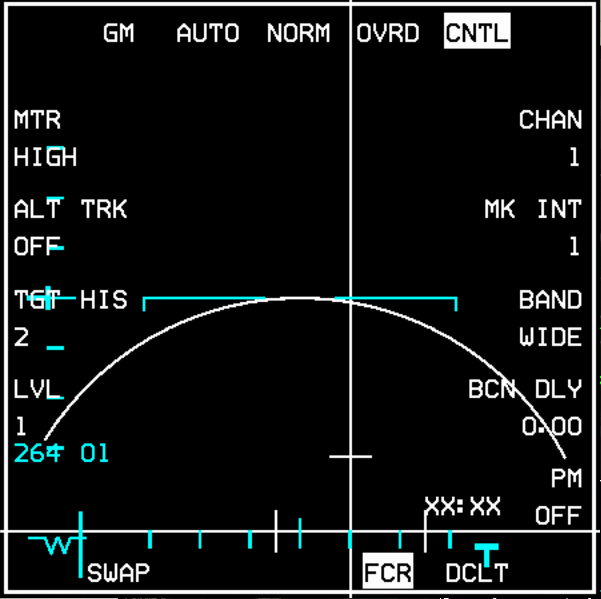
Przycisk OBS #7 pozwala na zamrożenie aktualnego widoku sytuacji radarowej

Przycisk OBS #8 pozwala na wybór kierunku promieniowania wiązki

Przycisk OBS #9 CZ (Cursor to Zero) pozwala na zresetowanie ustawienia kursora do jego domyślnej pozycji

Przycisk OBS #18 pozwala na wybór kąta azymutu skanowania (120⁰, 60⁰ oraz 20⁰), a w związku z tym także czasu przeszukiwania przestrzeni.

Przyciski OBS #19-20 podobnie jak w przypadku trybu AA służą do regulacji zasięgu Radarowego.

  
Rys. 2.14 Menu FCR tryb A-G, dodatkowy tryb GM

### 2.2.12. Menu RESET MENU

Zakładka RESET MENU osiągalna jest poprzez menu główne wyświetlacza MFD i naciśnięcie przycisku OBS #5. Pozwala ona na szereg ustawień samego wyświetlacza, transferu danych i jest rzadko używana. Ponowne wciśnięcie przycisku OBS #5 skutkuje wyjściem do menu głównego.

NVIS OVRD pozwala pilotowi na realizowanie nocnego lotu bez zakładania specjalnych gogli dostrajając natężenie oświetlenia wskaźników w kokpicie. Przy pomocy SBC DAY/NIGHT SET oraz SBC DAY/NIGHT RESET można ustawiać indywidualnie dla lotów dziennych i nocnych ustawienia oświetlenia w postaci gotowych programów.

Rysunek 2.15 przedstawia wygląd menu RESET MENU.

  
Rys. 2.15 Menu RESET MENU

* 1. OPRACOWANIE KONCEPCJI PRACY URZĄDZENIA MFD Z WYKORZYSTANIEM ISTNIEJĄCEGO STANOWISKA SYMULATORA LOTU SAMOLOTU ODRZUTOWEGO.

W pierwszej części podrozdziału omówiono kwestię budowy stanowiska symulatora lotu samolotu odrzutowego. Wyjaśniono założenia oraz czynniki, którymi należy kierować się podczas projektowania rozbudowy stanowiska o dodatkowe wskaźniki i przyrządy jak na przykład omawiany wyświetlacz wielofunkcyjny MFD. Celem tego jest zaznajomienie czytelnika z poszczególnymi urządzeniami wchodzącymi w skład stanowiska, systemem w oparciu o który funkcjonuje symulator oraz samą aplikacją symulatora. W dalszej części omówiono specyfikę transmisji danych, które muszą być współdzielone pomiędzy poszczególnymi urządzeniami aby było możliwe symulowanie ich funkcjonalności. Zapoznano czytelnika z podstawowymi technikami transmisji tych danych, których wykorzystanie umożliwia współdzielenie zasobów wykorzystywanych do wypracowania informacji zobrazowanej na danym wskaźniku. Omówiono wady i zalety poszczególnych rozwiązań, a także wybrano najkorzystniejsze w przypadku stanowiska symulatora.

Ostatnią część poniższego rozdziału stanowią rozważania na temat rozbudowy i przyszłościowego wykorzystania możliwości symulacyjnych stanowiska, a także urządzenia MFD. Wymieniono korzyści jakie płyną z rozbudowy i inwestowania w nowoczesne symulatory oraz ich walor edukacyjny. W tej części podrozdziału pokazano także najnowocześniejsze symulatory oraz rozwiązania jakie są stosowane do rozwiązywania problemów symulacji wskaźników i ekranów.

## 3.1. Przyjęte założenia

Urządzenie ekranu wielofunkcyjnego MFD powinno działać w sposób autonomiczny i uniwersalny, tak aby możliwe było rozbudowywanie stanowiska, aktualizowanie oprogramowania do nowszych wersji oraz dokonywanie zmian w strukturze aplikacji MFD nie powodujące zmiany całego systemu symulatora. Pod względem funkcjonalnym urządzenie MFD powinno pełnić dwukierunkową komunikację – z jednej strony odbierając sygnały sterujące od użytkownika, zaś z drugiej odbierając i przetwarzając dane parametrów lotu z symulatora.

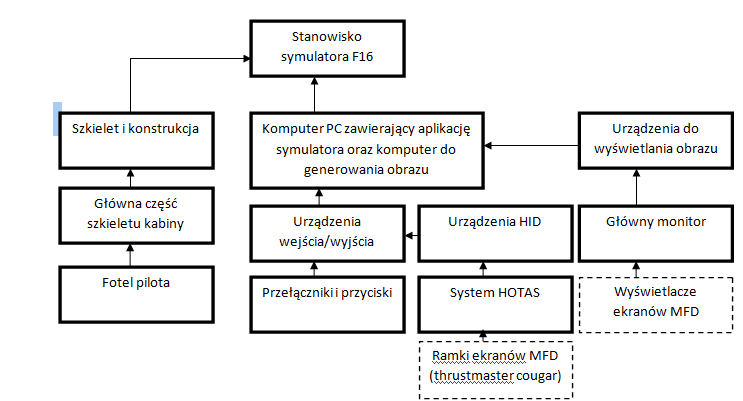
Założenia oraz wymagania jakie można sformułować w stosunku do pracy ekranu wielofunkcyjnego są następujące:

* Urządzenie powinno mieć możliwość autonomicznej pracy, tzn. działać również poza strukturą systemu symulatora;
* Urządzenie powinno pobierać dane parametrów lotu w czasie rzeczywistym;
* Urządzenie powinno na wzór rzeczywistego przyrządu korzystać z komputera zdolnego do przetwarzania danych i generowania grafiki w czasie rzeczywistym;
* Urządzenie powinno posiadać odpowiedni interfejs umożliwiający komunikację z użytkownikiem w postaci ramki z przyciskami;
* Aplikacja urządzenia powinna działać w trybie wielowątkowym tak aby jak najbardziej wydajnie zagospodarować dostępny czas pracy procesora
* Aplikacja powinna być uruchamiana w trybie rozszerzonego ekranu na dwóch monitorach dopasowanych wymiarami do ramek z przyciskami (rozmiar przekątnej ekranu 7”)
* Urządzenie powinno mieć możliwość jak najwierniejszego odwzorowania funkcjonalności ekranu MFD zaimplementowanego w symulatorze Falcon BMS.
* Monitory powinny posiadać standardowe wejścia wideo VGA, a urządzenia ramek powinny być urządzeniami typu HID z wejściem USB.

Spełnienie powyższych założeń projektowych gwarantuje wysoki stopień symulowania pracy rzeczywistego urządzenia. Ze względu na charakter autonomiczny i mobilny (przenośny) urządzenia, jego użytkownik powinien mieć możliwość uruchomienia wskaźnika nawet wówczas gdy nie posiada aplikacji symulatora Falcon BMS.

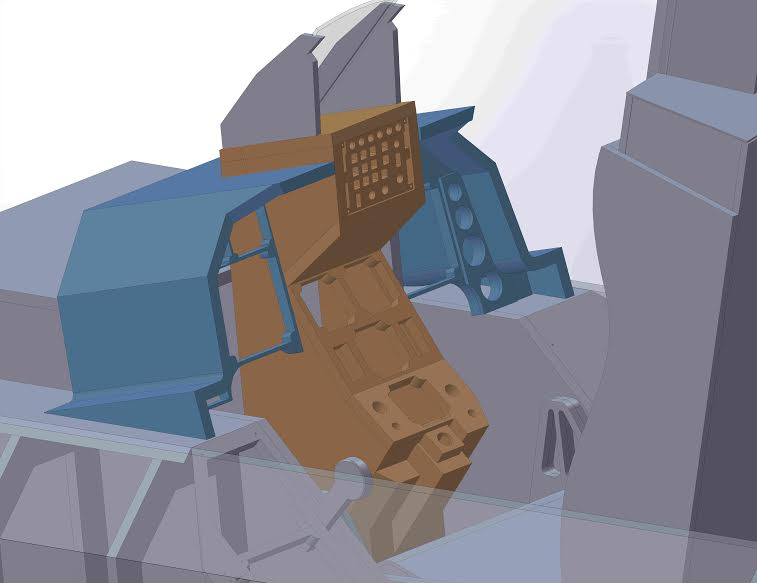
## 3.2. Opis stanowiska symulatora

Stanowisko symulatora samolotu odrzutowego F16 składa się z komputera desktopowego zawierającego oprogramowanie symulatora Falcon BMS, ekranu do wyświetlania obrazu, szkieletu konstrukcyjnego, fotela pilota oraz urządzeń wejścia/wyjścia w postaci zestawów przycisków, przełączników i kontrolerów HID (system HOTAS). Rysunek 3.1 przedstawia schematycznie strukturę stanowiska symulatora samolotu F16. Elementy tworzące fizyczną implementację wskaźników MFD zaznaczono w prostokątach linią kreskową.



*Rys 3.1 Struktura stanowiska symulatora samolotu F16*

Szkielet konstrukcyjny składa się z zasadniczej części oraz z części fotela pilota. Jego elementy konstrukcyjne wykonano tradycyjną technologią używając materiałów drewnianych i drewnopochodnych. Elementy każdej z części zaprojektowano w środowisku CAD, a następnie poddano podstawowej obróbce mającej na celu wycięcia otworów, frezowanie i inne czynności stolarskie. Elementy łączone są ze sobą poprzez technologię klejenia, a także łącznikami w postaci śrub i zacisków. Konstrukcja ta stanowi podstawę symulatora. W miejsca otworów przelotowych wmontowano urządzenia we/wy w postaci wieloelementowych paneli przełączników, przycisków, lampek, a także pojedyńcze elementy. Otwory montażowe dla wskaźników MFD zaprojektowane w programie typu CAD zobrazowano na rysunku 3.2. Część szkieletu zawierającą te otwory oznaczono kolorem niebieskim.

  
*Rys 3.2 Model CAD szkieletu konstrukcyjnego symulatora. Kolorem niebieskim oznaczono fragment zawierający otwory montażowe dla wskaźnika MFD*

Komputer desktopowy klasy PC obok szkieletu konstrukcji i urządzeń do prezentacji wizualnej stanowi główną część systemu symulatora. Odpowiada on za wykonywanie aplikacji symulatora Falcon BMS, a także pośredniczy w komunikacji pomiędzy użytkownikiem, a zainstalowanym oprogramowaniem. Poprzez odpowiedni interfejs (np. USB) urządzenia zewnętrzne (np. mikrokontrolery) wysyłają informację o sygnałach dyskretnych (np. wciśnięcie przycisku) lub ciągłych (np. zmiana położenia joystika sterującego) do komputera, gdzie dane te są przetwarzane i przesyłane do aplikacji symulatora lub aplikacji konkretnego urządzenia zewnętrznego (np. wskaźnika MFD). Komputer posiada złącze gniazda RJ45 umożliwiające połączenie kilku jednostek w sieć lokalną (LAN). Rozwiązanie takie pozwala na rozdzielenie zadania renderowania obrazu aplikacji głównej symulatora (Falcon BMS 4.32) oraz zadania rederowania grafiki wskaźników MFD, co znacznie poprawia wydajność systemu. Komputer (komputery) są także odpowiedzialne za dostarczenie danych obrazu do odpowiedniego wskaźnika (ekranu głównego lub ekranów poszczególnych urządzeń). W przypadku rozważanego stanowiska połączenie z urządzeniami do wizualizacji zrealizowano poprzez standard VGA.

Urządzeniami wejścia/wyjścia są wszystkie urządzenia zewnętrzne podłączone do odpowiednich portów komputera(komputerów) PC. Podzielono je ze względu na pełnioną przez nie funkcję w systemie. Panele z przyciskami i przełącznikami zaprojektowano w oparciu o działanie mikrokontrolera. Sygnały dyskretne lub ciągłe generowane przez użytkownika odbierane są bezpośrednio przez mikrokontroler i dalej przesyłane do komputera desktopowego interfejsem USB. Poprzez ten sam standard do komputera klasy PC podłączone są również ramki wskaźnika MFD, które są urządzeniami USB typu HID i spełniają założenie funkcjonalne urządzeń „Plug and Play” (ang. „Podłącz i używaj”). Innym rodzajem urządzeń typu HID wchodzących w skład stanowiska symulatora są urządzenia systemu HOTAS – Joystick do sterowania powierzchniami sterowymi płatowca oraz dźwignia kontroli pracy zespołu napędowego. Funkcjami dodatkowymi realizowanymi przez te urządzenia jest także kontrola przyrządów celowniczych, systemu uzbrojenia lotniczego oraz wybranych przyrządów w ramach cechy SOI („Sensor of Interest”).

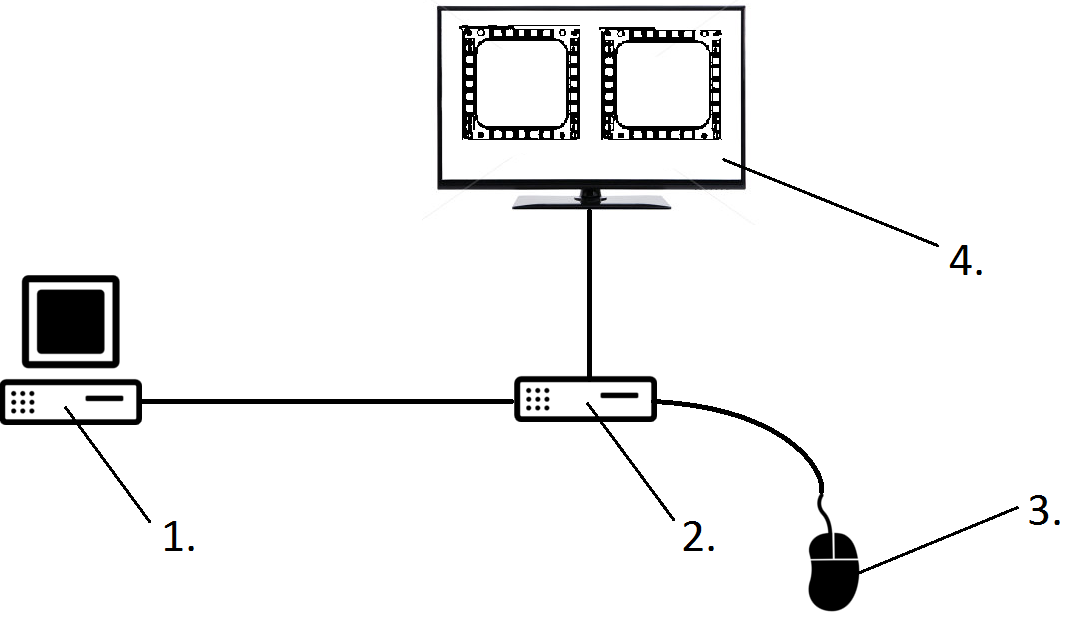
Urządzenia do wizualizacji to monitory i wyświetlacze segmentowe. Wyświetlacze segmentowe (np. do zobrazowania częstotliwości radiostacji i kanału) wchodzą w skład urządzeń wyjścia/wejścia i komunikują się z komputerem tak jak to opisano w poprzednim akapicie. Monitorami, które użyto w stanowisku symulatora to monitor główny do zobrazowania aplikacji Falcon BMS oraz dwa monitory dedykowane dla wskaźników MFD. Standardem, który wykorzystano do komunikacji z komputerem (komputerami) PC jest standard VGA. W celu odciążenia od pracy głównej jednostki obliczeniowej i poprawienia wydajności działania systemu, jak wyjaśniono to w poprzedniej części bieżącego rozdziału, możliwe jest podłączenie drugiego komputera PC wyspecjalizowanego do renderowania wskazań symboliki obydwu MFD. Wówczas ekrany dedykowane do wskaźników MFD podłączone są poprzez standard VGA do komputera na którym uruchomiona jest aplikacja symulatora MFD.

