

# Grafika komputerowa - Anaglify

Projekt nr. 3

*Autorzy:*

Jakub Haberko

Marcin Pojasek

Tomasz Prus

# 1 Tytuł projektu i autorzy projektu

Celem projektu pod tytułem „*Anaglify*” jest stworzenie oprogramowania, które będzie wyświetlać dany obiekt w dwóch kolorach i wzajemnie odpowiednio przesuniętych położeniach. Poprzez zastosowanie odpowiednich filtrów można uzyskać efekt przestrzennego widzenia.

Projekt będzie realizowanych przez 3 osobowy zespół w składzie:

- Marcin Pojasek - odpowiedzialny za komunikację z użytkownikiem (layout), plik pomocy, stronę WWW projektu.
- Jakub Haberk - algorytm generowania obrazów, przygotowanie danych wejściowych, testy
- Tomasz Prus - odczyt danych wejściowych, zapis do pliku, przygotowanie diagramu klas, nadzór nad dokumentacją.

## 2 Opis projektu

Widzenie przestrzenne jest wynikiem interpretacji jakiej dokonuje mózg porównując dwa, różniące się nieco obrazy pochodzące z każdego oka niezależnie. Tworzenie obrazów przestrzennych polega na przygotowaniu dwóch obrazów, oddzielnie dla lewego i prawego oka, a następnie wyświetlenie ich w taki sposób, aby odpowiedni obraz trafił do właściwego oka. Istnieje wiele metod realizacji tego zadania. Jedną z bardziej znanych polega na wyświetleniu każdego z obrazów w różnych kolorach na tym samym monitorze, a następnie oglądaniu ich przez specjalne, dwukolorowe okulary. Obrazy takie nazywamy anaglifami. Naszym celem będzie przygotowanie programu, który na podstawie zadanych danych wyświetli na ekranie trójwymiarowy obraz i pozwoli nim obracać wokół trzech prostopadłych osi. Dodatkowo poszerzymy jego funkcjonalność o możliwość ruchu dowolnych obiektów wg określonego toru.

## 3 Założenia przyjęte w realizacji projektu

Zakładamy, że program ma cechować się funkcjami:

- obsługa obiektów: krawędzie, kule, prostokąty oraz graniastosłup przezroczysty o podstawie prostokąta. Z pliku tekstowego, o strukturze opisanej dalej, będą wczytywane położenia i parametry obiektów. Użycie języka obiektowego umożliwi późniejszą łatwą rozbudowę o dodatkowe obiekty.
- zmiana grubości linii. W trakcie działania programu będzie można zmienić grubość wyświetlanych linii obiektów.
- obrót obiektu. Po wczytaniu obiektów możliwe będzie obracanie układu względem obserwatora.
- przybliżenie/oddalenie obserwatora. Podobnie jak obracanie, układ będzie można przybliżyć bądź oddalić.

- odczyt z pliku. Odpowiednia funkcja będzie zajmować się wczytywaniem obiektów z pliku tekstowego. Proponujemy porzucić ograniczenie do 500 obiektów i ograniczać je jedynie dostępnym miejscem w pamięci (całość będzie zapisywana w dynamicznej tablicy obiektów).
- zapis aktualnego widoku do pliku. Aktualny widok będzie można zapisać w dowolnej skali (rozdzielczości) do pliku graficznego.
- wydruk widoku. Użytkownik będzie miał możliwość wydrukowania anaglifa na drukarce.
- możliwość animowanego ruchu obiektu (możliwość uzyskania zegara z obracającą się wskazówką). W pliku z danymi wejściowymi każdy z parametrów (po za parametrem określającym rodzaj obiektu) będzie mógł być prostą funkcją czasu  $t$ . W programie wybranie stosownej opcji będzie powodowało wyświetlanie anaglifa dla kolejnych wartości  $t$ . Ostatecznie możliwe będzie odtwarzanie prostych animacji (obracająca się krawędzie, przesuwanie obiektów), niezależnie od obrotu czy zbliżania układu. Parser przetwarzający wyrażenie „rozumie” następujące działania  $+$ ,  $-$ ,  $*$ ,  $/$  oraz funkcje  $\sin$ ,  $\cos$ ,  $\tan$  oraz  $\cot$ . Operacje będzie można grupować przy użyciu zwykłych nawiasów.
- funkcja do dostrajania okularów. Proste okienko dialogowe z możliwością dostrojenia programu do posiadanych okularów 3D.
- wersje językowe. W programie będą zapisane komunikaty w języku polskim. Specjalna funkcja będzie zamieniać komunikaty na odczytane z pliku językowego (jego konstrukcja będzie opisana w dalszej części opracowania). Wybór wersji językowej będzie dostępny z okna dialogowego „Ustawienia”.
- strona WWW. Aby pozostawić ślad po naszej pracy. Uruchomimy stronę WWW projektu. Będzie można z niej pobrać najnowszą wersję programu, pliki językowe, źródła i dokumentacje. O miejsce na serwerze fatcat, będziemy starać się w Zakładzie Informatyki Stosowanej WFTiJ AGH.
- przykłady. Na zakończenie prac wykonamy kilka przykładów ukazujących możliwości programu.

## 4 Analiza projektu

### 4.1 Specyfikacja danych wejściowych

W treści zadania zostały narzucone parametry na format danych wejściowych. I tak, jest to plik tekstowy który w pierwszej linii zawiera liczbę elementów z których zbudowany jest anaglif. Kolejne linie to już określenie położenia i rozmiarów obiektów.

Pozwoliliśmy sobie na drobną modyfikację dotyczącą komentarzy w pliku wejściowym, jeśli linia zaczyna się od znaku „#” lub „//” jest ignorowana. Natomiast pierwsza linia z komentarzem jest traktowana jako tytuł anaglifa i wyświetlana w okienku na którym ów anaglif jest rysowany.

Linie określające parametry poszczególnych obiektów mają następujący format: pierwsza liczba określa rodzaj obiektu, a kolejne liczby oddzielone spacjami definiują parametry dotyczące położenia i rozmiarów obiektu. Jeśli w linii po określeniu wszystkich parametrów

obiektu występują nadmiarowe dane to są one ignorowane. Obecnie w program obsługuje cztery rodzaje obiektów. Określenie liczbowe rodzaju obiektu różne od 1,2,3,4 i 5 jest ignorowane i obiekt nie jest inicjowany. W tabeli zestawiono obiekty i parametry je określające.

Linia w pliku wejściowym	Interpretacja przez program
1 $x_1$ $y_2$ $z_1$ $x_2$ $y_2$ $z_2$	Krawędź rozpięta na dwóch punktach $A(x_1, y_1, z_1)$ i $B(x_2, y_2, z_2)$
1 0 0 0 10 1.5 20	Krawędź wychodząca z początku układu współrzędnych do punktu o współrzędnych (10,1.5,20)
2 $x$ $y$ $z$ $d$	Kula o środku w punkcie $(x, y, z)$ i średnicy $d$ . Tworzona jest siatka kuli o regulowanej gęstości
2 0 0 0 10.5	Kula o średnicy w początku układu współrzędnych i średnicy 10.5
3 $x_1$ $y_1$ $z_1$ $x_2$ $y_2$ $z_2$ $x_3$ $y_3$ $z_3$	Prostokąt opisany na punktach $A(x_1, y_1, z_1)$ , $B(x_2, y_2, z_2)$ oraz $C(x_3, y_3, z_3)$
3 0 0 0 5 0 0 5 5 0	Kwadrat leżący w płaszczyźnie XY, początek w środku układu współrzędnych i boku równym 5.
4 $x_1$ $y_1$ $z_1$ $x_2$ $y_2$ $z_2$ $x_3$ $y_3$ $z_3$ $x_4$ $y_4$ $z_4$	Graniastosłup o podstawie prostokąta (punkt ABC) druga podstawa prostokątna przesunięta o wektor AC, gdzie $C(x_4, y_4, z_4)$ . Bryła jest przezroczysta.
4 0 0 0 5 0 0 5 5 0 0 0 5	Sześcian o jednym z wierzchołków w początku układu współrzędnych, długość krawędzi równa 5.

Widać że liczba rozumianych obiektów nie jest zbyt imponująca, ale z racji użytego obiektowego języka programowania łatwa do dalszej rozbudowy.

Wartości liczbowe określające obiekty są w jednostkach umownych, a program umożliwia ich skalowanie. Podawane wartości mogą być typu rzeczywistego (kropka jest separatorem dziesiętnym).

Szczególnie interesującą rzeczą jest możliwość określenia danego parametru funkcją zależną

od czasu (numeru klatki animacji „ $t$ ”). Stosowny parser czyta wyrażenie i wartość jest przeliczana dla kolejnych wartości „ $t$ ”.

Przykładowo:

$2\ 0\ 0\ 0\ 10 * \sin(t) * \sin(t)$  stworzy animację kuli która będzie naprzemiennie kurczyć się do zera i rosnać do średnicy równej 1.

Parser rozumie funkcje  $\sin$ ,  $\cos$ ,  $tg$  i  $ctg$ . Oczywiście zdefiniowane są operatory dodawania, odejmowania, mnożenia i dzielenia. Działania można grupować nawiasami.

Poza plikiem określającym dany anaglif, program po uruchomieniu odczytuje z rejestru ustawienia, tj. domysłą grubość linii, rodzaj posiadanych okularów, gęstość siatki kuli, kolor tła itp.

## 4.2 Opis oczekiwanych danych wyjściowych

W projekcie wyjściem (wynikiem działania programu) jest gotowy anaglif oglądany pod danym kątem i z ustalonej przez użytkownika odległości. Zatem wyjście pojawia się na ekranie. Dodatkowo może zostać zapisane do pliku graficznego o dowolnej rozdzielczości (operujemy grafiką wektorową, więc skalowanie do danej rozdzielczości jest bardzo proste i bezstratne). Dodatkowo przewidujemy możliwość wydruku anaglifów na drukarce.

Anaglif jest wyrysowany na obiekcie klasy TCanvas dostępnej standardowo w Borland C++. Taki obiekt, można wyświetlić, wydrukować czy zapisać do pliku.