## 3.3. Realizacja ekranu MFD

Działanie urządzenia MFD opiera się o symulację stworzoną w formie aplikacji uruchamianej na platformie Microsoft Windows. Aplikacja jest wykonywana na osobnym komputerze klasy PC. Obydwa komputery (główny oraz renderujący MFD) połączone są we wspólnej sieci lokalnej (LAN), w której mogą się komunikować oraz wymieniać między sobą informacje poprzez transmisję danych pomiędzy komputerem symulatora, a komputerem wskaźnika. Przetworzone dane następnie wykorzystywane są do renderowania grafiki czasu rzeczywistego. Interfejs graficzny (GUI) zobrazowany jest w postaci dwóch okien jako odpowiedników lewego i prawego ekranu MFD. Użytkownik w obydwu oknach widzi wskazania urządzeń MFD. Okna można przemieszczać, a ich rozmiar (600 pikseli w pionie i poziomie) dopasowany jest do dedykowanych ekranów, dołączonych do niniejszej pracy jako praktyczna realizacja urządzenia. Ekrany połączone są z komputerem PC poprzez standard VGA i funkcjonują w ramach tzw. rozszerzonego pulpitu, więc ustawienie okien jest dowolne. W ramach wskaźnika do pracy dołączono także specjalne dedykowane ramki MFD Cougar firmy Thrustmaster. Ramki te są urządzeniami USB klasy HID zawierającymi 20 przycisków bistabilnych OBS oraz 4 przełączniki bistabilne GAIN, BRT, SYM i CON. Dokładne parametry zastosowanych urządzeń (ekranów oraz ramek z przyciskami) zestawiono w tabeli 3.1.

Parametry zastosowanych ekranów i ramek Tab. 3.1

|  |  |
| --- | --- |
| **Ekrany** | |
| **Parametr** | Wartość |
| **Przekątna (cale)** | 7 |
| **Natywna rozdzielczość (piksele)** | 800x600 |
| **Wymiar matrycy** |  |
| **Obsługiwane rodzaje złącz** | HDMI, VGA, 2xAV |
| **Zasilanie (volt)** | 12 |
| **Ramki Coughar** | |
| **Liczba przycisków na ramkę** | 20 + 4 |
| **Wymiar** |  |
| **Klasa urządzenia USB** | HID |
| **Programowane diody led (szt.)** | 2 |

Rys 3.3 Wygląd ramek oraz wyświetlaczy MFD w trakcie renderowania symboliki MFD

Rozwiązanie takie zapewnia wysoki stopień odwzorowania działaia prawdziwych wskaźników MFD. Ponadto „rozbicie” działania wskaźnika oraz samej aplikacji symulatora na inne komputery znacznie skraca czas renderowania grafiki i powoduje przyspieszenie działania obydwu aplikacji. Inną korzyścią wynikającą z rozdzielenia symulatora oraz wskaźnika na dwa komputery jest możliwość autonomicznej pracy wskaźnika bez konieczności działania symulatora co pozwala na testowania niektórych funkcji aplikacji MFD. Możliwa jest także rozbudowa aplikacji i jej aktualizowanie bez przenoszenia stanowiska symulatora – w przypadku omawianego symulatora nie byłoby to problemem jednak w większych i bardziej rozbudowanych modelach jest to bardzo kłopotliwe, a wręcz niemożliwe do zrealizowania w praktyce – dlatego też wymaga się od wskaźnika autonomiczności. W przypadku awarii jednostki centralnej symulatora wskaźnik jest nadal sprawny i wystarczy podłączyć nowy komputer z zainstalowaną aplikacją Falcon BMS 4.32 aby odzyskać pełną funkcjonalność symulatora.

Wadami stosowania wielu jednostek wykonawczych (komputerów PC) jest rosnący stopień skomplikowania układu, zwiększone ryzyko awarii systemu i jego rosnąca cena oraz koszta eksploatacji.

Użytkownik może zastosować jeden lub dwa monitory jako wskaźniki MFD. W przypadku posiadania tylko jednego ekranu symulacja zarówno lewego jak również prawego MFD zobrazowana jest na jednym i tym samym ekranie. Jest to rozwiązanie najprostsze i najbardziej uzasadnione z ekonomicznego punktu widzenia. Stopień symulacji wówczas jest nieporównywalnie mniejszy niż w przypadku zastosowania dwóch dedykowanych do tego celu ekranów.

## 3.4. Techniki transmisji danych i współdzielenia zasobów

Aby było możliwe korzystanie z jakichkolwiek dodatkowych urządzeń współpracujących z symulatorem, a właściwie komputerem na którym znajduje się aplikacja symulatora potrzebna jest właściwa ekstrakcja danych z aplikacji symulatora, przesłanie danych do urządzenia docelowego oraz ich odpowiednia obróbka w czasie rzeczywistym w docelowym urządzeniu. To do producenta oprogramowania (w tym przypadku producenta symulatora lotu) należy zapewnienie programistom sposobu na pozyskiwanie danych z aplikacji. Dane te służą do wypracowania odpowiedniego zobrazowania na urządzeniu docelowym. Producent może udostępniać je do dyspozycji programistów na kilka sposobów. Najprostszym z nich jest mechanizm pamięci współdzielonej.

Pamięć współdzielona polega na wykorzystaniu wspólnej przestrzeni adresowej dla kilku procesów (programów). Domyślnie system Windows przydziela każdemu procesowi pewien obszar w przestrzeni adresowej w którym program umieszcza zmienne, posiada stos, a także dynamicznie przydziela i zwalnia część z tej pamięci. Każdorazowe odwołanie się do adresu spoza tego obszaru skutkuje natychmiastowym wstrzymaniu procesu przez system operacyjny i wyświetleniem błędu wyjścia poza zakres pamięci. Jak więc programy w systemie Windows współdzielą wymagane zasoby pomiędzy sobą? Rozwiązaniem są biblioteki dołączane dynamicznie (DLL) do których kodu obiektowego programy mogą się odwoływać w czasie uruchomienia. Nie ma także ograniczeń w liczbie programów korzystających z biblioteki dynamicznej. Program może wywoływać funkcje z biblioteki, a także odwoływać się do zdefiniowanych w bibliotece zmiennych. Zmienne zdefiniowane w kodzie obiektowym jako nadające się do wyeksportowania mogą być współdzielone pomiędzy wieloma procesami.

Producenci aplikacji Falcon BMS 4.32 udostępnili bibliotekę dynamiczną w której znajduje się definicja klasy zawierającej pola odpowiadające poszczególnym zmiennym parametrom lotu, a także metody pobierania i zaktualizowania wartości tych zmiennych oraz metody przygotowujące przed użyciem klasę. Rozwiązanie problemu dostępu do zmiennych symulatora polega więc na dołączeniu do własnego programu biblioteki dynamicznej (np. jako referencja w C# udostępniając przestrzeń nazw), zadeklarowaniu w programie obiektu typu „FlighData” oraz wywołaniu odpowiednich metod klasy w celu zainicjalizowania obiektu do dalszej pracy i zaktualizowania zawartości pól wartościami zmiennych parametrów lotu. Jeśli aktualizowanie danych parametrów będzie następowało w powtarzającej się pętli, wówczas w czasie rzeczywistym będziemy mieć dostęp do zmiennych wartości parametrów z poziomu kodu źródłowego.

Mając aplikację o stałym dostępie do danych parametrów lotu pobieranych w czasie rzeczywistym z symulatora potrzeba rozwiązać kwestię transmisji danych pomiędzy dwoma komputerami (między symulatorem i aplikacją wyciągającą z niego dane – działającymi na wspólnym komputerze, a komputerem z aplikacją do wyrenderowania grafiki stanowiącej zobrazowanie tej informacji). Istnieje bardzo wiele sposobów realizacji tego zagadnienia. Popularnym rodzajem interfejsu jest interfejs USB, dane można także przesyłać pomiędzy urządzeniami przez sieć (poprzez kabel lub bezprzewodowo). Transmisja sieciowa może odbywać się na bardzo duże odległości, dlatego też jest rozwiązaniem bardzo ciekawym. Komputer z danymi wyciągniętymi z symulatora może pełnić rolę serwera do którego zgłaszają zapytania klienci (hosty w sieci) i odpowiednim rodzajem protokołu pobierają dane w postaci ciągu pakietów. Istnieją dwa podstawowe rodzaje protokołów komunikacji sieciowej w trybie klient-serwer (poprzez pakiety IP). Są to protokoły TCP oraz UDP. Różnią się one od siebie głównie ideą działania i pewnością transmisji. Pakiety TCP są pakietami wolniejszymi, ale zarazem pewniejszymi w stosunku do UDP. Zawierają sumy kontrolne i użytkownik wysyłając taki pakiet ma pewność, że dotrze on do odbiorcy. Zaleca się stosowanie tego typu pakietów do większości zastosowań. Jednak w niektórych dziedzinach techniki większą wagę przywiązuje się do szybkości transmisji, a nie do jej wysokiej niezawodności. Wówczas niezastąpione są pakiety UDP które mimo ryzyka utraty części danych świetnie sprawdzają się w małych sieciach lokalnych np. podczas wideokonferencji, przesyłania dużych ilości niewielkich rozmiarowo danych itp.

Ze względu na popularność oraz pewność transmisji w tworzonej aplikacji symulującej działanie wskaźnika MFD do transmisji danych pomiędzy komputerem źródłowym z zainstalowaną aplikacją Falcon BMS 4.32, a aplikacją docelową symulującą działanie wskaźnika MFD wybrano rodzaj transmisji TCP/IP. Aplikacja zainstalowana na komputerze zawierającym symulator pełni funkcję serwera który dysponując danymi (parametrami lotu) wysyła je w formie pakietów do komputera docelowego na którym renderowana jest aplikacja MFD.

## 3.5. Możliwości rozbudowy stanowiska i wskaźników MFD

Stanowisko symulatora samolotu odrzutowego F16 jest obecnie stale rozwijane i rozbudowywane. Posiada duży potencjał rozwojowy i oferuje wysoki stopień symulacji funkcjonowania prawdziwego samolotu. Niektóre jego elementy nadal są w fazie projektowania i sprawa technicznego rozwiązania ich budowy jest nadal „otwarta”.

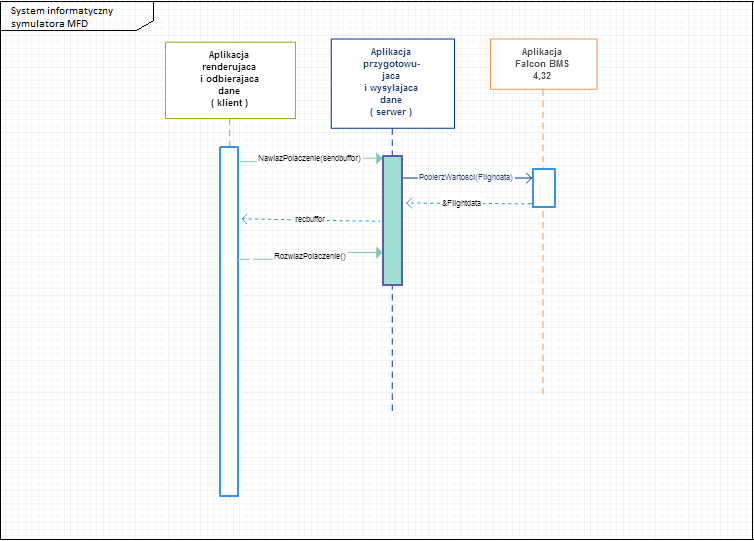
Jednym z najważniejszych wskaźników (ze wzlędu na praktyczne jego wykorzystanie) we wnętrzu kokpitu F16 jest wskaźnik HUD. Obecnie symulator F16 nie jest wyposażony w żaden wskaźnik HUD przez co stopień symulacji tego urządzenia jest stosunkowo niski. Jego realizacja w przyszłości może polegać na wykorzystaniu technologii optycznej i symulowania działania poprzez rzutnik, zwierciadła i filtry optyczne. Z pewnością jego budowa będzie dużym wyzwaniem inżynierskim.

Istotną rolę odgrywa także sposób wizualizacji środowiska w którym porusza się SP. Najprostszy sposób wizualizacji za pomocą monitora w niewielkim stopniu oddziałuje na zmysły użytkownika przyczyniając się do niskiego poziomu wrażeń z symulacji. Nowoczesne symulatory wyposażone są w wielkie ekrany rozmieszczone dookoła kabiny oraz sieć rzutników optycznych generujących obrazy na ekranach. W przypadku takiej wizualizacji otoczenia użytkownik ma wrażenie, że znajduje się we wnętrzu prawdziwego samolotu, a zmieniające się otoczenie w bardzo dużym stopniu oddziałuje na odbierane wrażenia.

Obok wskaźnika HUD podstawową rolę prowadzenia walki powietrznej pełni wskaźnik nahełmowy HMD. Z pewnością zasymulowanie jego działania i budowy należy do najtrudniejszy i najbardziej kosztownych funkcji symulatora. Z uwagi na to, że nie jest on wymagany w trakcie prowadzenia rzeczywistych operacji lotniczych na SP, symulatora nie wyposażono w takie urządzenie. Jego ewentualna symulacja ogranicza się do renderowania jego symboliki na głównym monitorze.

Realizacja ekranów MFD w wysokim stopniu odzwierciedla ich realny wygląd i funkcjonowanie. Potencjalne możliwości rozbudowy ograniczają się do wymiany ekranów na ekrany lepiej dopasowane do ramki, zastosowanie ekranów lepszej jakości, montaż ramek i ekranów do jednej obudowy w celu uczynienia wskaźnika MFD jeszcze bardziej autonomicznym i niezależnym od architektury symulatora.

Na rys. 3.3 przedstawino ideowy schemat działania stanowiska funkcjonującego w architekturze klient-serwer, w formie diagramu sekwencji UML.



Rys. 3.3 Schemat sekwencji UML projektowanego systemu informatycznego

* 1. PROJEKT I WYKONANIE OPROGRAMOWANIA SYMULUJĄCEGO DZIAŁANIE MFD ORAZ JEGO INTEGRACJA ZE STANOWISKIEM SYMULATORA SAMOLOTU F16.

Poniższy rozdział stanowi główną część niniejszej pracy dlatego poświęcono mu najwięcej miejsca starając się szeroko opisać rozwiązania techniczne zastosowane podczas tworzenia oprogramowania symulującego działanie wskaźnika MFD. Opisano w nim problematykę tworzenia tego typu aplikacji, założenia projektowe, a także przedstawiono zestaw niezbędnych narzędzi i bibliotek wspomagających tworzenie oprogramowania i znacznie przyspieszających ten proces. Czytelnik może dowiedzieć się również z jakich programów korzystano podczas tworzenia dokumentacji technicznej aplikacji, która stanowi równie ważną część całego procesu wytwórczego oraz pełni nieocenioną rolę na etapie późniejszych modyfikacji i ulepszeń oprogramowania. W dalszych podrozdziałach zaprezentowano sposób w jaki zaimplementowano algorytmy realizujące poszczególne etapy pracy programu do renderowania, niezbędne klasy oraz tzw. maszynę stanu aplikacji. Omówiono z jakich komponentów składa się program, a także jakie posiada zasoby. Wyjaśniono rolę jaką pełnią w aplikacji tzw. „Shadery” czyli programy cieniujące, sposób zarządzania danymi geometrii (wierzchołki i współrzędne teksturowe), a także proces wczytywania i filtrowania tekstur. Ponadto opisano pracę dodatkowego wątku odpowiedzialnego za połączenie sieciowe z aplikacją serwera oraz za wczytywanie bieżących wartości parametrów i zmiennych. Pełny i wyczerpujący opis elementów aplikacji, a także schematy standardu UML umieszczono w dokumentacji technicznej znajdującej się w załączniku do niniejszej pracy. Dodatkowo opisowi poddano aplikację serwera pełniącą rolę gromadzenia danych, ich obróbki i dalszej wysyłki do docelowej aplikacji. W rozdziale przedstawiono również sposób dystrybucji oprogramowania i proces tworzenia dokumentacji.

## 4.1. Założenia projektowe i wymagania stawiane aplikacji

Przed przystąpieniem do projektu i wykonania pierwszych prototypów poszczególnych algorytmów należy określić wstępne wymagania oraz założenia jakie musi spełniać oprogramowanie w celu zagwarantowania aby działało ono w sposób prawidłowy i zgodny z oczekiwaniami odbiorcy. Wymagania można podzielić na sprzętowe (dotyczące docelowego środowiska aplikacji) oraz wymagania strukturalne – czyli ogólny zarys i szkielet programu, interakcja z użytkownikiem i inne czynniki dotyczące samej aplikacji. W ten sposób można określić następujące założenia i wymagania mające wpływ na działanie oprogramowania po jego wdrożeniu.

Wymagania sprzętowe:

* Docelowa platforma (system operacyjny)
* Niezbędny sprzęt do uruchomienia aplikacji
* Wersja sterownika
* Zajmowany obszar pamięci masowej użytkownika

Docelowa platforma aplikacji ma wpływ na wynikowy plik uruchamiany przez użytkownika, sposób przechowywania danych programu i zmiennych w pamięci oraz wiele innych rzeczy. Każdy system operacyjny ma swój własny interfejs przeznaczony do komunikacji z aplikacją w postaci API dostarczanego programistom. System taki przypomina komunikację klient-serwer. Strona kliencka (aplikacja) wywołuje odpowiednią funkcję w celu uzyskania jakiegoś efektu (np. utworzenia okna dialogowego), a strona serwerowa (system) w odpowiedzi na zapytanie klienta realizuje ten proces. Sposób w jaki to robi jest całkowicie ukryty przed użytkownikiem przez co stanowi to swego rodzaju interfejs. Interfejs ten jest różny dla różnych platform (systemów) więc niemożliwym jest napisanie wieloplatformowej aplikacji stosując pojedyncze API od producenta systemu. Problem ten rozwiązuje się stosując równocześnie kilka interfejsów i dyrektyw które w czasie uruchomienia programu (run-time) przełączają go na właściwe ścieżki prowadzące do wywołań interfejsu platformy na której został uruchomiony. Niestety jest to okupione większym miejscem w przestrzeni pamięci jakie zajmuje więcej kodu, jego komplikacją, a także niekiedy spowolnieniem i tworzeniem się tzw. wąskich gardeł aplikacji (ten sam algorytm może działać z różną szybkością na różnych platformach). Ponadto jeśli do powyższego problemu dochodzą inne czynniki jak np. systemy 32 i 64 bitowe znacznie komplikuje to tworzenie cross-platformowych aplikacji i czasami rozsądnym rozwiązaniem wydaje się być wybranie jednego docelowego systemu. W przypadku aplikacji będącej przedmiotem niniejszej pracy wybrano 32 bitową platformę z rozdziny Microsoft Windows w wersji XP oraz nowszych (Vista, 7 oraz 8.1) z uwagi na popularność systemu oraz fakt, że na tym systemie znajduje się oprogramowanie symulatora samolotu F-16 Falcon. Starsze wersje systemu mogą nie obsługiwać niektórych makr i funkcji z uwagi na brak niektórych bibliotek dynamicznych .dll. Ponadto Microsoft w starszych wersjach nie implementował kontrolek renderowanych przez akceleratory graficzne i najczęściej wspierał starsze wersje biblioteki, dzisiaj oznaczone jako przestarzałe.

Kolejnym czynnikiem i ważnym założeniem jest sprzęt na jakim zostanie uruchomiona aplikacja. Decydującą rolę odgrywa tu ilość pamięci oraz sprzęt do akceleracji graficznej. W projektowanej aplikacji pamięć nie powinna stanowić problemu – w nowoczesnych urządzeniach znajdują się jej znaczne zasoby, a aplikacja zajmować będzie niewielki ułamek procenta całej przestrzeni (większość wczytywanych tekstur, które z pewnością pochłaniają najwięcej miejsca zostanie od razu przesłana do pamięci karty graficznej, a niepotrzebne dane będą zwalniane w trakcie pracy aplikacji). Kwestia akceleratora graficznego (GPU) także nie powinna stanowić problemu dla aplikacji. Współczesne karty graficzne doskonale radzą sobie z potokowym przetwarzaniem znacznych ilości geometrii i obliczeń jednostek cieniujących (tzw. Shaderów) i przewyższają pod tym względem procesory.

Dla programisty ważnym czynnikiem jest sterownik udostępniany przez producenta sprzętu graficznego gdyż od tego zależy która wersja biblioteki OpenGL może zostać uruchomiona oraz która wersja programu cieniującego jest obsługiwana przez sterownik. Do wersji biblioteki 3.2 wersje programów cieniujących oznaczano jako 1.20, 1.30, 1.40 oraz 1.50. Nowocześniejsze wersje biblioteki (np. 4.0) odpowiadają nazwie programu Shadera (4.0). Jest to na tyle ważne, że po zadeklarowaniu użytkowania Shadera w odpowiedniej wersji (np. 1.50) w przypadku posiadania przez odbiorcę Shadera w wersji niższej program nie zostanie uruchomiony. Jednak dozwolone jest użytkowanie wielu różnych Shaderów i dostosowywanie odpowiedniego do platformy na której uruchomiona została aplikacja.

Zajmowana pamięć masowa przez aplikację nie powinna przekraczać kilkudziesięciu megabajtów włączając w to dane tekstur, pliki konfiguracyjne, aplikację i zasoby, pliki shakerów oraz dodatkowe pliki dla użytkownika jak instrukcja obsługi czy tzw. pliki LOG. W celu poprawienia tego można zastosować różne metody kompresji gromadzonych na dysku danych, jednak w przypadku niewielkich aplikacji do jakich należy projektowana aplikacja nie jest to niezbędne.

Wymagania strukturalne:

* Główna idea programistyczna (budowa programu)
* Wybór języka
* Interakcja z użytkownikiem i przetwarzanie sygnałów wejściowych
* Zastosowane narzędzia w postaci środowiska oraz bibliotek
* Końcowa postać aplikacji

Wymagania jakie postawiono aplikacji pod względem jej budowy to stworzenie systemu składającego się z dwóch komponentów działających w trybie klienta i serwera. Strona serwera odpowiada za przygotowanie danych wykorzystywanych w procesie renderowania, ich obróbki oraz wysyłki do aplikacji renderującej. To w aplikacji serwerowej następuje wydobywanie danych z aplikacji symulatora Falcon BMS. Rozwiązanie takie ma wiele zalet choćby fakt, że dysponując odpowiednią dokumentacją użytkownik może sam napisać aplikację wysyłającą dane do aplikacji renderującej integrując w ten sposób aplikację z dowolnym systemem informatycznym. Z tego też względu aplikacja serwerowa powinna być zbudowana jak najprościej, w ograniczonym stopniu impelentując interfejs GUI i przetwarzająca dane jak najszybciej. Aplikacja serwera powinna być w jak największym stopniu obiektowa i wielowątkowa w celu przyspieszenia jej działania. Z tego powodu dobrym wyborem jest język wysokiego poziomu w pełni obiektowy np. C#. Aplikacja kliencka, odbierające dane od serwera i renderująca obraz wynikowy powinna cechy obiektowości, a jednocześnie być bardzo wydajna tak aby czas wyrenderowania pojedynczej klatki obrazu był jak najmniejszy. Jej budowa powinna obejmować główną pętlę programu, struktury danych reprezentujące renderowany obraz i okna, struktury danych kontrolujące sygnaly wejściowe oraz pracę akceleratorów graficznych. Użytkownikowi musi zostać wyrenderowany obraz w postaci dwóch okien (na wzór dwóch ekranów MFD) w których zostaną wyświetlone tekstury odwzorowujące wygląd elementów wyświetlacza. Użytkownik za pomocą myszy musi generować sygnały wejścia, a aplikacja je przetwarzać i zobrazowywać odpowiedzi. Wymagania te spełnia język C++ ,w którym możliwe jest obiektowe reprezentowanie danych oraz duża swoboda w operowaniu na przestrzeni pamięci operacyjnej i niskopoziomowej kontroli sprzętu.

Sposób interakcji z użytkownikiem powinien stanowić system obsługi zdarzeń działający na zasadzie, że w momencie zaistnienia pewnego zdarzenia (np. kliknięcia przyciskiem na ramce MFD Coughar) aplikacja reaguje wywołując odpowiednią funkcję lub metodę odpowiedzi na dane zdarzenie. Istotnym założeniem jest to, że użytkownik może wchodzić w interakcję z aplikacją również za pomocą myszy lub innego urządzenia wskazującego i włączyć lub wyłączyć wirtualną ramkę MFD w postaci tekstury. Proces sprawdzania wystąpienia zdarzeń powinien być prowadzony w ciągłej pętli, w postaci pobierania wiadomości z kolejki wiadomości aplikacji (do której wiadomości zapisywane są przez system).

Zastosowane narzędzia powinny obejmować środowisko programistyczne dla platformy Windows 32 bit, bibliotekę graficzną, bibliotekę matematyczną do przekształceń geometrycznych, bibliotekę pomocniczą do ładowania tekstur z dysku oraz ewentualne narzędzia do obróbki graficznej i pracy z obrazami. Jako założenie przyjęto korzystanie ze środowiska Microsoft Visual Studio 2013, biblioteki OpenGL, biblioteki GLM, programu Gimp 2 oraz biblioteki SOIL. Wszystkie te niezbędne narzędzia opisano w kolejnym rozdziale.

Postać końcowa aplikacji powinna składać się z pliku wykonywalnego .exe który po uruchomieniu przeprowadzi proces instalacji w domyślnym lub wskazanym przez użytkownika katalogu, utworzy skróty na pulpicie oraz w menu start, a także w przypadku wyrażenia takiej woli przez użytkownika uruchomi natychmiast zainstalowaną aplikację. Podsumowując przedstawione wymagania sprzętowe i strukturalne dla aplikacji renderującej dane zestawiono w tabeli 4.1.

*Wymagania sprzętowe i strukturalne stawiane aplikacji Tabela 4.1*

|  |  |  |
| --- | --- | --- |
| **Minimalne wymagania systemowe** | | |
|  | Aplikacja renderująca obraz “klient” | Aplikacja renderująca obraz “serwer” |
| **System operacyjny** | Microsoft Windows XP lub nowszy | Microsoft Windows XP lub nowszy z platformą .dotnet |
| **Sprzęt** | Dowolna platforma z akceleratorem graficznym z pamięcią ponad 32MB oraz pamięcią operacyjną większą niż 16MB, wyposażenie w mysz | - |
| **Wersja sterownika** | Dowolny obsługujący GLSL 1.20 i nowszy | - |
| **Miejsce na dysku** | 100MB | Mniej niż 1MB |
| **Wymagania strukturalne** | | |
| **Budowa programu** | Obiektowy z elementami strukturalnymi, korzystający z WinAPI | Obiektowy, wielowątkowy |
| **Język programowania** | C++ 11 | C# |
| **Interakcja z użytkownikiem i przetwarzanie sygnałów wejściowych** | Za pomocą tzw. obsługi zdarzeń, prezentowanie efektów w postaci GUI, możliwość włączenia lub wyłączenia wirtualnej ramki MFD | brak GUI, tryb konsolowy, wymagane podanie portu przez użytkownika |
| **Narzędzia** | MS Visual Studio 2013, biblioteka OpenGL, biblioteka GLM, biblioteka SOIL, program graficzny Gimp 2 |  |
| **Końcowa postać aplikacji** | Plik instalacyjny .exe |  |

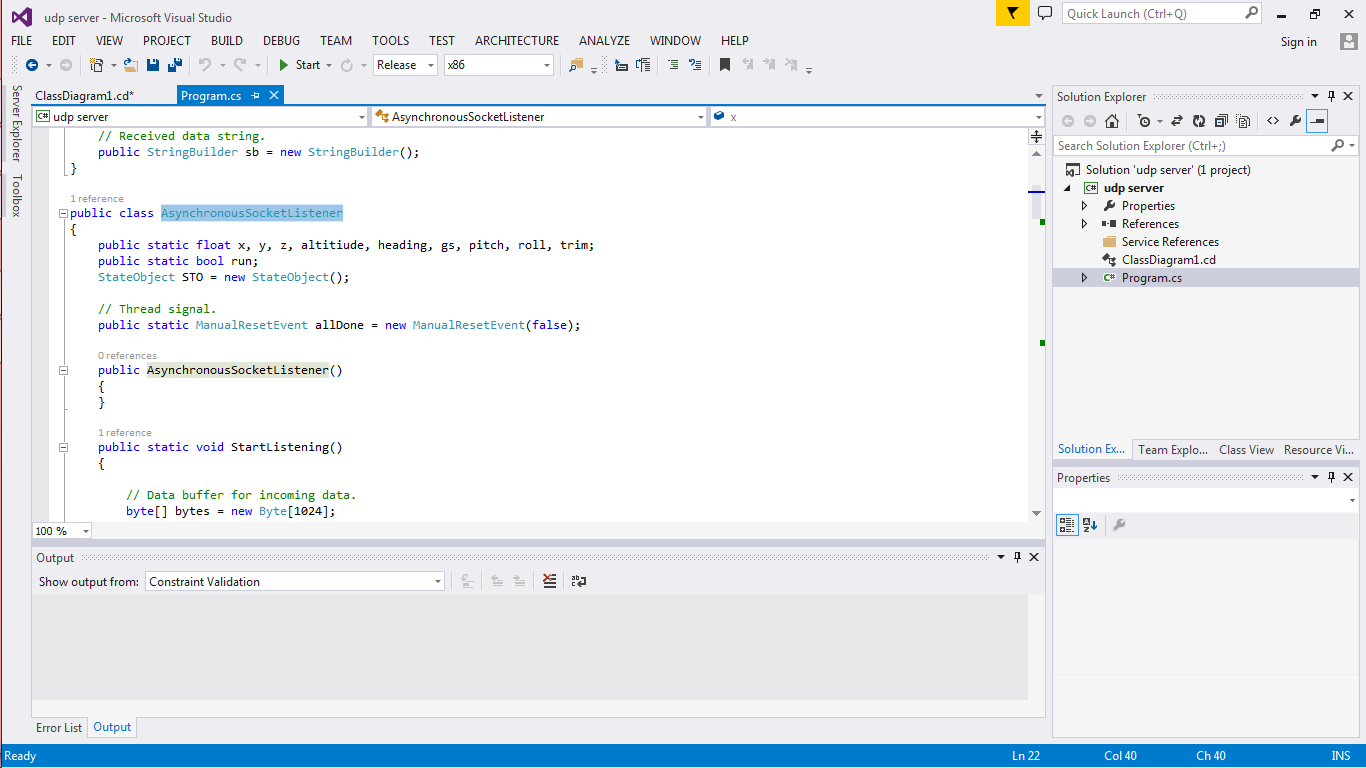
## 

## 4.2. Opis środowiska i wykorzystanych narzędzi

W celu ułatwienia i usprawnienia procesu tworzenia oprogramowania zastosowano szereg narzędzi wspomagających w postaci zewnętrznych programów i bibliotek zawierających gotowe funkcje i rozwiązania. Aplikacja korzysta ze zintegrowanego środowiska programistycznego MS Visual Studio 2013, biblioteki OpenGL, biblioteki GLM, biblioteki GLEW, biblioteki SOIL, WinAPI dostarczanego przez Microsoft, a także programu graficznego Gimp 2. Wykorzystywanie tych dodatkowych zasobów znacznie wspomaga proces tworzenia. Poniżej zamieszczono opis poszczególnych komponentów.

### 4.2.1. Microsoft Visual Studio 2013

Jest to zintegrowane środowisko programistyczne zaopatrzone we wszystkie niezbędne elementy do tworzenia kodu dla platformy Microsoft Windows x86 oraz x64. Zawiera kompilator, linker oraz Debuger. Ponadto umożliwia właściwie dowolne formatowanie kodu i szereg elementów ułatwiających pisanie tekstu (podkreślenia błędów składniowych, automatyczne proponowanie wstawienia odpowiedniej nazwy zmiennej i wiele innych rzeczy). W Visual Studio tworzony program ma postać projektu. Możliwe jest utworzenie wielu projektów dla jednej aplikacji w ramach tzw. solucji. Środowisko IDE czuwa nad lokalizacjami plików dołączanych do projektu, zawiera edytor zasobów i okien dialogowych, a także ograniczony edytor graficzny. Po poddaniu procesowi kompilacji projekt kompilowany jest do postaci Debug lub Release gdzie znajdują się pliki wykonywalne gotowe od uruchomienia przez użytkownika. Ważnym elementem środowiska VS2013 jest możliwość wygenerowania projektu pliku instalatora dzięki rozszerzeniu InstallShield. Plik tego typu jest finalną wersją dostarczaną do odbiorcy. Rysunek 4.1 przedstawia wygląd środowiska IDE MS Visual Studio 2013.