## 4.3 Zdefiniowanie struktur danych

W programie konieczne jest przechowywanie 3 typów danych. Pierwszy typ – dane wczytane z zewnętrznego pliku zawierające obiekty składowe anaglifów. Program wczytując dane dotyczące kolejnych obiektów tworzy je i umieszcza wskaźniki dostępowe w tablicy obiektów klasy abstrakcyjnej (Obiekty), z której dziedziczą podklasy. Podklasy umożliwiają przechowywanie parametrów obiektów: ich rozmiarów, położeń itd. Natomiast z punktu widzenia anaglifów ważną jest wirtualna metoda Narysuj, która dostając wskaźnik do obiektu klasy TCanvas rysuje na nim swój anaglif. Szczegóły tego rozwiązania są omówione w dalszej części opracowania.

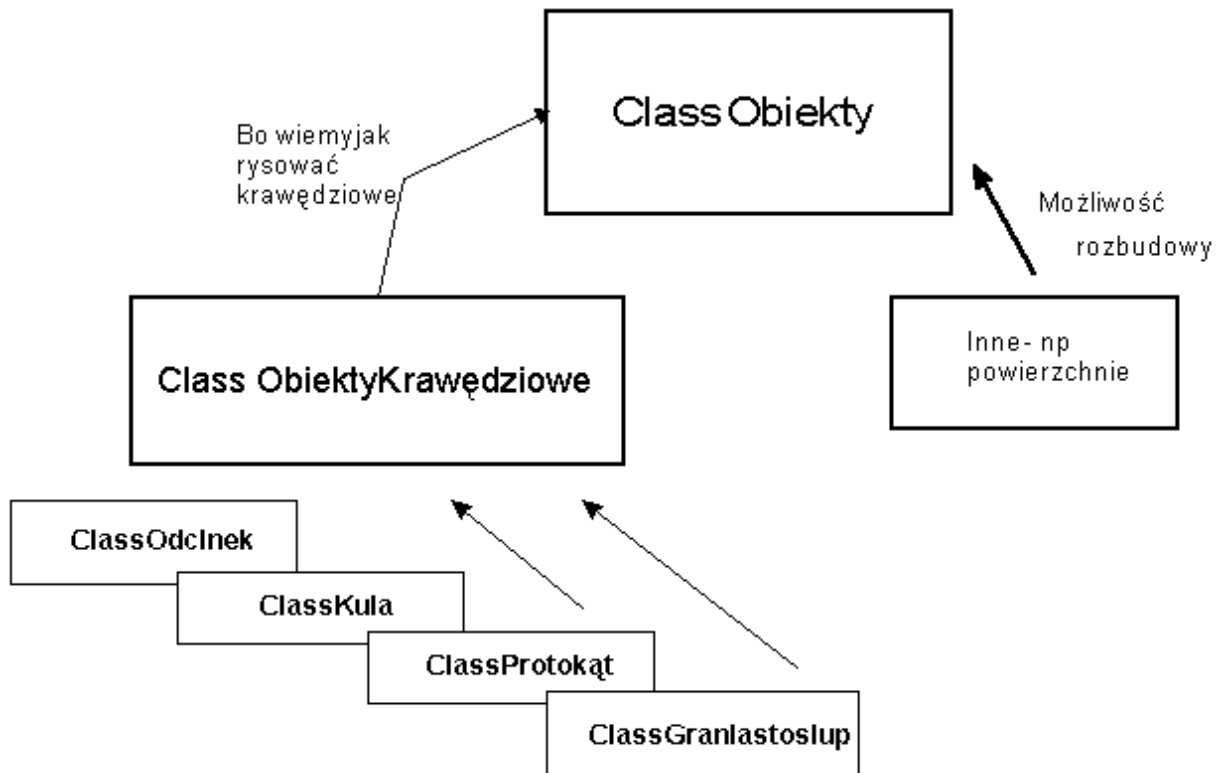
Ponieważ wszystkie obsługiwane przez nas obiekty są typu siatkowego - mają wierzchołki i połączenia między nimi. Wygodnie było utworzyć wspólną klasę ObiektyKrawędziowe, która zawiera wsłona dla wszystkich metode rysowania.

Pozostały do omówienia jeszcze dwa typy danych, tj. dane dotyczące ustawień programu. Są one zmieniane przez dwa okna dialogowe interfejsu i przechowywane w rejestrze systemu Microsoft Windows. W trakcie działania programu są one umieszczone w klasie TMainForm. Ostatni typ danych to dane dotyczące generowania i oglądania anaglifów, tj. kąt obrotu układu, odległość od obserwatora, skalowanie, przesunięcie, czas (dla animacji). Dane te przechowywane są w klasie statycznej Anaglif i są zmieniane poprzez interfejs użytkownika. Co więcej, każda zmiana implikuje odświeżenie anaglifów.

## 4.4 Specyfikacja interfejsu użytkownika

Interfejs zbudowany jest jako aplikacja wielookienkowa (MDI), tzn. jest formą główną z której wykonuje się podstawowe polecenia. Każdy nowy otwarty plik wczytywany jest do okienka (tzw. dziecka). Poniżej przedstawiamy bardziej szczegółowo interfejs.