Rys 4.1 Środowisko programistyczne MS Visual Studio 2013

### 4.2.2. Gimp 2

Gimp 2 jest programem graficznym umożliwiającym wykonywanie bardzo wielu operacji na plikach graficznych różnych rozszerzeń. Posiada tzw. warstwy dzięki czemu można nakładać na siebie różne obrazy uzyskując wiele efektów. Ponadto program ten posiada bardzo przydatne funkcje do obróbki tekstur i ustawiania kanałów alfa w obrazie dzięki czemu niektóre elementy obrazu mogą być poddane w procesie renderowania tzw. mieszaniu kolorów.

### 4.2.3. Biblioteka OpenGL

Biblioteka OpenGL stanowi niskopoziomowe API grafiki komputerowej. Jest ona pośrednikiem pomiędzy aplikacją użytkownika, a sterownikiem na karcie graficznej. Użytkownik wywołuje polecenia OpenGL które realizowane są na karcie graficznej poprzez sterownik. Polega to najczęściej na przygotowaniu przez aplikację użytkownika danych np. geometrii, wysłania danych wywołania odpowiedniego polecenia OpenGL i wyrenderowania przez sprzęt graficzny przygotowanej geometrii. Biblioteka ta umożliwia wykonywanie bardzo wielu niskopoziomowych operacji na danych teksturowych (filtrowanie tekstur) dzięki językowi programowania GLSL (GL Shadow Language). GLSL wykorzystywany jest do pisania tzw. programów cieniujących (Shaderów) które kompilowane są za każdym razem gdy uruchamiana jest aplikacja na karcie graficznej. Biblioteka OpenGL umożliwia wykorzystanie tzw. mechanizmu rozszerzeń czyli dodatkowych funkcji udostępnianych przez producenta konkretnego sprzętu graficznego. W tym celu aplikacja użytkownika sama musi zapytać bibliotekę o obsługiwane rozszerzenia i pobrać wskaźniki na wybrane funkcje. Proces ten ułatwia jednak biblioteka GLEW która sama sprawdza jakie dostępne są na danym sprzęcie rozszerzenia i udostępnia je w postaci nagłówków aplikacji użytkownika.

### 4.2.4. Biblioteka GLEW

Biblioteka GLEW stanowi wspomaganie mechanizmu rozszerzeń udostępnianych przez producenta konkretnego sprzętu graficznego i udostępnianych użytkownikowi. W czasie uruchomienia programu (Run-Time) umożliwia załadowanie wybranych rozszerzeń.

### 4.2.5. Biblioteka GLM

Podczas renderowania geometrii niezbędnym elementem są przekształcenia matematyczne jak przesuwanie, obrót skalowanie i inne. Wybrany wierzchołek jest reprezentowany przez wektor zawierający trzy współrzędne kartezjańskie oraz czwartą reprezentującą stopień skalowania tych współrzędnych. Przemnożenie takiego wektora przez macierz o wymiarach 4 wiersze na 4 kolumny da w wyniku nowy przekształcony wektor. Praca z geometrią polega więc na tworzeniu odpowiedniej macierzy przesunięcia, obrotu lub skalowania i pomnożenia wierzchołka przez tę macierz. Definiowanie macierzy za każdym razem przez programistę byłoby wielce nieefektywne tak więc podczas programowania aplikacji korzystano z biblioteki matematycznej która umożliwia generowanie i zarządzanie takimi macierzami, a także wieloma innymi operacjami matematycznymi.

### 4.2.6. Biblioteka SOIL

Biblioteka ta umożliwia wczytywanie tekstur i zarządzanie nimi w pamięci operacyjnej. Pliki tekstur skompresowane do formatu .jpg są rozpakowywane do pamięci i użytkownikowi zwracany jest wskaźnik na nagłówek opisujący plik graficzny i dane. Ponadto biblioteka ta umożliwia bezpośrednie wczytanie danych tekstury do pamięci karty graficznej i zwrócenie unikalnego identyfikatora OpenGL będącego reprezentacją danej tekstury. Większość stosowanych tekstur ma format .jpg, a tam gdzie wymagana jest mała utrata danych w kompresji i dodatkowe informacje jak np. kanał alfa format .bmp.

### 4.2.7. Doxygen

Ważnym elementem jest również sporządzenie dokumentacji technicznej. W tym celu zastosowano narzędzie wspomagające tworzenie i generowanie takowej dokumentacji o nazwie Doxygen. Aplikacja ta wyszukuje automatycznie pliki źródłowe i nagłówkowe, a dzięki zastosowaniu specjalnego formatowania komentarzy w plikach źródłowych Doxygen generuje dokumentację w postaci .html oraz .pdf. Programista ma dość szeroki zakres możliwości wpływania na końcowy wygląd pliku dokumentacji. Ponadto narzędzie to umożliwia wygenerowanie wielu grafów standardu UML dołączanych do dokumentacji w celu lepszego zobrazowania funkcjonalności kodu.

## 4.3. Aplikacja odbierająca i renderująca dane

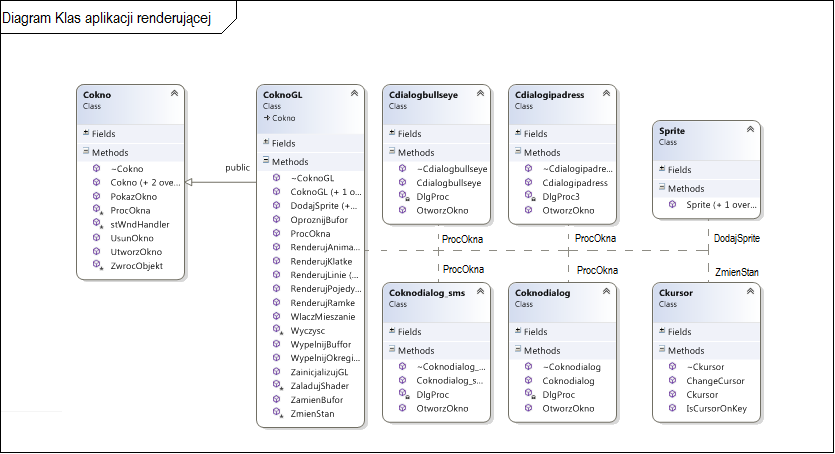
Zgodnie z założeniami aplikację tą wykonano w języku C++ . Aplikacja dzieli się na dwa równolegle pracujące wątki. Jeden wątek odpowiedzialny jest za transmisję sieciową, pobranie ramki z danymi lotu oraz zapisanie danych do pól klasy Cflighdata. Drugi, równoległy wątek odpowiada za podstawowe operacje w oknie i wyrenderowanie wybranej przez użytkownika sceny. Jego zadaniem jest również przygotowanie całej aplikacji do procesu renderowania, zainicjalizowanie wartości buforów na karcie graficznej, wczytanie do buforów geometrii oraz załadowanie tekstur. Ponadto wątek główny obsługuje wszystkie zdarzenia związane z użytkownikiem, w tym obsługa myszy oraz klawiatury.

Komponenty (pliki) z jakich składa się aplikacja renderująca:

* main.cpp - główny plik z pętla programu
* Cflighdata.cpp – plik klasy Cflighdata
* Cokno.cpp – plik klasy Cokno
* CoknoGL.cpp – plik klasy CoknoGL
* Ckursor.cpp – plik klasy Ckursor
* Cobiekt.cpp – plik klasy Cobiekt.cpp
* Cdialog.cpp – plik klasy Cdialog
* Cdialog\_sms.cpp – plik klasy Cdialog\_sms
* Cdialogipadress.cpp – plik klasy Cdialogipadress
* Cdialogbullseye.cpp – plik klasy Cdialogbullseye

Każdy plik zawierający ciało danej klasy (jej definicję) oraz plik main.cpp zawiera korespondujący plik nagłówkowy z deklaracją klasy oraz niezbędnymi typami danych.

Na poniższym rysunku został przedstawiony schematycznie diagram klas standardu UML Aplikacji MFD Simulator.



Rys 4.2 Diagram klas aplikacji renderującej

Klasa CoknoGL dziedziczy po klasie Cokno. Pozostałe klasy cechują się relacją zależności pomiędzy nimi a klasą CoknoGL. Klasy Cokno oraz CoknoGL stanowią reprezentację aplikacji – rendera. Zawierają pola zmiennych uchwytu okien, kontekstu renderowania oraz innych danych służących do manipulowania samym oknem i niskopoziomowymi operacjami generowania obrazu. Ponadto klasy te zawierają implementację metody ProcOkna, która przetwarza wiadomości wstawiane przez system operacyjny do kolejki wiadomości aplikacji, a następnie w odpowiedzi na zainstniałe zdarzenie ustawiają odpowiednie pola klasy. Dostęp do tych pól możliwy jest z poziomu pętli głównej programu, a dzięki temu w całej funkcji WinMain. Klasy okien dialogowych wykorzystywane są w metodzie ProcOkna w odpowiedzi na wybór odpowiedniego przycisku z menu kontekstowego. Ich zadanie polega na wyświetleniu okna dialogowego i zebraniu wymaganych informacji od użytkownika. Obiekty tych klas istnieją w pamięci tak długo jak istnieje instancja klasy CoknoGL. Klasa Sprite reprezentuje dowolny obiekt 2D renderowany w oknie. Przechowuje informacje o rozmiarach obiektu, położeniu i współrzędnych tekstur. Klasa CoknoGL po przekazaniu jej obiektu Sprite zapisuje dane o wierzchołkach w buforach będących jej polami statycznymi. Klasa kursora odpowiada za aktualne położenia kursora względem współrzędnychc okna oraz za sprawdzanie czy kursor znajduje się nad przyciskiem i jeśli tak to którym.

### 4.3.1. Punkt wejściowy i dyrektywy preprocesora

Punktem wejściowym od którego rozpoczyna się działanie aplikacji jest funkcja WinMain. Jej definicja znajduje się w pliku main.cpp. Na tej funkcji program kończy również swoją pracę. Wcześniej jednak pre-procesor przetwarza wszystkie dyrektywy w plikach projektu zmieniając kod. Pierwszym typem dyrektyw są instrukcje dołączenia plików nagłówkowych zawierających deklaracje funkcji, struktur, klas, a także typów wyliczeniowych.

Poniższy fragment kodu (listing 4.1) pokazuje sposób dołączania plików nagłówkowych w aplikacji.

Listing 4.1

#pragma once

#define WIN32\_LEAN\_AND\_MEAN

#include <stdlib.h>

#include <stdio.h>

#include <string>

#include <math.h>

#include <Windows.h>

#include <GL\glew.h

#include <gl\GL.h>

#include <gl\GLU.h>

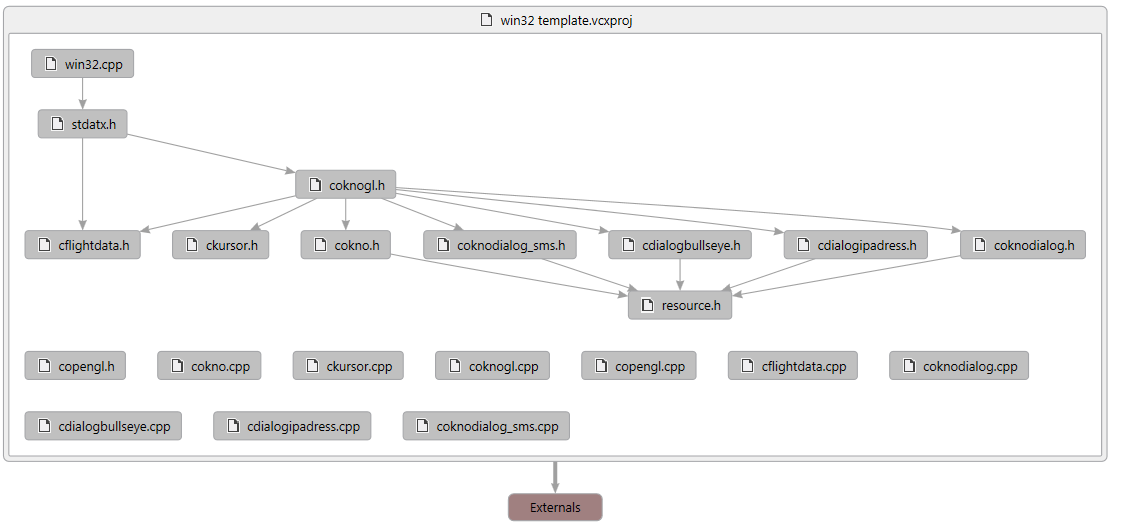
#include <SOIL.h>

#include "CoknoGL.h"

#include "Cflightdata.h"

W aplikacji nagłówki dołączane są nie w pliku main.cpp lecz w osobnym nagłówku stdatx.h w celach praktycznych. Dyrektywa #pragma once informuje kompilator, że nagłówki mają być dołączane jednokrotnie w celu uniknięcie redefinicji nazw. Dyrektywy #define WIN32\_LEAN\_AND\_MEAN służą do przyspieszenia procesu kompilacji poprzez zmniejszenie dołączanych niepotrzebnych plików nagłówkowych w nagłówku <Windows.h>.

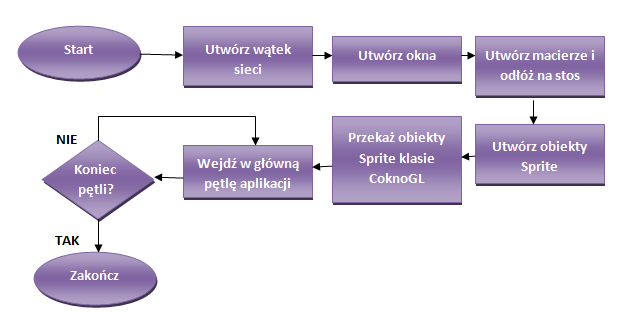
Na poniższym rysunku znajduje się diagram zależności wszystkich plików nagłówkowych w aplikacji i ich związki między sobą. Szczegółowy opis dołączonych nagłówków znajduje się w dodatku do pracy w dokumentacji technicznej.



Rysunek 4.3 Pliki nagłówkowe aplikacji renderującej

Funkcja WinMain służy jako miejsce do rozpoczęcia drugiego wątku sieciowego, definiowania przez użytkownika geometrii i macierzy przekształceń, wywołania odpowiednich metod do zainicjalizowania instancji klas CoknoGL oraz przetwarzania wiadomości aplikacji w głównej pętli programu. Funkcja główna w zależności od stanu obiektu CoknoGL kieruje program do właściwych fragmentów kodu odpowiedzialnych za kreowanie sceny.

Algorytm aplikacji przedstawiony jest na rysunku 4.4



Rys 4.4 Algorytm Aplikacji renderującej

Tworzenie wątku odbywa się poprzez wywołanie funkcji CreateThread należącej do WinAPI systemu Windows. Po tej instrukcji aplikacja tworzy dynamicznie dwie instancje klasy CoknoGL reprezentujące dwa okna i zapisuje wskaźniki do tych klas w zmiennych okno1 oraz okno2. Kolejne funkcje WinAPI – GetWindowRect oraz SetWindowPos mają za zadanie wypozycjonować na ekranie drugie okno, tak aby nie pokrywało pierwszego. Czynności te przedstawione zostały na listingu 4.2.

Listing 4.2

hUchwytWatku = CreateThread(

NULL,

0,

WatekSieci,

&flightdata,

0,

&ThreadID

);

okno1 = new CoknoGL(MFD\_MAIN);

okno1->UtworzOkno();

okno2 = new CoknoGL(MFD\_BLANK);

okno2->UtworzOkno();

RECT rect;

ZeroMemory(&rect, sizeof(rect));

GetWindowRect(okno2->hUchwytOkna, &rect);

SetWindowPos(okno2->hUchwytOkna, NULL, 585, 0, rect.right-rect.left, rect.bottom-rect.top, 0);

Tworzenie i dodawanie obiektów Sprite reprezentujących obiekt 2D w oknie odbywa się poprzez utworzenie instancji klasy Sprite i przekazaniu do jej konstruktora porządanych właściwości obiektu (położenie X i Y, współrzędne tekstur U i V oraz szerokość i wysokość). Następnie obiekt klasy Sprite przekazywany jest do klasy CoknoGL poprzez wywołanie jej metody DodajSprite. Opis konstruktora parametrycznego klasy Sprite oraz metody DodajSprite klasy CoknoGL dodano do dokumentacji technicznej załączonej do niniejszej pracy.