- Menu Główne



#### + Plik

- \* Otwórz plik - Umożliwia otwarcie pliku zawierającego dane obiektu, otwierany plik musi mieć rozszerzenie \*.agf
- \* Eksport do BMP - Istnieje możliwość zapisu anaglifów jako obrazka. Przed zapisem wymagane jest podanie wysokości i długości zapisywanego anaglifów do pliku \*.bmp
- \* Drukuj, wydruk anaglifów

#### + Pomoc

- \* o programie
- \* Podrecznik użytkownika - Pozwala poznać dostępne opcje programu
- \* Strona WWW - link do oficjalnej strony projektu, dostępne najnowsze wersje programu, kontakt z autorami, możliwość ściągnięcia nowych plików \*.agf

#### - Okno Opcji anaglifów

- \* Obroty wg osi OX, OY albo OZ - Obrót obiektu/anaglifów wokół wybranej osi. Po prawej stronie scrolla istnieje przycisk (play) za pomocą którego można w sposób automatyczny spowodować obrót obiektu.
- \* Offset x,y - Względne przesunięcie obiektu/anaglifów wzdłuż osi OX lub OY
- \* Odległość - Operowanie scroll'em spowoduje zbliżanie/oddalanie obiektu/anaglifów (dokładnie układu w którym się znajduje).
- \* Klatka - Interuje parametr „t” od 0 do 1000, umożliwia to animowanie obiektów, ponieważ każda współrzędna można zapisać w postaci wyrażenia zależnego od t.

- \* Skala - skaluje współrzędne pierwotne anaglifu
- \* Prędkość animacji - dobór prędkości animacji
- Okno ustawień programu
  - \* Dobór kolorów okularów
  - \* Język
  - \* Tło
  - \* Grubość Linii
  - \* Rozstaw oczu [cm]
  - \* Odl. monitor-oczy [cm]
  - \* Gęstość siatki kuli
- Okno anaglifu - prawym przyciskiem myszy wywołujemy menu kontekstowe z możliwością wydruku, zapisu czy skopiowania do schowka. Dolny „wyskakujący” pasek informuje o tytule i złożoności obiektu.
- Forma główna obsługuje technike Drag-And-Drop;

## 4.5 Wyodrębnienie i zdefiniowanie zadań

Cały projekt można podzielić na moduły

- komunikacji z użytkownikiem
- odczytu danych z pliku wejściowego
- stworzenie klas obiektów
- algorytm generowania anaglifu na podstawie tablicy obiektów i parametrów dotyczących odległości i obrotu układu.
- prezentacja anaglifu na ekranie, zapis do pliku graficznego, wydruk

## 4.6 Wybór narzędzi programistycznych

Projekt jest zrealizowany w Borland C++. Wybór został podyktowany dwoma aspektami. Po pierwsze obiektowe C znacznie uprości zarządzanie obiektami wczytywanymi z pliku i późniejszą ich obsługę. Drugim aspektem jest środowisko IDE Borlanda, które jest potężnym i wygodnym narzędziem do tworzenia interfejsu użytkownika i zarządzania całym projektem. Wykorzystywać będziemy wersje Personal do użytku nie komercyjnego.

## 5 Podział pracy i analiza czasowa

Całość prac nad projektem podzieliliśmy na cztery etapy. Pierwszy z nich to głównie rozpisanie pomysłów na kartce papieru. Drugi i trzeci to już część z kodowaniem. Etap ostatni to testy, tworzenie systemu pomocy, strony WWW i innych.

Etap	Zadanie	Kto wykonuje	Czas trwania
1	Przygotowanie diagramu klas	Tomasz Prus	2 tygodnie
	Opracowanie layout-u programu	Marcin Pojasek	2 tygodnie
	Opracowanie algorytmów operujących na obiektach (konstrukcja anaglifu, obroty, przybliżanie/oddalanie obserwatora).	Jakub Haberko	2 tygodnie
2	Kodowanie - odczyt z pliku, zapis obrazka	Tomasz Prus	1 tydzień
	Konstrukcja metody wyświetlającej obiekt w formie anaglifu o określonym obrocie i odległości widza	Jakub Haberko	1 tydzień
3	Wywoływanie odpowiednich metod w interakcji z użytkownikiem	Marcin Pojasek	1 tydzień
4	Testy programu, przygotowanie przykładowych danych wejściowych	Jakub Haberko	1 tydzień
	Konstrukcja strony WWW projektu, plik pomocy do programy	Marcin Pojasek	1 tydzień
	Dopracowanie dokumentacji programu	Tomasz Prus	1 tydzień

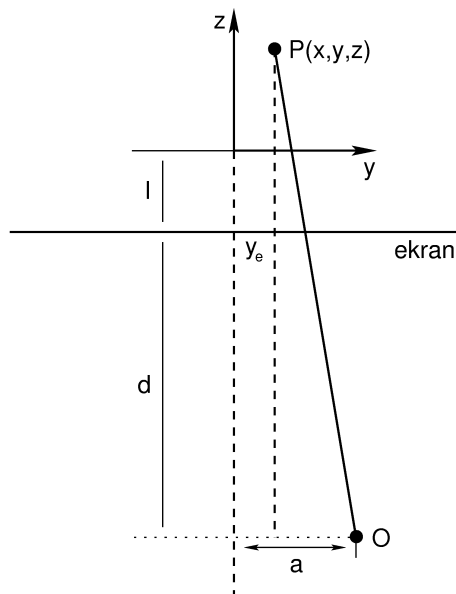
## 6 Opracowanie i opis niezbędnych algorytmów