Kolejnym etapem w działaniu aplikacji jest rozpoczęcie głównej pętli programu w której pobierane są wiadomości z kolejki wiadomości aplikacji, wybierane sceny renderowania oraz obliczenia aktualnych parametrów. Pętlę aplikacji oraz klasę CoknoGL omówiono w kolejnym podrozdziale.

Gdy w kolejce wiadomości pojawi się wiadmość WM\_QUIT główna pętla programu jest przerywana i aplikacja kończy swoje działanie. Przed zwróceniem wartości do systemu wywoływane są metody obiektów okno1 oraz okno2 usuwające wszystkie składniki wykorzystywane przez aplikację i zwalniające zasoby pamięci. Po tych czynnościach zwraca wartość do systemu i kończy swoje wszystkie wątki w tym wątek sieciowy utworzony na początku funkcji WinMain.

### 4.3.2. Główna pętla aplikacji i klasa okna

Główna pętla programu pełni funkcję zapewniającą jego ciągłą pracę. W pętli wywoływane są funkcje systemowe jak PeekMessage oraz DispatchMessage, które pobierają wszystkie wiadomości jakie system umieści w kolejce wiadomości aplikacji i wysyłają je do procedury przetwarzającej klasy odpowiedniego okna. Procedura ta jest rejestrowana w systemie wraz z zarejestrowaniem klasy okna. W przypadku aplikacji symulatora procedura ta jest w istocie metodą klasy CoknoGL – jest wywoływana za każdym razem gdy pojawi się jakakolwiek wiadomość okna aplikacji. Pętla wykonuje się w sposób nieprzerwany – jedynym warunkiem jej opuszczenia jest umieszczenie w kolejce wiadomości wartości WM\_QUIT.

Dodatkowymi elementemi wykonywanymi w trakcie trwania głównej pętli programu są dodatkowe obliczenia takich wartości jak położenie wskaźnika kursora ekranu z radarem, przeliczenie jednostek odległościowo – kątowych na jednostki okna systemowego, wyliczenie zmiennych animacji przejść, ustanowienie napisu na belce tytułowej okna, sprawdzenie stanu zmiennych użytkownika, a przede wszystkim wywołanie procedury WybierzScene która służy do wybrania na podstawie wewnętrznego stanu obiektu okna odpowiedniej sceny (zakładki, submenu).

Fragment procedury wyboru renderowania sceny przedstawia listing 4.2

Listing 4.2

void WybierzScene()

{

switch (okno1->stan)

{

case MFD\_MAIN:

RysujMain(okno1);

break;

case MFD\_BLANK:

RysujBlank(okno1);

break;

case MFD\_HAD:

RysujHad(okno1);

break;

case MFD\_RESET:

RysujReset(okno1);

break;

case MFD\_SMS:

RysujSms(okno1);

break;

case MFD\_HSD:

RysujHsd(okno1);

break;

case MFD\_HSD\_CEN:

RysujHsdCen(okno1);

break;

case MFD\_HSD\_CPL:

RysujHsdCpl(okno1);

break;

// …

Jak pokazano na listingu stan wewnętrzny obiektu reprezentowany jest przez typ wyliczeniowy E\_STAN o wartościach MFD\_XX gdzie XX oznacza aktualną zakładkę wybraną przez użytkownika. Procedura WybierzScene na podstawie tego stanu wywołuje metodę rysującą i przekazuje do niej jako jedyny parametr wskaźnik do klasy okna. Takich procedur jest niemal tyle samo ile możliwych wartości typu wyliczeniowego (niektóre jednak powtarzają się i z uwagi na oszczędność kodu nie powielano funkcji lecz wywoływano im odpowiadające). Funkcje odpowiedzialne za kompozycję sceny są różne w zależności od typu stanu jaki muszą wyrenderować. Posiadają jednak wspólny schemat wykonywania podstawowych czynności. Czynnościami tymi są:

* Wyczyszczenie zapisu z bufora obrazu (pozbycie się poprzedniej klatki)
* Ustanowienie jednej lub wielu w zależności od potrzeby macierzy przekształceń
* Wywołanie metody klasy okna renderującej wskazany indeksem obiekt (obiekty) Sprite oraz teksturę, wraz z ewentualnym przekształceniem geometrii
* Wyrenderowanie kanałów alfa, beta oraz gamma ( będącymi stanem tzw. „ szybkich zakładek” które użytkownik może programować i błyskawicznie przełączać się pomiędzy nimi
* Wyrenderowanie animacji przejścia jeśli ustawiono pole Animacja klasy CoknoGL
* Wyrenderowanie ramki stanowiącej alternatywny interfejs GUI z użytkownikiem

Przykładową funkcję układającą scenę dla zakładki MFD\_DTE (Data Terminal Equipement ) przedstawia poniższy listing.

Listing 4.3

void RysujDte(CoknoGL\* obiekt)

{

static float Animacja = 80.1f;

if (obiekt->DteLoad)

{

Animacja -= 0.5f;

}

obiekt->OproznijBufor();

obiekt->RenderujKlatke(6, 7);

glm::mat4 mat1 = glm::translate(stosmacierzy[3], glm::vec3(80, 400, 0.0f));

if ((Animacja < 80.0f) & (Animacja > 70.0f)) obiekt->RenderujPojedynczyObiekt(186, 186, mat1);

if ((Animacja < 70.0f) & (Animacja > 60.0f)) obiekt->RenderujPojedynczyObiekt(187, 186, mat1);

if ((Animacja < 60.0f) & (Animacja > 50.0f)) obiekt->RenderujPojedynczyObiekt(188, 186, mat1);

if ((Animacja < 50.0f) & (Animacja > 40.0f)) obiekt->RenderujPojedynczyObiekt(189, 186, mat1);

if ((Animacja < 40.0f) & (Animacja > 30.0f)) obiekt->RenderujPojedynczyObiekt(190, 186, mat1);

if ((Animacja < 30.0f) & (Animacja > 20.0f)) obiekt->RenderujPojedynczyObiekt(191, 186, mat1);

if ((Animacja < 20.0f) & (Animacja > 10.0f)) obiekt->RenderujPojedynczyObiekt(192, 186, mat1);

if ((Animacja < 10.0f) & (Animacja > 0.0f)) obiekt->RenderujPojedynczyObiekt(193, 186, mat1);

if (Animacja < 0.0f)

{

obiekt->RenderujPojedynczyObiekt(186, 186, mat1);

obiekt->RenderujPojedynczyObiekt(187, 186, mat1);

obiekt->RenderujPojedynczyObiekt(188, 186, mat1);

obiekt->RenderujPojedynczyObiekt(189, 186, mat1);

obiekt->RenderujPojedynczyObiekt(190, 186, mat1);

obiekt->RenderujPojedynczyObiekt(191, 186, mat1);

obiekt->RenderujPojedynczyObiekt(192, 186, mat1);

obiekt->RenderujPojedynczyObiekt(193, 186, mat1);

}

float AnimacjaPrzejscia = 0.0f;

obiekt->WlaczMieszanie(TRUE);

if (obiekt->hUchwytOkna == okno1->hUchwytOkna)

AnimacjaPrzejscia = AnimacjaPrzejsciaOkno1;

else AnimacjaPrzejscia = AnimacjaPrzejsciaOkno2;

if (AnimacjaPrzejscia != 1.0) obiekt->RenderujAnimacje(3, stosmacierzy[3], glm::vec3(1, 0, 0), AnimacjaPrzejscia);

if(obiekt->b\_ramka)

obiekt->RenderujRamke(0);

RysujKanaly(obiekt);

obiekt->WlaczMieszanie(FALSE);

obiekt->ZamienBufor();

if (Animacja <= -5.0f)

{

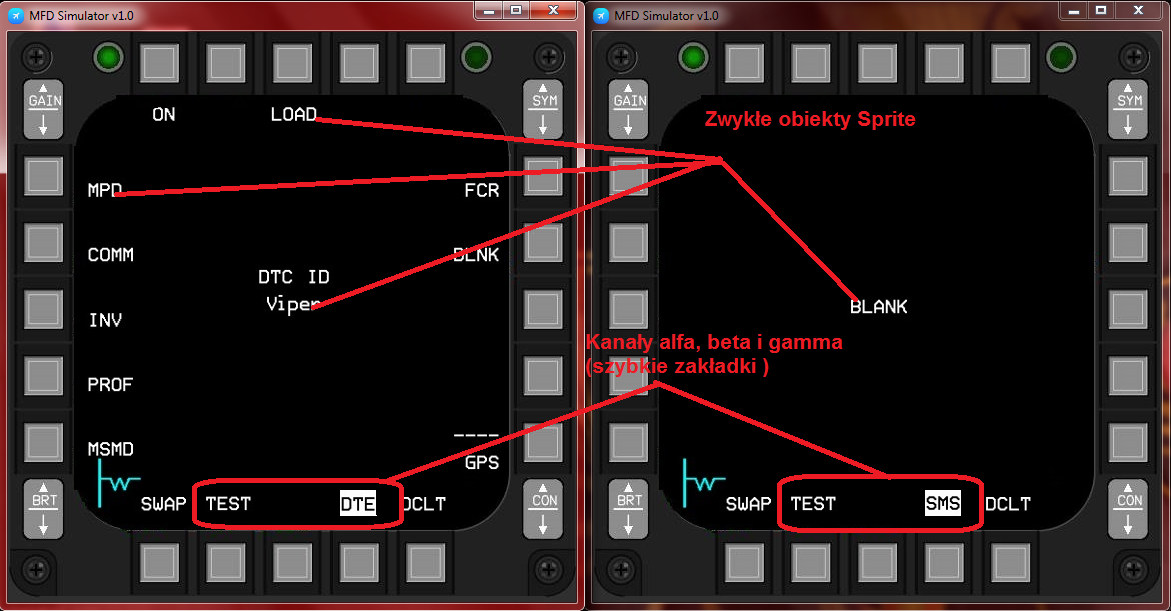
Animacja = 80.1f;

obiekt->DteLoad = FALSE;

}

}

Na początku funkcji zadeklarowano zmienną statyczną „Animacja” reprezentującą animację renderowaną podczas ładowania danych przez DTE. Gdy w klasie okna ustawiony jest znacznik DteLoad wartość zmiennej Animacja zmienia się za każdym razem wywołania funkcji RysujDte. W ten sposób możliwe jest sekwencyjne renderowanie wybranych obiektów na scenie i sprawienie wrażenia ich sekwencyjnego pojawiania się i znikania. Gdy wartość zmiennej Animacja osiągnie wartość mniejszą niż -5.0f, wartość ta jest ustawiana na swój stan początkowy, a znacznik animacji w klasie okno zerowany. Metoda obiektu okna RenderujKlatke(6,7) renderuje jeden obiekt Sprite o indeksie 6 oraz odpowiadającą mu teksturę. Za pomocą biblioteki glm możliwe jest utworzenie macierzy przekształceń (przesunięcie, obrót lub skalowanie) i przekazanie jej w funkcji RenderujPojedynczyObiekt. Funkcja ta jako parametry przyjmuje numer obiektu Sprite, numer tekstury oraz macierz przekstałcenia. W ten sposób tworząc odpowiednie macierze i przemieszczając obiekty na scenie modeluje się wygląd sceny. Funkcja RenderujKanaly działa podobnie jak funkcje rysujące sceny jednak odpowiada ona za wyrenderowanie kanałów szybkich zakładek. Funkcja RenderujRamke jest unikatowa, służy do wyrenderowania ramki stanowiącej interfejs GUI – ważnym elementem jej towarzyszącym jest włączenie mieszania kolorów poprzez metodę klasy okna i pozbycie się wnętrza ramki. Ramka ta jest renderowana jedynie wówczas gdy użytkownik włączył tryb renderowania alterantywnej ramki MFD (klawisz H). Na sam koniec wywoływana jest metoda ZmienBufor klasy Okno pozwalająca zamienić bufor obrazu tylny z buforem przednim (domyślnie biblioteka OpenGL w dwubuforowym kontekście urządzenia renderuje obraz do bufora tylnego). Ostateczny wygląd sceny DTE przedstawia Rysunek 4.5.



Rys 4.5 Dwa okna aplikacji wraz z obiektami Sprite, kanałami alfa, beta i gamma oraz ramką

Klasa okna jest największą klasą w aplikacji i pełni w niej najważniejszą rolę. Dziedziczy ona po klasie Cokno dodając dodatkowe funkcjonalności związane z biblioteką graficzną OpenGL. Zawiera w sobie bardzo dużo pól i metod odpowiedzialnych za operacje renderowania, zarządzanie geometrią, przetwarzanie procedury obsługi wiadomości Jej działanie polega na udostępnianiu użytkownikowi niewielkiego interfejsu dzięki któremu wywołując odpowiednią metodę może obsługiwać podstawowe operacje w oknie bez wydawania niskopoziomowych poleceń bibliotek.

Z punktu widzenia użytkownika klasa ta działa jak maszyna stanu. Gdy zaistnieje określona sytuacja np. Użytkownik wybierze przyciskiem myszy inną zakładkę klasa najpierw przetworzy wiadomość myszy i w zależności od kontekstu w jakim została wygenerowana ustawi swój wewnętrzny stan na pożądany przez użytkownika. Aplikacja w głównej pętli programu sprawdza ustawiony stan i reaguje tak jak zostało to wyjaśnione w poprzedniej części tego podrozdziału. W istocie stan aplikacji jest zbiorem zmienny (pól klasy) które są ustawiane na odpowiednie wartości.

Pola klasy okna reprezentujące stan wewnętrzny podzielono na ogólne – wykorzystywane zawsze w pętli programu oraz dodatkowe oznaczające stan konkretnej części aplikacji (np. Wyświetlanie menu w zakładce HSD lub tryb pracy radaru). Opis wszystkich zmiennych pól klasy CoknoGL zamieszczono w dokumentacji.

Pierwszą metodą wywoływaną po konstruktorze klasy i metodzie utworzenia okna jest metoda inicjalizacji sterownika OpenGL oraz przystosowanie okna do renderowania w jego obszarze.:

void CoknoGL::ZainicjalizujGL()

{

//

}

Działanie tej metody składa się z kilku etapów. Pierwszym z nich jest znalezienie odpowiedniego deskryptora pikseli dla danego systemu. Gdy system udostępnia odpowiedni deskryptor, aplikacja ustawia go jako domyślny dla okna i inicjalizuje bibliotekę glew (udostępniającą predefiniowane wskaźniki na rozszerzenia sterownika OpenGL).

Kolejną czynnością wykonywaną przez program jest utworzenie buforów na karcie graficznej w celu przechowywania danych geometrii obiektów Sprite, współrzędnych tekstur oraz samych obiektów tekstur. Aplikacja robi to poprzez wywołania funkcji glGenBuffer oraz glBindBuffer. Tekstury wczytywane są z dysku do pamięci operacyjnej przy pomocy biblioteki SOIL. Program wczytuje do pamięci także tzw. Shadery czyli programy wykonywane na karcie graficznej do przetwarzania potokowego geometrii.

Gdy biblioteka OpenGL została zaincjalizowana dla danego okna aplikacja zwalnia kontekst renderowania za pomocą funkcji wglMakeCurrent.

Wszystkie zwrócone przez bibliotekę identyfikatory przechowywane są w polach klasy CoknoGL. Jakiekolwiek operacje związane z biblioteką OpenGL wymagają najpierw ustawienia kontekstu renderowania za pomocą funkcji wglMakeCurrent i następnie zwolnienia go. Pola klasy związane z identyfikatorami i biblioteką są unikalne dla każdego okna, zaś pola związane ze stanem wewnętrznym aplikacji (np. Aktualnym trybem radaru itp.) są polami statycznymi niezależnymi od instancji klasy.

Kolejnym elementem klasy okna jest procedura obsługi wiadomości wysyłanych przez system. Wiadmości wysyłane są poprzez funkcje DispatchMessage znajdujące się w pętli głównej programu. Nagłówek procedury obsługi wiadomości wygląda następująco

LRESULT CALLBACK CoknoGL::ProcOkna(HWND hwnd, UINT umsg, WPARAM wparam, LPARAM lparam)

{

//

}

Parametr hwnd to uchwyt do okna do którego wysyłana jest wiadomość, umsg to kod wiadomości, a wparam i lparam określają dodatkowe parametry opisujące wiadomość. Obsługa wiadomości polega na zaimplementowaniu instrukcji switch, która w zależności od przechwyconej wiadomości modyfikuje stan wewnętrzny klasy. W procedurze obsługi wiadomości generowane są także okna dialogowe jako odpowiedź na wybraną przez użytkownika opcję z menu kontekstowego.

Klasa zawiera także kilka funkcji renderujących geometrię (obiekty Sprite) w zależności od potrzeb. Ich nagłówki przedstawiono poniżej:

Listing 4.4

void RenderujKlatke(GLint nStartindex, GLint nEndindex);

void RenderujRamke(int i);

void RenderujAnimacje(int nIndexObiektu, glm::mat4 macierzmvp, glm::vec3 color, float przezroczystosc);

void RenderujPojedynczyObiekt(int nIndexObiektu, int nIndexTextury);

void RenderujPojedynczyObiekt(int nIndexObiektu, int nIndexTextury, glm::mat4 macierztr);

void WypelnijBuffor();

void RenderujLinie(glm::mat4 macierz, glm::vec3 color, int start, int end,

glm::vec3 skala);

void RenderujLinie(glm::mat4 macierz, glm::vec3 color, int start, int end,

glm::vec3 skala, float fGrubosc);

void WypelnijOkregiem(float fPromien);

Funkcje te przyjmują podobne parametry jednak ich przeznaczenie jest odmienne. Funkcja RenderujKlatke renderuje pojedyńczy obiekt Sprite lub kilka kolejnych w zakresie od nStartindex do nEndindex. Obiekty renderowane są wraz z odpowiadającymi im indeksami tekstur ( obiekt nr 3 – tekstura nr 3 itd. ). RenderujAnimacje przeznaczona jest do wyrenderowania animacji płynnych przejść pomiędzy zmieniającymi się zakładkami. Wykorzystuje ona odmienne Shadery w odróżnieniu od pozostalych funkcji. Jako parametry przyjmuje indeks obiektu Sprite, macierz transformacji, kolor obiektu oraz jego przezroczystość. Funkcja RenderujPojedynczyObiekt rysuje pojedyńczy Sprite o numerze nIndexObiektu oraz o teksturze nIndexTekstury. Dzięki przeciążeniu funkcji możliwe jest także przekazanie do funkcji jako parametru macierzy transformacji. Funkcje RenderujLinie pełnią podobą rolę jednak w odróżnieniu od poprzednich funkcji nie renderują trójkątów lecz obiekty linii GL\_LINES. Funkcja WypelnijOkregiem wypełnia bufor klasy CoknoGL współrzędnymi okręgu o podanym przez użytkownika promieniu fPromien. Funkcja WypelnijBufor przesyła aktualne dane jakie znajdują się w buforze klasy okna do sterownika OpenGL.

Metodą odpowiedzialną za logikę aplikacji jest metoda ZmienStan.

Listing 4.5

void CoknoGL::ZmienStan(E\_KEY klawisz)

{

//

}

Metoda ta jest wywoływana za każdym razem gdy użytkownik naciśnie przyciskiem myszy w obszarze okna. Wówczas w procedurze przetwarzającej wiadomości inny obiekt o nazwie Ckursor oblicza w którym miejscu nastąpiło kliknięcie i jeśli nastąpiło to w obrębie przycisku na ramce, obiekt zwraca stan w jakim powinna znaleźć się aplikacja i przekazuje go do metody ZmienStan. W metodzie ZmienStan w zależności od tego w jakim stanie poprzednio znajdował się obiekt, jaki jest stan kanałów itp. ustawiany jest stan wewnętrzny klasy. W ten sposób w nowym cyklu głównej pętli programu aplikacja pobiera aktualny stan w jakim powinien znajdować się wskaźnik i na tej podstawie komponuje scene jak przedstawiono to poprzednio.

Obok wymienionych powyżej metod klasa okna zawiera dodatkowe pomocnicze, krótkie metody inline jak np. zmiana buforów obrazu, włączenie i wyłączenie funkcji mieszania itp. Z uwagi na to, że klasa wykorzystuje mechanizmy dziedziczenia zawiera destruktor oznaczony słowem kluczowym virtual.



Rys 4.6. Podstawowe kluczowe komponenty (metody) klasy CoknoGL

### 4.3.3. Zarządzanie teksturami i geometrią

Ważnym elementem w aplikacji renderującej jest sprawne dodawanie geometrii do buforów na karcie graficznej, tworzenie macierzy przekształceń oraz zarządzanie teksturami. Jak wspomniano w poprzedniej części podrozdziału obiekty na scenie reprezentuje klasa Sprite do której wraz z tworzeniem obiektu przekazywane są dane współrzędnych wierzchołków i tekstur. Konstruktor obiektu Sprite ma nagłówek przedstawiony na listingu poniżej:

Listing 4.6

class Sprite

{

public:

Sprite();

Sprite(GLfloat x, GLfloat y, GLfloat u, GLfloat v, GLfloat szerokosc, GLfloat wysokosc);

std::vector<GLfloat> wierzcholki;

std::vector<GLfloat> wspolrzedneuv;

};

Klasa ta w całości oblicza wymagane przez bibliotekę współrzędne na podstawie parametrów przekazanych przez użytkownika i zapisuje je w swoich polach. Po przekazaniu obiektu Sprite do klasy okna (CoknoGL), obiekty te dodawane są do buforów na karcie graficznej, a identyfikatory do poszczególnych obiektów zapisywane w tablicy w klasie CoknoGL. Tablica ta może pomieścić do 250 identyfikatorów do obiektów.

Przekształcenia geometrii polegają na wygenerowaniu odpowiedniej macierzy oraz przekazaniu tej macierzy do programu cieniującego, który mnoży każdy wierzchołek przez macierz przekształcenia, otrzymując w wyniku nowe współrzędne wierzchołka. Wierzchołki w języku C++ mogą być reprezentowane przez jednowymiarowy wektor do którego zapisywane są kolejno współrzędne X, Y, Z oraz ewentualna wspólrzędna W skalująca trzy poprzednie. Generowaniem macierzy przekształceń zajmuje się biblioteka GLM. W tym celu udostępnia ona funkcje generujące macierz, macierze translacji (przesunięcia), obrotu czy skalowania. Duże znaczenie ma kolejność wykonywanych przekształceń gdyż różna kolejność da w wyniku inny wygląd sceny. Poniższy listing przedstawia funkcje biblioteki GLM które wykorzystano w aplikacji.

Listing 4.7

glm::mat4 macierz1 = glm::mat4(1.0f); macierz1 = glm::translate(macierz1, glm::vec3(-30.0f, 0.0f, 0.0f));

macierz1 = glm::rotate(macierz1, 45.0f, glm::vec3(0.0f, 0.0f, 1.0f));

Ostatnią czynnością wykonywaną w przekształceniach geometrii jest przesłanie macierzy do programu Shadera. W tym celu w odpowiedniej funkcji renderującej (np. RenderujPojedynczyObiekt ) jako parametr przekazywana jest macierz. Macierz jest stała w całej serii wierzchołków więc aplikacja przekazują ją w postaci zmiennej uniform (Wykorzystane programu cieniujące omówione są w dalszej części podroździału).

Dane teksturowe poddano procesowi obróbki w programie graficznym Gimp 2. Po dopasowaniu wymiarów i zapisaniu danych z odpowiednim rozszerzeniem aplikacja wczytuje dane podczas wywołania metody ZainicjujGL. Tekstury umieszczono w oddzielnym katalogu dla zachowania przejrzystości segregacji komponentów programu. Do wczytywania danych tekstur zastosowano bibliotekę SOIL, która obsługuje różne rozszerzenia plików graficznych (w tym .jpg i .bmp). Nazwy plików wraz ze ścieżkami przekazywane są poprzez funkcję DodajSprite. Klasa okna wewnętrznie tworzy bufor nazw tekstur z którego w pętli biblioteka SOIL ładuje dane prosto do pamięci karty graficznej. Klasa CoknoGL posiada także bufory identyfikatór które zwracane są przez bibliotekę (w tym identyfikatorów zwracanych przez funkcje ładujące tekstury). W momencie wywołania funkcji renderującej i przekazaniu do niej odpowiedniego parametru biblioteka szuka obiektu Sprite po identyfikatorze z tablicy identyfikatorów i odpowiedniej tekstury z tablicy identyfikatorów tekstur. W efekcie możliwe jest renderowanie obiektu Sprite z różnymi danymi tekstur.

Biblioteka OpenGL obsługuje tekstury wielopoziomowe jednak w aplikacji nie zastosowano tej metody generowania obrazu z uwagi na większą komplikację kodu źródłowego. W większości tekstur zastosowano równanie mieszania (GL\_BLEND), które usuw czarne piksele z tekstury dając efekt całkowitej przezroczystości czarnych powierzchni.

Listing 4.8

glEnable(GL\_BLEND);

glBlendFunc(GL\_ONE, GL\_ONE\_MINUS\_SRC\_ALPHA);

glDisable(GL\_BLEND);

Efekt ten uzyskano implementując w programie Shadera fragmentów równanie dodające do czarnych pikseli ( RGB(0,0,0)) zerowy kanał alfa ( RGB(0,0,0,0)). Równanie mieszania wykorzystano w animacji płynnego przejścia pomiędzy zakładkami. Najpierw wyrenderowano scenę i jej elementy (bez ramki), następnie czarny obiekt ze zmienną przekazywaną do Shadera, a na końcu ramkę. Zmieniająca się w czasie wartość zmniennej sprawiała, że obiekt stawał się wraz z upływającym czasem przezroczysty odsłaniając stopniowo scenę (ramka pozostała całkowicie nieprzezroczyta). Dzięki temu użytkownik ma wrażenie płynnego pojawienia się elementów przełączanej zakładki.

### 4.3.4. Obsługa sterowania użytkownika

Użytkownik generuje sygnały determinujące wygląd calej sceny. Głównym ich źródłem jest ramka MFD Coughar oraz (alternatywnie) mysz za pomocą której może wybierać poszczególne zakładki i opcje z menu kontekstowego. Z tego też względu źródła sygnałów sterujących które zastosowano w aplikacji podzielono na:

* Sterowanie zakładkami poprzez przyciski na ramce lub LPM
* Sterowanie menu kontekstowym poprzez prawy i lewy przycisk myszy
* Sterowanie wskaźnikiem kursora radaru, wyborem SOI wskaźnika MFD oraz wyborem i zwolnieniem celu śledzenia radaru poprzez klawiaturę

Użytkownik po umieszczeniu kursora nad jednym z przycisków ramki może zaobserwować zmianę kursora myszy sugerującą możliwość wyboru. Po naciśnięciu LPM klasa okna zmienia swój stan wewnętrzny na wybrany przez użytkownika i dzięki temu możliwy jest wybór funkcji komponowania sceny z poziomu pętli głównej programu.

Wiadomościami wstawianymi do kolejki wiadomości aplikacji po zdarzeniu naciśnięcia przycisku myszy lub przemieszczania wskaźnika nad oknem to WM\_LBUTTONDOWN, WM\_LBUTTONUP, WM\_RBUTTONDOWN oraz WM\_MOVE . Poniższy listing przedstawia metodę obsługi wyżej wymienionych wiadomości.

Listing 4.9

case WM\_LBUTTONUP:

wglMakeCurrent(hUchwytKontekstu, hUchwytRendera);

SetCursor(cursor);

klawisz = kursor.IsCursorOnKey(lparam);

ZmienStan(klawisz);

SetFocus(hwnd);

wglMakeCurrent(NULL, NULL);

break;

case WM\_LBUTTONDOWN:

SetCursor(cursor);

break;

case WM\_RBUTTONDOWN:

GetWindowRect(hwnd, &rect);

TrackPopupMenu(

hUchwytSubMenu,

TPM\_LEFTALIGN | TPM\_TOPALIGN,

rect.left +LOWORD(lparam),

rect.top +HIWORD(lparam),

0,

hUchwytOkna,

0);

break;

case WM\_MOUSEMOVE:

klawisz = kursor.IsCursorOnKey(lparam);

kursor.ChangeCursor(klawisz);

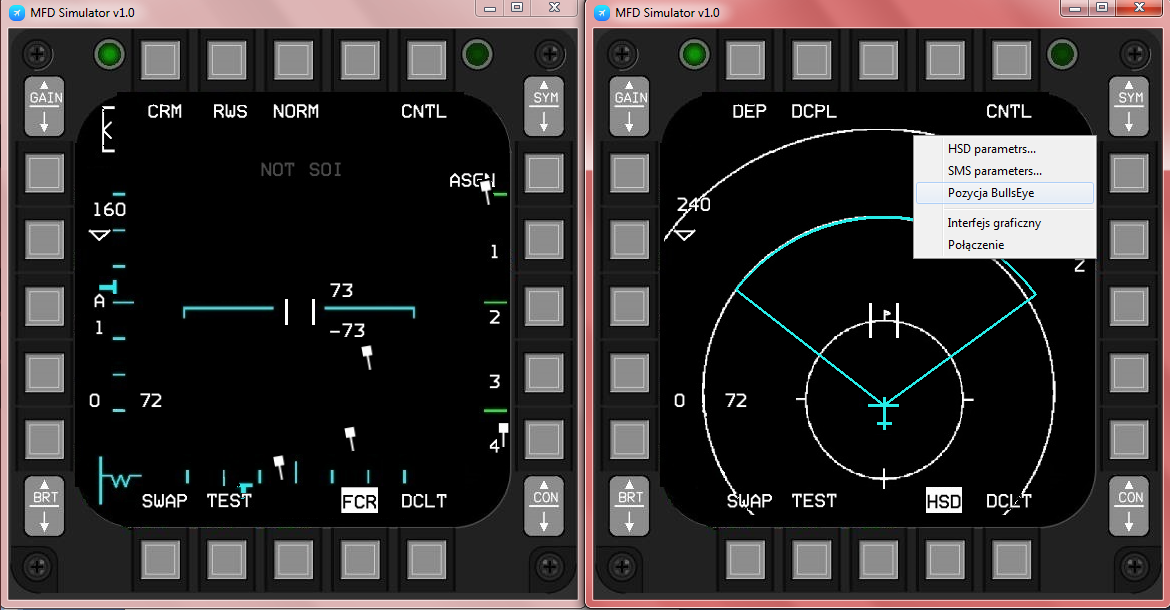
cursor = GetCursor();

break;

Funkcja klasy Ckursor::IsCursorOnKey sprawdza czy wskaźnik myszy znajduje się nad którymś z przycisków ramki. Jeśli użytkownik tylko przemieszcza wskaźnik kursora myszy w wiadomości WM\_MOSEMOVE zmieniana jest na ten czas ikona kursora. Jeśli użytkownik wybierze LPM wywoływana jest funkcja ZmienStan z parametrem nowego stanu wewnętrznego okna. W przypadku naciśnięcia RPM (kod obsługi tej wiadomości znajduje się w załączniku) użytkownik uruchamia menu kontekstowe. Gdy podejmie jakąś czynność w jego obrębie (np. wciśnie przycisk myszy) do okna rodzica wysyłana jest wiadomość WM\_COMMAND. Na rysunku 4.5 pokazano sytuację w której użytkownik uruchomił menu kontekstowe i ustawił wskaźnik kursora myszy nad jedną z jego opcji. Jeśli w tym momencie naciśnąłby LPM do kolejki wiadomości wstawiona by została wiadomość WM\_COMMAND z identyfikatorem odpowiedniego menu z menu kontekstowego i możliwe stało by się obsłużenie tego zdarzenia (np. wyświetlenie okna dialogowego).

Kolejnym rodzajem sygnałów zaimplementowanych w aplikacji jest klawiatura. Za jej pomocą użytkownik może przemieszczać kursor radaru, przełączać tryb SOI wybranego MFD (Sensor of Interest) oraz zaznaczać i odznaczać śledzony cel radaru. Obslugę klawiatury zastosowano w sposób zbliżony do obsługi myszy. W tym przypadku system również wstawia do kolejki wiadomości wiadomość WM\_KEYDOWN oraz WM\_KEYUP, które zostały obsłużone w procedurze ProcOkna. Klasa CoknoGL posiada odpowiednie znaczniki, np. gdy użytkownik naciśnie klawisz strzałki ustawiany jest znacznik przesunięcia kursora. Wówczas w każdym cyklu pętli programu do zmiennej reprezentującej położenie kursora dodawana jest lub odejmowana wartość. Gdy użytkownik zwolni klawisz znacznik ustawiany jest na początkową wartość i zmienna kursora nie jest dalej modyfikowana. W tabeli 4.2 zestawiono sygnały wejścia pochodzące od użytkownika i ich przeznaczenie.