### 6.1 Obliczanie ekranowych współrzędnych anaglifów

Trójwymiarowe bryły reprezentowane są na ekranie przy pomocy rzutu perspektywnego. W celu uzyskania obrazu dowolnego punktu  $P$  należy wykreślić prostą przechodzącą przez ten punkt i oko osoby siedzącej przed monitorem. Punkt przecięcia tej prostej z płaszczyzną ekranu to szukany rzut punktu  $P$ . Procedurę tę należy powtórzyć dla drugiego oka i uzyskany punkt narysować na ekranie innym kolorem, tak, aby po założeniu dwukolorowych okularów uzyskać złudzenie trójwymiarowości.

Współrzędne  $y$ -owe punktów na ekranie obliczamy w sposób następujący:



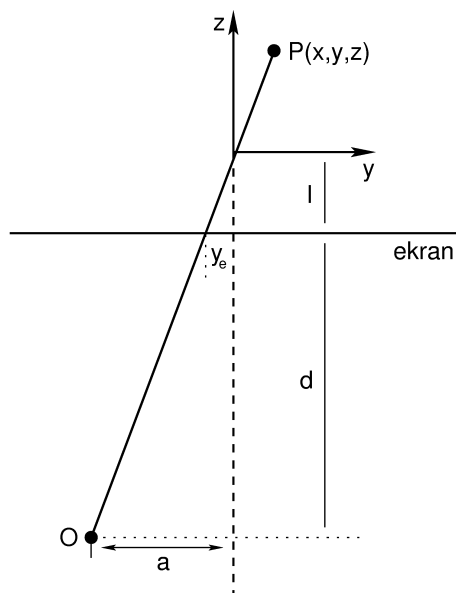


Dla *prawego oka* z podobieństwa odpowiednich trójkątów mamy:

$$\frac{y_e - y}{l + z} = \frac{a - y}{l + d + z} \quad (1)$$

co, po odpowiednich przekształceniach, daje:

$$y_e = y \frac{d}{l + d + z} + a \frac{l + z}{l + d + z} \quad (2)$$



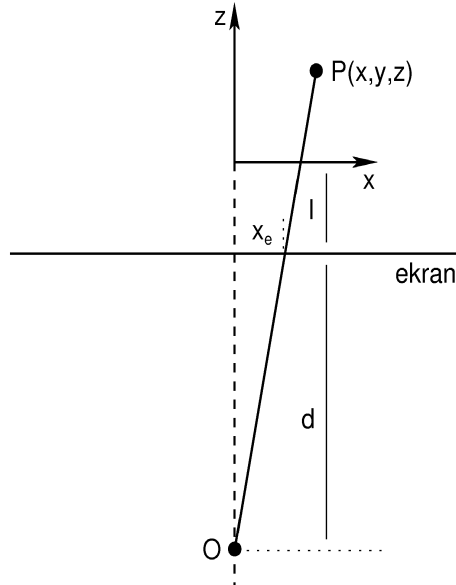
Dla *lewego oka* obliczenia są podobne:

$$\frac{a + y}{d + l + z} = \frac{y - y_e}{l + z} \quad (3)$$

skąd otrzymujemy:

$$y_e = y \frac{d}{l + d + z} - a \frac{l + z}{l + d + z} \quad (4)$$

Obliczenia dla współrzędnych  $x$ -owych są nieco prostsze:



$$x_y = x \frac{d}{d + l + z} \quad (5)$$

## 6.2 Obroty

Aby obrócić obiekt w przestrzeni trójwymiarowej wystarczy przemnożyć kolumnę jego starych współrzędnych przez odpowiednią macierz obrotu ( $X'_{nowe} = MX_{stare}$ ). Nie będziemy przytaczać tej macierzy ponieważ można ją znaleźć w każdym podręczniku geometrii. Ważne jest że punkt 3D obracamy wg ustawionych kątów i następnie z takiej bryły tworzymy anaglif.

## 6.3 Parser

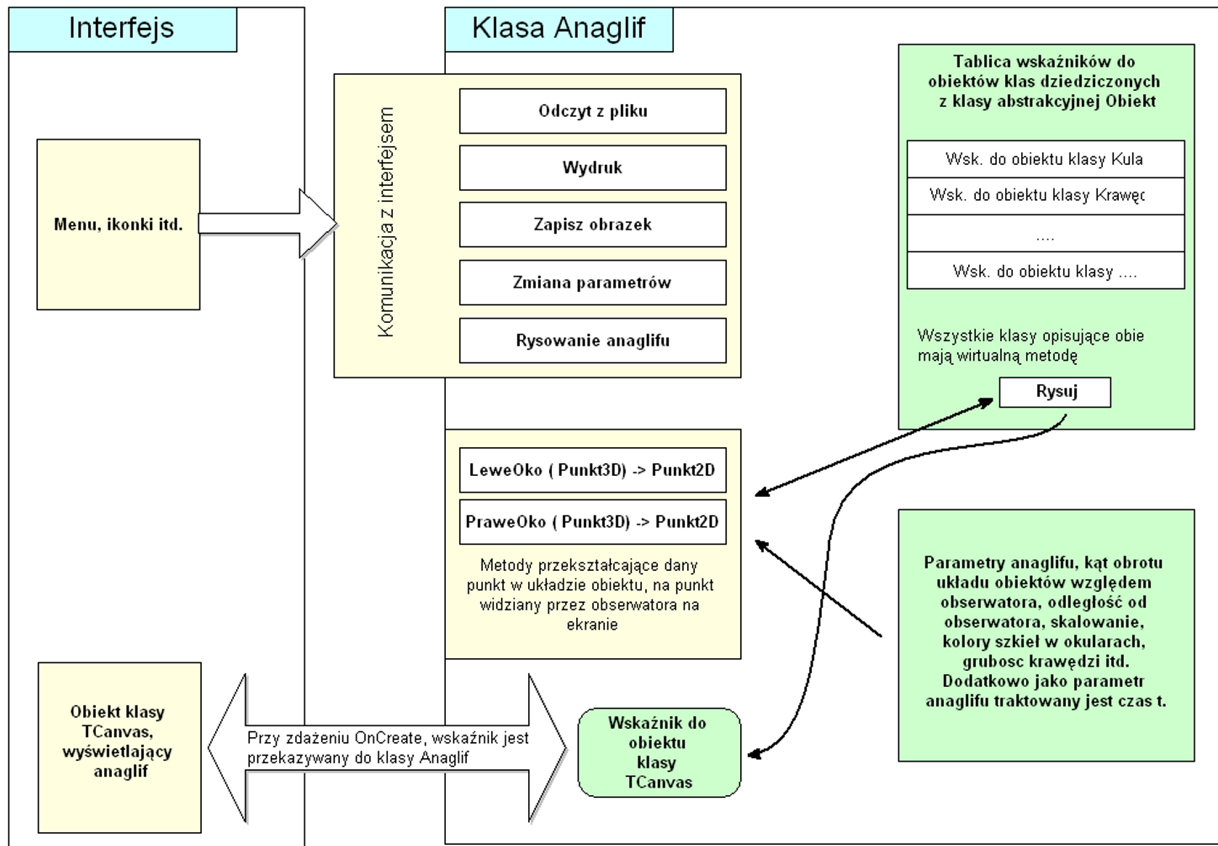
Algorytm obliczania wartości liczbowej ze zmienną  $t$  nie będzie tu opisany, ponieważ nie jest to główny cel projektu. Jednak dla kompletności rozważań dodam że najpierw wyrażenie jest dzielone wg tokenów i dokonywane jest przekształcenie do Odwrotnej Notacji Polskiej. W dalszej kolejności wyliczana jest wartość liczbową.

# 7 Kodowanie

## 7.1 Schemat blokowy

Na schemacie poniżej zaznaczono główne elementy programu. Widać że właściwy program został oddzielony od interfejsu poprzez stworzenie klasy zawierającej tylko metody

i zmienne statyczne o nazwie Anaglif. Odpowiednie zdarzenia generowane przez użytkownika powodują wywoływanie właściwych metod tej klasy. Przy czym wynik (anaglif) wraca do interfejsu i jest prezentowany na ekranie.



## 7.2 Opis klas, funkcji i zmiennych

Poniżej prezentujemy zbiory nagłówkowe klas wykorzystywanych w programie. Pominąmy klasy typu okien uzyskane z poziomu Borland-a.

### Klasa pomocnicza — Punkt2D

```
class Punkt2D {
public:
    double x,y;
};
```

### Klasa pomocnicza — Punkt3D

```
public class Punkt3D : public Punkt2D {
public:
    double z;
};
```

Widać, że klasa Punkt3D dziedziczy z klasy Punkt2D dodając jedynie swój parametr – składową z. Obiekty tych klas są wykorzystywane do wyświetlania anaglifów oraz zapisu punktów charakterystycznych w pamięci operacyjnej komputera. Obiekt tych klas reprezentuje punkt w przestrzeni 2D lub 3D.

## Klasa pomocnicza — Parser

```
class Parser {
    private:
        char *wyrazenie; // wyrażenie
        char **wyrONP; // tablica wyrażenia ONP
        int DlugoscONP; // dlugosc tablicy z ONP
        int TypToken(const char); // typ tokena
        int PriorytetToken(const char*); // priorytet tokena
    public:
        Parser (void);
        Parser (char *);
        ~Parser();
        void ONP();
        Parser & operator=(const char*);
        double ObliczDlaT (int); // przelicza dla t
        bool CzyToWyrazenie(); // sprawdza czy jest
        // to liczba czy wyr.
};

// klasa pomocnicza - obsługa stosu
class StosInt {
    private:
        int Stos[MAX_TOKEN];
        int IleNaStosie;
    public:
        StosInt(void);
        int DodajNaStos(int);
        int ZdejmijZeStosu();
        int Wielkosc();
        bool CzyStosPusty();
};
```

Klasa Parser jak sama nazwa wskazuje zajmuje się obsługą wyrażeń tekstowych obliczając ich wartość liczbową. Zawiera konstruktor domniemany oznaczający wartość 0 oraz konstruktor do przekazania tekstu będącego wyrażeniem do dalszych obliczeń. Wyrażenie to powinno zawierać zmienną  $t$ . Klasa Treść wyrażenia przechowywana jest w składniku prywatnym klasy, rozmiar nie ograniczony. Zaraz po stworzeniu obiektu uruchamiana jest metoda ONP, która dokonuje rozkładu wyrażenia do beznawiasowej odwrotnej notacji polskiej. Tablica zawierająca kolejne składniki ONP jest pamiętana przez cały czas życia obiektu

Szczególnie istotną jest metoda ObliczDlaT, która dla podanej wartości  $t$  bazując na ONP oblicza i zwraca wartość wyrażenia.