W przypadku posiadania przez użytkownika ramek MFD Coughar aplikacja uzyskuje dostęp do stosu sterowników urządzeń USB klasy HID. Poprzez sterownik możliwe są do odczytania deskryptory urządzeń, raportów, interfejsów oraz punktów końcowych. Funkcją odpowiedzialną za te czynności oraz za wygenerowanie odpowiednich wiadomości dla okien poprzez funkcję SendMessage, jest funkcja GetMFDpushButton opisana w dokumentacji.



Rys 4.7. Menu kontekstowe, ramka z przyciskami wyboru oraz radar z kursorem

Sygnały pochodzące od użytkownika Tabela 4.2

|  |  |  |
| --- | --- | --- |
| **Sygnał sterujący** | **Wiadomość obsługiwana w procedurze okna** | **Przeznaczenie sygnału sterującego** |
| **Naciśnięcie lub zwolnienie lewego lub prawego przycisku myszy (LPM)** | WM\_LBUTTONDOWN  WM\_LBUTTONUP  WM\_RBUTTONDOWN  WM\_RBUTTONUP | Wybór zakładki (menu) MFD  Włączenie menu kontekstowego  Wybór opcji z menu kontekstowego  Wyłączenie menu konktekstowego |
| **Ruch myszy** | WM\_MOUSEMOVE | Obliczenie położenia kursora  Zmiana kursora (jeśli nad przyciskiem) |
| **Klawisz**  **Spacji** | WM\_KEYDOWN / WM\_KEYUP  Identyfikator : VK\_SPACE | Wybór SOI wskaźnika MFD |
| **Klawisze strzałki** | WM\_KEYDOWN / WM\_KEYUP  Identyfikatory: VK\_LEFT, VK\_RIGHT, VK\_UP, VK\_DOWN | Ruch kursora radaru |
| **Klawisz F1** | WM\_KEYDOWN / WM\_KEYUP  Identyfikator: VK\_F1 | Wybór obiektu śledzenia radaru |
| **Klawisz F2** | WM\_KEYDOWN / WM\_KEYUP  Identyfikator: VK\_F2 | Odrzucenie obiektu śledzenia radaru |

### 4.3.5 Klasa parametrów lotu i wątek transmisji sieciowej

Klasa odpowiedzialna za przechowywanie i manipulowanie danymi z których korzysta aplikacja to klasa Cflightdata. Jako jedyna jej instancja zadeklarowano obiekt w zmiennej globalnej, dzięki czemu dostęp do aktualnych parametrów umożliwiono w obrębie całego pliku main.cpp. Egzemplarz tej klasy przekazywany jest również do klasy CoknoGL. Część parametrów możliwa jest do zdefiniowania w obrębie samej aplikacji MFD. Użytkownik poprzez wybór z menu kontekstowego opcji HSD parameters może zdefiniować podstawowe parametry nawigacyjne takie jak punkty trasy, predefiniowane zagrożenia i ich zasięg lub dodatkowe linie pomocnicze wyświetlane na panelu HSD. W ramach menu kontekstowego SMS parameters użytkownik może określić jaki typ uzbrojenia jest podwieszony do belek samolotu. Wówczas aplikacja na bierząco pokazuje stan uzbrojenia w zakładce SMS.

Drugim rodzajem parametrów są parametry przesyłane przez sieć do których dostep aplikacja uzyskuje po nawiązaniu połączenia z aplikacją serwera. Po uzyskaniu bufora Aplikacja renderująca wypełnia swoje pola klasy Cflightdata nowymi danymi. Klasa stanowi połączenie pomiędzy dwoma wątkami poprzez które wymieniane są dane. Nagłówek klasy Cflightdata zobrazowany został na listingu 4.10.

Listing 4.10

class Cflightdata

{

public:

Cflightdata();

Cflightdata(char\* ipadress);

virtual ~Cflightdata();

std::string adres;

std::string port;

WSADATA wsadata;

SOCKET ConnectSocket;

struct addrinfo \*result, \*ptr, hints;

std::string bufor;

char recvbuf[DEFAULT\_BUFLEN];

int iResult;

int recvbuflen;

int nRadarContacts;

float fContactsPos[40];

float fContactsAngle[20];

float fContactsHigh[20];

float fContactsSpeed[20];

float fBullseye\_x, fBullsEye\_y;

float BullsEyeOdleglosc, BullsEyeKat;

float BullsEyeOdlegloscKur, BullsEyeKatKur;

float x, y, z, altitiude, heading, gs, pitch, roll;

float crsPos[2];

float target\_speed, target\_heading;

bool run;

bool target\_track;

short track\_num;

short MFD1\_menu\_z, MFD2\_menu\_z;

void TryConnection();

void PrzygotujBuforZwrotny();

};

Jak pokazano na listingu 4.10 większość pól i metod klasy ma dostęp publiczny. Dzięki temu możliwe jest odwołanie się do dowolnej zmiennej i metody. Ponieważ nawiązywanie połączenia odbywa się w drugim równoległym wątku aktualne dane są uzupełniane niezależnie od wątku renderującego.

Połączenie sieciowe odbywa się na zasadzie gniazd (Windows Sockets). Funkcja TryConnection wykonuje się nieprzerwanie w pętli cały czas łącząc się z aplikacją serwera i pobierąc od niej bufor danych.

Zakończenie połączenia

Odebranie bufora

Wysłanie bufora

Operacje  
udane?

Nawiązanie połączenia

Utworzenie socketu

Ustalenie adresu i portu

Ustalenie adresu i portu

START

Rys. 4.11 Mechanizm komunikacji sieciowej drugiego wątku aplikacji symulatora MFD

Funkcją realizującą właściwego adresu jest funkcja getaddrinfo. Wypełnia ona strukturę związaną z adresem. Dysponując adresem oraz numerem portu (które są aktualizowane zawsze gdy użytkownik zmieni ich wartości poprzez wybór opcji z menu kontekstowego) utworzono socket za pomocą funkcji socket oraz połączono się z nim za pomocą funkcji connect. W przypadku wykrycia nieprawidłowego działania lub zwróconych przez funkcje wartości pętla wykonuje się od nowa. Najważniejszymi funkcjami są send oraz recv. Pozwalają one na wysłanie i odbiór danych w formacie tekstowym dzięki czemu wyeliminowano niskopoziomowe operacje na pakietach. Pętla nawiązywania połączenia pobierania i wysyłania danych powtarza się w nieskończoności, a działanie wątku jest przerywane jedynie gdy swoją pracę zakończy wątek główny aplikacji.

Dużą zaletę zastosowanego rozwiązania pozyskiwania danych stanowi fakt, że użytkownik może napisać samodzielnie swój program serwera i odpowiednio przesyłając dane na wskazany adres i port zintegrować wskaźnik MFD z dowolnym źródłem generowania danych. Poniżej przedstawiono schemat ramki danych przesyłanej we wskaźniku MFD.

Zmienne parametrów oddzielone są w ramce znakiem ‘$’. Kolejność defniniowania zmiennych jest następująca:

* 1 zmienna położenie X statku powietrznego w milach morskich
* 2 zmienna położenie Y statku powietrznego w milach morskich
* 3 zmienna położenie Z statku powietrznego w milach morskich
* 4 zmienna kąt pochylenia w stopniach
* 5 zmienna kąt przechylenia w stopniach
* 6 zmienna wysokość barometryczna w stopach
* 7 zmienna kurs w stopniach
* 8 zmienna prędkość względem ziemi w węzłach
* 9 zmienna liczba kontaktów na radarze
* 10 zmienna kurs obiektu śledzonego w stopniach
* 11 zmienna prędkość obiektu śledzonego w węzłach
* 12-32 tablica kątów azymutu kontaktów na radarze (+/- 60 stopni)
* 32-52 tablica odległości kontaktów na radarze w milach morskich
* 52-72 tablica kątów kursowych względem SP w stopniach
* 72-92 tablica wysokości kontaktów w stopach
* 92-112 tablica prędkości kontaków względem SP w węzłach

Zmienne wysyłane w buforze z Aplikacji także oddzielono od siebie znakami ‘$’. Zdefiniowane są w nastepującej kolejności:

* 1 zmienna tryb ekranu MFD1
* 2 zmienna tryb ekranu MFD2
* 3 zmienna pozycja X bullseye w milach morskich
* 4 zmienna pozycja Y bullseye w milach morskich
* 5 zmienna odległość w kursora radaru od SP w milach morskich
* 6 zmienna kąt kursora radaru względem SP w stopniach (+/- 60)
* 7 zmienna logiczna tryb śledzenia radaru
* 8 zmienna numer kontaktu śledzonego przez radar

Domyślnym adresem jest localhost, a portem port 4333. Użytkownik może zmienić obydwa parametry komunikacji sieciowej z poziomu okna dialogowego po wybraniu odpowiedniego przycisku z menu kontekstowego.

### 4.3.6 Klasy okien dialogowych

W aplikacji zastosowano kilka okien dialogowych służących do komunikacji z użytkownikiem oraz pobrania od niego danych. Każde okno dialogowe posiada swoją unikatową klasę, która zarządza operacjami dostosowanymi do potrzeb w jakich utworzono okno.

Klasy okien dialogowych działają według podobnego schematu. Pierwszą czynnością jest utworzenie obiektów w zmiennych statycznych w procedurze obsługi komunikatów klasy CoknoGL. Wówczas każda zmienna statyczna (istniejąca przez cały czas działania programu w unikatowych egzemplarzu) reprezentuje okno dialogowe. Kiedy użytkownik wybierze za pomocą menu kontekstowego konkretne okno w procedurze obsługi komunikatu poprzez zmienną statyczną tworzone jest okno dialogowe, a cala aplikacja zostaje wstrzymana.

Przykładową klasą okna dialogowe jest Coknodialog. Nagłówek tej klasy przedstawiono na listingu 4.11.

Listing 4.11

enum DIALOG\_STAN { DIALOG\_WAYPOINTS, DIALOG\_ZAGROZENIA, DIALOG\_STERPOINTY};

class Coknodialog

{

public:

Coknodialog();

virtual ~Coknodialog();

INT\_PTR OtworzOkno(int uParameter, HWND hParent);

static float xwaypoints[20];

static float ywaypoints[20];

static float x\_zagrozenia[20];

static float y\_zagrozenia[20];

static float r\_zagrozenia[20];

static float x\_sterpointy[20];

static float y\_sterpointy[20];

static DIALOG\_STAN sStan;

static int nIndex, nIndex2, nIndex3;

private:

INT\_PTR hUchwytOkna;

RECT rObszar;

static INT\_PTR CALLBACK DlgProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

};

Klasa ta służy do wyświetlania okna dialogowego oraz pobrania od użytkownika wartości punktów nawigacyjnych, predefiniowanych zagrożen oraz dodatkowych symboli nawigacyjnych. Klasa posiada unikatową procedurę obsługi komunikatów, która jest wywoływana dla wiadomości przeznaczonych dla okna dialogowego podobnie jak w przypadku głównego okna. W procedurze można obsługiwać komunikaty dotyczące kontrolek, a także wyświtetlać kolejne okna. Same okna dialogowe dodawane są do aplikacji w formie tzw. zasobów. Określony zasób cechuje indywidualny indentyfikator za pomocą którego można wykonywać różne operacje związane z tym zasobem. Okna dialogowe i inne zasoby omówiono w dalszej części tego podrozdziału.

Zestawienie wszystkich klas okien dialogowych przedstawiono w tabeli 4.3

Zestawienie klas okien dialogowych oraz ich przeznaczenia Tabela 4.3

|  |  |
| --- | --- |
| Nazwa klasy | Funkcja w aplikacji |
| Coknodialog | Zbiera od użytkownika dane nawigacyjne, punkty trasy predefiniowane zagrożenia |
| Coknodialog\_sms | Zbiera od użytkownika informacje na temat aktualnych podwieszeń na belkach |
| Cdialogbullseye | Zbiera od użytkownika informacje na temat współrzędnych położenia obiektu BullsEye |
| Cdialogipadress | Zbiera od użytkownika informacje na temat adresu IP oraz numeru portu |

### 4.3.7 Zasoby aplikacji

W programie zdefiniowano tzw. Zasoby aplikacji czyli elementy które dołączane są do wynikowego pliku .exe. Zasobami jakie zastosowano w aplikacji są:

* Ikona aplikacji
* Okna dialogowe
* Tablica łańcuchów znakowych
* Menu kontekstowe

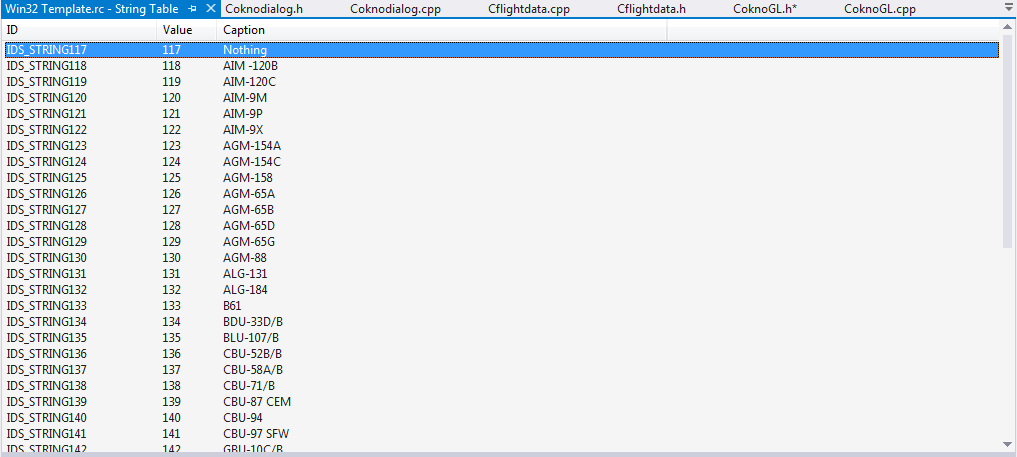
Ikona aplikacji dołączona została do zasobów w formie pliku .ico. Za pomocą funkcji WinAPI system operacyjny odczytuje informację o ikonie i ustanawia jej wygląd na takich elementach okna jak belka tytułowa, pasek narzędzi lub ikona na pulpicie. Ikonę aplikacji MFD simulator przedstawia rysunek 4.9.



Rys 4.12 Ikona aplikacji – zasób aplikacji MFD simulator

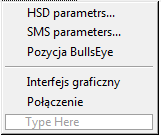
Inne zasoby jakie dodano do aplikacji to okna dialogowe. Odpowiadają za graficzny interfejs GUI pomiędzy aplikacją, a użytkownikiem. Ich działanie nadzorowane jest z poziomu klas okien dialogowych opisanych wcześniej.

Tablicę łańcuchów znakowych dodano do zasobów aplikacji w celu szybszego przetwarzania nazw uzbrojenia podwieszanego na belkach. Nazwy pocisków i środków rażenia zostały zdefiniowane w jednym miejscu jako zasób aplikacji, a odwoływanie się do nich poprzez indeksowanie znacznie przyspieszyło proces implementacji. Rysunek poniżej przedstawia wygląd zdefiniowanej tablicy łancuchów znakowych w programie Microsoft Visual Studio 2013.



Rys. 4.13 Tablica łańcuchów znakowych – zasób aplikacji MFD simulator

Ostatnim zasobem użytym w aplikacji jest zasób menu kontekstowego. W łatwy sposób możliwe jest zdefiniowanie wyglądu menu oraz nadanie poszczególnym jego elementom identyfikatorów. Wówczas z poziomu klasy CoknoGL można za pomocą tak zdefiniowanych identyfikatorów dokonywać selektywnego wyświetlania wybranych przez użytkownika okien dialogowych. Poniżej przedstawiono wygląd zasobu menu aplikacji.



Rys 4.14 Menu kontekstowe – zasób aplikacji MFD simulator

Dzięki zastosowaniu technologi identyfikatorów manipulowanie zasobami z poziomu kodu jest niezwykle proste i pomocne dla programisty. Ponadto możliwe jest szerokie definiowanie wyglądu części zasobów jak okna dialogowe czy menu. Okupione jest to dodatkowym miejscem zajmowanym przez plik wykonywalny .exe oraz wolniejszym wczytywaniem poszczególnych zasobów jednak stosując tego typu rozwiązanie w odniesieniu do prosty elementów jak ikony, menu czy okna dialogowe można zwiększyć efektywność pracy.