**Klasy definiujące obiekty (w sensie graficznym) obsługiwane przez program.**

```

// klasa abstrakcyjna, przodek wszystkich obiektow mozliwych
// do wyrysowania w postaci anaglif
class Obiekty {
protected:
bool CzyJestAnimacja; // informuje o tym czy obiekt jest animowany
// ! parametr nie wykorzystywany obslugiwany
public:
// narysuj obiekt na płótnie
void virtual Narysuj(TCanvas *Canvas, Anaglif *ParAnaglif) {};
// przelicz wspolrzedne dla nowego t
void virtual PrzeliczDlaT(int t) {};
};

// Obiekty krawedziowe

class ObiektyKrawedziowe : public Obiekty {
protected:
Punkt3D *TabWierzcholki; // tablica wierzchołkow
        int IloscWierzchołkow;
        int IloscKrawedzi;
        int *TabKrawedzie; // tablica krawedzi
Parser *TabWyrazenia; // tablica wyrazen okreslajaca
// odczytanych z pliku
public:
ObiektyKrawedziowe(void);

// obiekty krawedziowe to wiemy jak rysowac
void Narysuj(TCanvas *Canvas, Anaglif *ParAnaglif);

void virtual PrzeliczDlaT(int t) {};
        ~ObiektyKrawedziowe();
};

// Odcinek

class Odcinek : public ObiektyKrawedziowe {
        private:
                void Init(void); // metoda okreslajaca tablice krawedzi
                void KonstrukcjaWierzchołkow(int t);
        public:
                Odcinek(void);
                Odcinek(char * wsp[]);
                ~Odcinek();
void PrzeliczDlaT(int t);
};

// Prostokat

```

```

class Prostokat : public ObiektyKrawedziowe {
    private:
        void Init(void); // metoda okreslajaca tablice krawedzi
        void KonstrukcjaWierzchoлков(int t); // konstrukcja wierzchołkow
    public:
        Prostokat(void);
        Prostokat(char * wsp[]);
        ~Prostokat();
void PrzeliczDlaT(int t);
};

// Graniastosłup

class Graniastosłup : public ObiektyKrawedziowe {
    private:
        void Init(void);
        void KonstrukcjaWierzchoлков(int t);
    public:
        Graniastosłup(void);
        Graniastosłup(char * wsp[]);
        ~Graniastosłup();
void PrzeliczDlaT(int t);
};

// Kula

class Kula : public ObiektyKrawedziowe {
    private:
        int nIloscRownPolud; // jaka gestosc siatki
// = ilosc poludnikow=ilosc rownolezniow
// odczytywana z okna ustawien podczas dzialnia konstruktora
        void Init(void);
        void KonstrukcjaWierzchoлков(int t);
    public:
        Kula(void);
        Kula(char * wsp[]);
        ~Kula();
void PrzeliczDlaT(int t);
};

```

Zgodnie ze schematem ze strony 5 dotyczącym definiowania struktury przechowywującej dane, mamy klasę abstrakcyjną o nazwie *Obiekty*. Klasa ta definiuje jedną zmienną o nazwie *CzyJestAnimacja*. Pole to informuje o tym, czy stworzona figura posiada jeden z parametrów zależny od czasu. Klasa ta również deklaruje dwie funkcje wirtualne, są to: *PrzeliczDlaT* oraz *Narysuj*.

Kolejne klasy dziedziczące, precyzują o jaki konkretnie obiekt chodzi. I tak klasa *Krawędź* definiuje swoje dwa parametry – punkty A i B na których rozpięta jest krawędź. Klasa ta posiada swoje konstruktory. Szczególnie istotny jest konstruktor który jako parametry otrzymuje 6 łańcuchów znaków. Konstruktor sprawdza czy to są liczby jeśli tak tworzona jest krawędź (bez możliwości animowania) o określonych parametrach. Jeśli jest to wyrażenie to tworzone są obiekty klasy *Parser* i wartość współrzędnych może być w każdej chwili przeliczona dla nowego *t* - zajmuje się tym wirtualna metoda *PrzeliczDlaT*.

Natomiast metoda `Narysuj` na przekazanym wskaźniku do obiektu klasy `TCanvas` rysuje anaglif - w tym przypadku dwie proste w różnych kolorach. Kolor pobierany jest ze statycznego pola klasy `Anaglif`. Podobnie punkty A i B są transformowane na płaszczyznę ekranu wykorzystując statyczne metody klasy `Anaglif` - `LeweOko` i `PraweOko`.

## Klasa `Anaglif` - przechowuje i przelicza parametry anaglif

```
class Anaglif {
private:
    char ** ExplodeString(char *,int ile);
    Punkt3D SkalaObrot(Punkt3D);
public:
    static TColor KolorPraweOko; // kolor obrysow dla prawego oka
    static TColor KolorLeweOko;  // kolor obrysow dla lewego oka
    static double OdlEkranOko; // odleglosc oczu od monitora
    static double RozstawOczu; // RozstawOczu
    static int IloscRownPolud;

    TColor KolorTla;           // kolor tla anaglif
    int GruboscLini;           // grubosc krawedzi
    int ObrotOX;                // kat obrotu ukladu wzgledem osi OX
    int ObrotOY;                // kat obrotu OY
    int ObrotOZ;                // ---- OZ
    double Odleglosc;           // odleglosc ekran - uklad
    double OffsetX;             // przesuniecie widoczności ekranu X
    double OffsetY;             // --- Y
    double Mnoznic;             // mnoznik skalujacy
    int t;                      // czas dla animacji
    TCanvas* ACanvas;           // wskaznik do obiektu klasy TCanvas
                                // na ktorym bedzie rysowany anaglif
    char *OpisAnaglif;          // opis - pierwsza linia, jesli zaczyn sie od #
    // deklaracja tablicy wczytanych obiektow
    Obiekty **TablicaObjektow;
    long IleObjektow;

    // konstruktor, wiaze ACanvas z odpowiednim plotnem, otwiera plik
    Anaglif(TCanvas*,char*);

    ~Anaglif();

    // kolejne metody zmieniajace powyzsze parametry
    static void ZmienKolorOkularow(TColor lewe,TColor prawe);
    void UstalCzas(int nowy);

    // metoda zwraca string z opisem anaglif - pamiec, ilosc obiektow itd;
    char *Info();

    // metoda przerysowywujaca anaglif
    void NarysujAnaglif(TCanvas * Canvas);
    void NarysujAnaglif();
}
```

```

// metoda odczytująca anaglif z plik
void OdczytajAnaglif(char * plik);

// dwie metody -> przekształcające Punkt3D w układzie
// w którym znajdują się figury na Punkt2D nadający się
// do wyświetlenia
Punkt2D LeweOko(Punkt3D);
Punkt2D PraweOko(Punkt3D);
};

```

Powyższa klasa jest szczególnie istotna z punktu widzenia całego programu. Jest to klasa przechowująca parametry anaglif, tablice wskaźników do obiektów oraz metody generujące anaglif. Znaczenie składników ich zostało opisane w komentarzach. Wskaźnik do obiektu tej klasy zawiera każde okienko typu MDIChild formy głównej.

## 8 Testowanie, określenie niezmienników

Uzyskiwany efekt trójwymiarowości po ubraniu okularów jest zadawalający. Ważne jest aby dobrze dostroić okulary (okno ustawień).

Samą konstrukcję anaglif również można sprawdzić, i tak: zmniejszenie rozstawu oczu do zera powoduje, zgodnie z oczekiwaniami, zlanie się obrazków dla prawego i lewego oka. Zwiększanie tej zmiennej powoduje natomiast oddalanie się obrazów.

Zwiększanie zmiennej Odległość (odległości od ekranu do początku wirtualnego układu współrzędnych za ekranem) skutkuje zmniejszeniem i oddaleniem się obrazków. Można to wyjaśnić w następujący sposób: weźmy dowolny punkt leżący za ekranem i połączmy go odcinkami z lewym i prawym okiem. Punkty przecięcia tych odcinków z płaszczyzną ekranu wyznaczają ekranowe współrzędne naszego punktu dla lewego i prawego oka. Jeśli teraz będziemy nasz punkt oddalać, to odcinki stają się równoległe, a odległość ich punktów przecięcia z ekranem rośnie.

## 9 Wdrożenie, raport i wnioski

Wszystkie podstawowe wymagania projektu zostały wykonane. Narzucony przez nas bagaż dodatkowych opcji udało się w większości zrealizować. Niestety podczas prac nad projektem pojawiło się kilka komplikacji:

- Problem z wydrukami. Okazało się że Borland bardzo nie chętnie kopiuje rysunek (StretchDraw) na obiekt Canvas klasy TPrinter (ponoć zależy to od drukarki). Da się to zrobić stosując odpowiednie metody, ale niestety brakło na to czasu. Co gorsza link ze strony przedmiotu na temat drukowania w Borlandzie nie działa.
- Dość późno ujawnił się problem z parserem, który nie poprawnie odczytuje wyrażenia zaczynające się od znaku minus (np.  $-\sin(t)$  lub  $-(1 + 2) * 3$ ). Szybka poprawa nie była możliwa i rozwiązanie przejściowe polega na dość prymitywnej sztuczce uzyskiwania znaku ujemnego np  $(0-1)*\sin(t)$ .
- Wersje językowe nie zostały zaimplementowane z braku czasu.



- Plik pomocy został znacząco uproszony.

Program nieco możnaby rozubować w kolejnych wersjach o

- obsługę brył o kolorowych ścianach
- obroty z wykorzystaniem myszki (coś na wzór AutoCAD-a)
- dorobić moduł edycyjny, bo ręczna konstrukcja obrazków jest bardzo żmudna.