### 4.3.7 Programy cieniujące

W aplikacji MFD Simulator zastosowano potokowe przetwarzanie geometrii zgodnie z założeniami nowego profilu biblioteki OpenGL. W tym celu w odróżnieniu od poprzednich wersji biblioteki należało zastosować programy do cieniowania, tzw. Shadery. Przetwarzanie geometrii polega na działaniu dwóch programów Shaderów – Shadera wierzchołków oraz Shadera fragmentów. Działają one w sposób potokowy. Program Shadera wierzchołków wywoływany jest dla każdego wierzchołka w renderowanej serii danych z bufora na karcie graficznej. Poniższy listing przedstawia kod jednego z Shaderów wierzchołków jakie zastosowano w aplikacji.

Listing 4.12

#version 140

in vec3 vertexposition;

in vec2 textureposition;

uniform mat4 mvp;

out vec2 uv;

void main()

{

gl\_Position = mvp \* vec4(vertexposition.xy, 0, 1);

uv = textureposition;

}

Program ten nie jest skomplikowany. Za pomocą zmiennych vertexposition oraz textureposition dostarczane są współrzędne wierzchołka oraz tekstury przechowywane w buforze na karcie graficznej. Zmienna uniform przesyła macierz przekształcenia przez którą mnożone są wierzchołki w celu uzyskania przetransformowanych współrzędnych. Zmienna uv przepuszczana jest dalej do Shadera fragmentów w którym jest płynnie interpolowana dla każdego teksela.

Listing 4.13

#version 140

uniform sampler2DRect texturesampler2d;

in vec2 uv;

out vec4 FragmentColor;

void main()

{

vec4 wektor = texture(texturesampler2d, uv);

if(wektor == vec4(0,0,0,1))

{

FragmentColor = vec4(wektor.xyz, 0);

}

else

{

FragmentColor = wektor;

}

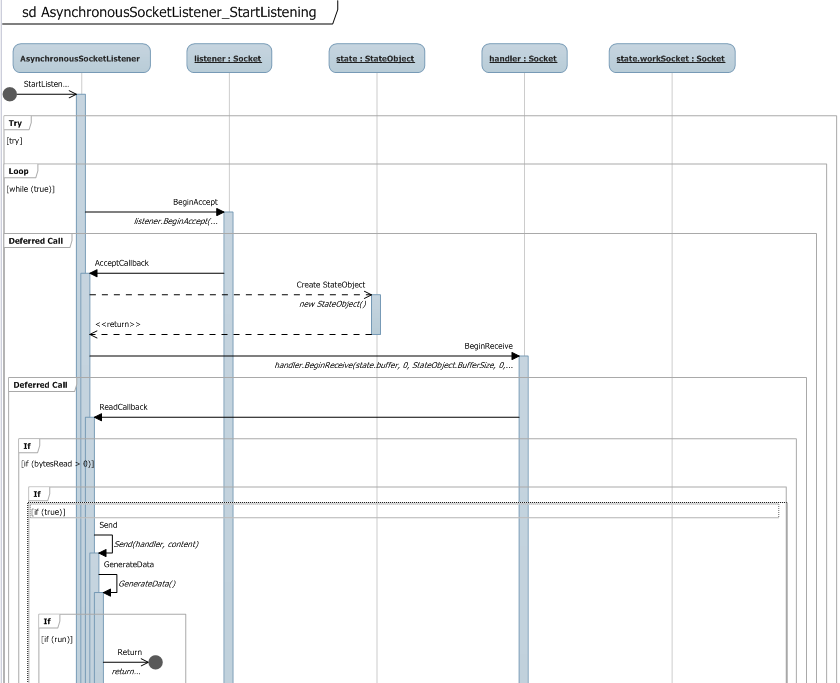
}

Przeznaczeniem Shadera fragmentów jest przyjęcie interpolowanej zmiennej współrzędnych tekstur, próbki tekstury oraz przeliczenie wartości każdego teksela. W wyniku końcym pracy programu Shadera fragmentu w zmiennej FragmentColor przekazano nową wartość koloru teksela. Program ten sprawdza w instrukcji warunkowej wartość koloru każdego próbkowanego teksela i jeśli jest on czarny ustawia zmienną alfa na wartość 0, przez co w równaniu mieszania teksel ten jest przezroczysty.

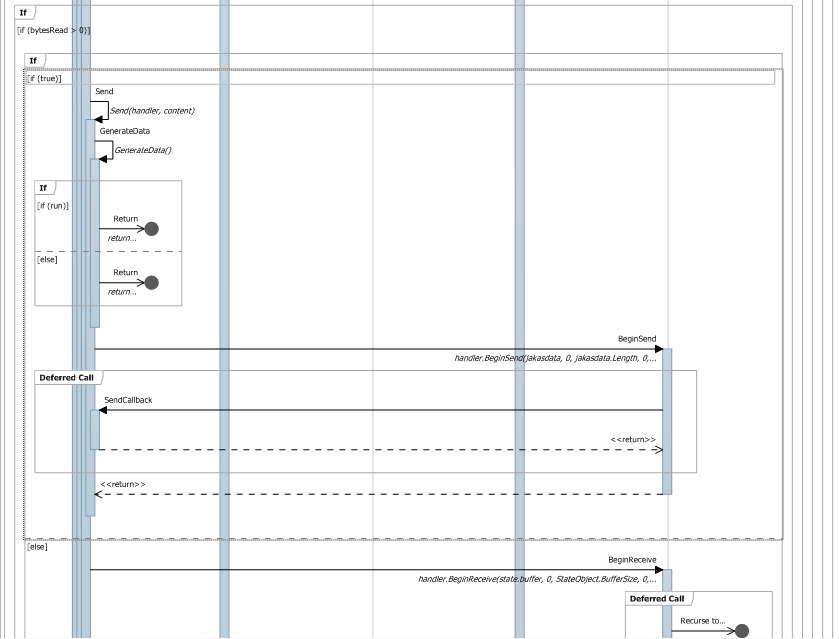
Obok wymienionych Shaderów zastosowano dodatkowe Shadery nieznacznie różniące się wersją programu GLSL w celu poprawienia niezawodności programu dla starszych wersji sterownika OpenGL.

## 4.4 Aplikacja serwerowa

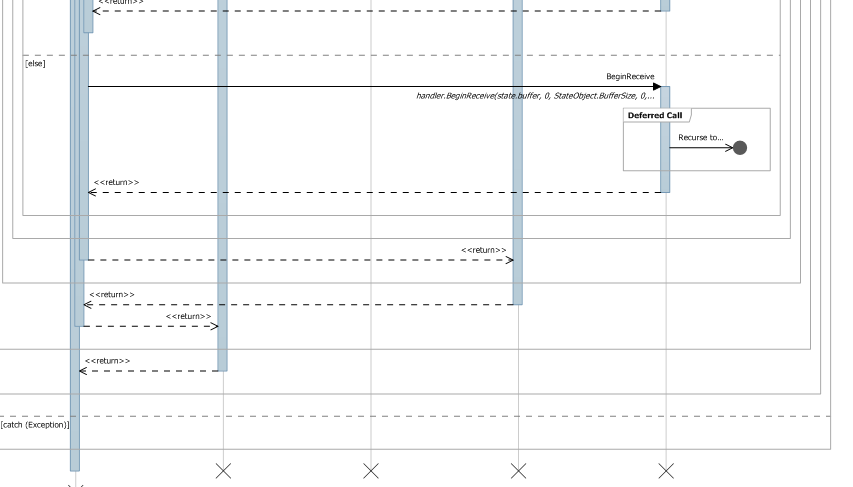
Aplikację serwera napisano w języku obiektowym C#. Jej głównym zadaniem jest pobieranie informacji o przychodzących połączeniach klienckich i po zaakceptowaniu wybranych połączeń przesyłaniu bufora danych. Dane wyciągane są z aplikacji symulatora Falcon BMS 4.32 dzięki bibliotece F4SharedMem.dll. Aplikację serwera napisano z myślą o prostocie jej działania. Nie udostępnia ona żadnego interfejsu graficznego GUI, jedynie wyświetla w podstawowym zakresie komunikaty w oknie konsoli. Poniżej przedstawiono schemat działania aplikacji serwera.



Rys 4.15 Diagram sekwencji Aplikacji serwera – cz. 1



Rys 4.16 Diagram sekwencji Aplikacji serwera – cz. 2



Rys 4. 17 Diagram sekwencji Aplikacji serwera – cz. 3

W programie zaimplementowano dwie klasy. Klasa StateObject odpowiada za przechowywanie informacji o aktualnym Sockecie oraz przechowywanie bufora do odbioru transmisji danych. Listing 4.14 przedstawia kod klasy StateObject.

Listing 4.14

public class StateObject

{

// Client socket.

public Socket workSocket = null;

// Size of receive buffer.

public const int BufferSize = 1024;

// Receive buffer.

public byte[] buffer = new byte[BufferSize];

// Received data string.

public StringBuilder sb = new StringBuilder();

}

Drugą z klas jest klasa AsynchronousSocketListener, która steruje logiką aplikacji. Jedną z jej metod jest funkcja StartListening, która przygotowuje Socket do transmisji oraz w zależności od tego czy występują wykryte połączenia na danym porcie zatwierdza połączenie i kieruje je do nowego wątku. Wielowątkowe segregowanie połączeń umożliwia pracę Socketu w trybie asynchronicznym. Metoda AcceptCallback akceptuje połączenie i przekazuje je dalej do metody ReadCallback, która odczytuje dane przychodzące ze strony klienta do bufora w klasie StateObject. Gdy dane zostaną odczytane metoda Send ustanawia wątek wysyłający dane z Aplikacji Falcon BMS i w tym celu wywołuje metodę SendCallback. Zanim to nastąpi wywoływana jest metoda GenerateData, która poprzez bibliotekę F4Sharedmem.dll wyciąga dane z aplikacji Falcon BMS 4.32 oraz formuje je w bufor ramki zgodny ze specyfikacją . Funkcja zwraca bufor, który jest przekazywany do funkcji Send oraz SendCallback, skąd zostaje wysłany protokołem TCP/IP pod adres klienta. Aplikacja jest bardzo prosta i nie korzysta z danych przychodzących ze strony klienta. Poniższy listing przedstawia kod metody GenerateData formułującej bufor danych i zwracającej ten bufor do funkcji wysyłających dane.

Listing 4.15

public static byte[] GenerateData()

{

F4SharedMem.FalconDataFormats dataFormat = FalconDataFormats.BMS4;

F4SharedMem.Reader reader = new F4SharedMem.Reader(dataFormat);

run = reader.IsFalconRunning;

if (run)

{

F4SharedMem.FlightData flightdata = reader.GetCurrentData();

x = flightdata.x;

y = flightdata.y;

z = flightdata.z;

Console.Write(x.ToString() + " // " + y.ToString() + "\n");

altitiude = flightdata.aauz;

heading = flightdata.currentHeading;

gs = flightdata.kias;

pitch = flightdata.pitch;

roll = flightdata.roll;

string jakas = "";

jakas += "1";

jakas += "$";

jakas += x;

jakas += "$";

jakas += y;

jakas += "$";

jakas += z;

jakas += "$";

jakas += pitch;

jakas += "$";

jakas += roll;

jakas += "$";

jakas += altitiude;

jakas += "$";

jakas += heading;

jakas += "$";

jakas += gs;

jakas += "$";

return Encoding.ASCII.GetBytes(jakas.ToCharArray());

}

else

{

return Encoding.ASCII.GetBytes("0");

}

}

Aplikacja serwera może być również napisana przez użytkownika i dopasowana do wymagań innego środowiska niż Falcon BMS. Wówczas użytkownik musi jedynie zadbać o poprawną kolejność definiowania poszczególnych zmiennych parametrów lotu, zgodnie ze specyfikacją przytoczoną w tej pracy w poprzedniej części rozdziału.

* 1. WNIOSKI WYNIKAJĄCE Z REALIZACJI ZADANIA

Po zrealizowaniu założeń projektowych i celów niniejszej pracy wyciągnięto liczne wnioski odnoszące się do samego programu (aplikacji) jak również do całości systemu. Proces projektowania aplikacji jest niewątpliwe bardzo złożony, składa się na niego wiele czynników. Uwzględnienie wszystkich zależności często na wczesnych etapach prac jest niemożliwe i dlatego projekt zmieniał się w czasie. Dużą efektywność pracy osiągnięto dzięki wstępnym nakreśleniom problematyki, zrozumieniu funkcjonowania struktury wskaźnika MFD oraz wydajnym narzędziom wspomagającym proces programowania. Obecne możliwości oprogramowania typu CASE pozwalają znacznie podnieść standardy i jakość aplikacji, a także skrócić czas pracy programisty. Największymi problemami z jakimi zetknięto się w czasie pisania programu MFD Simulator była różnorodność sprzętu u docelowego odbiorcy, a zarazem konieczność wprowadzenia powtarząjących się elementów aplikacji innych dla różnych wersji sterownika OpenGL. Stosowanie API udostępnianych przez producenta konkretnego sprzętu lub systemu ogranicza zakres stosowalności aplikacji w obrębie tego systemu. Ważnym wnioskiem wynikającym z tego faktu jest konieczność stosowania zewnętrznych bibliotek i gotowych platform uruchomieniowych. Stosowanie tego typu narzędzi jest oszczędne – pozwala na skupieniu się na problemie bez konieczności rozpisywania od początku całej procedury obsługi okien w danym systemie i operacji zarządzania nimi. Ważnym czynnikiem i miarą niezawodności aplikacji jest uwzględnienie odpowiednich testów podczas wytwarzania oprogramowania. Testowanie oprogramowania jest złożonym procesem i na potrzeby niniejszej pracy uwzględniono etap testowania jedynie w podstawowym zakresie. Przyjęto model warstwowy oddzielając interfejs użytkownika GUI, logikę aplikacji oraz warstwę danych. Każdą z poszczegolnych warstw potraktowano jako osobny komponent, który w wyniku podania do niego odpowiednich sygnałów wejścia, musi wypracować oczekiwane (zestawione w wymaganiach) sygnały wyjść. Gdy wypracowane sygnały wyjść odpowiadały założeniom uznano działanie komponentu za zadowalające i zdatne do integracji z aplikacją.

Realizacja wskaźnika MFD jest zadowalająca. Jego funkcjonowanie i budowa jest zbliżona do rzeczywistego przyrządu. Zrezygnowano z zaimplementowania niektórych funkcji jak tworzenie map radarowych czy obrazy z kamer. Wskaźnik stanowi przykład działania najistotniejszych funkcji ekranu MFD. Ważnym wnioskiem jaki wyciągnięto w trakcie realizacji zadania była możliwość stworzenia dwóch wskaźników złożonych z monitorów oraz dopasowanych do nich ramek. Dzięki temu stopień symulacji działania wskaźnika jest nieporównywalnie większy niż w przypadku używania jednego monitora i alternatywnej wirtualnej ramki.

Należy wyraźnie podkreślić, że kierunki rozwoju w dziedzinie lotniczych technik zobrazowania informacji zmierzają do stosowania cyfrowych ekranów na których dostępna jest zintegrowana informacja. Nie sposób zliczyć zalet takiego podejścia do technologii zobrazowania informacji, a przykłady najnowszych maszyn wojskowych jakie przedstawiono w pierwszej części tej pracy potwierdzają tą regułę. Coraz bardziej złożone oprogramowanie przejmuje coraz więcej zadań i odciąża w pracy pilota. Nowe języki obiektowe, oraz obiektowe podejście do inżynierii oprogramowania pozwala pisać wydajne aplikacje w bardzo dużym stopniu odwzierciedlające złożoność świata. Zastosowanie „glass-cockpitów” w większości samolotów wydaje się być kwestią czasu.

* 1. Bibliografia

AVIONICS AND NONNUCLEAR WEAPONS DELIVERY FLIGHT. (2013).

Axelson, J. (2001). *USB Complete.* Madison: Lakeview Research.

Dougherty, M. J. (2011). *Nowoczesne uzbrojenie lotnicze.* Bremen: MAK.

*F-35 mission systems*. (2008). Pobrano 2015 z lokalizacji F-35 Lighting II: https://www.f35.com/about/capabilities/missionsystems

Wright, R. S., Haemel, N., Sellers, G., & Lipchak, B. (2011). *OpenGL księga eksperta.* Gliwice: HELION.