Projekt

Sterowniki robotów

Raport – etap III

Prymitywny animowany wyświetlacz 2.0
PAW 2.0

Skład grupy (IV):
Michał Antoszko, 226439
Maciej Mikusek, 226471

Termin: wtTN11/17

Prowadzący:
mgr inż. Wojciech Domski

28 lutego 2019

Spis treści

1 Opis projektu **2**
1.1 Cele projektu 2
1.2 Ogólne założenia 2

2 Konfiguracja mikrokontrolera **3**
2.1 Konfiguracja pinów 3
2.2 Peryferia 5
2.3 Konfiguracja portów oraz zegara 7

3 Rozszerzenia projektu PAW **8**
3.1 Rekonstrukcja kodu źródłowego 8
3.2 Projekcja obrazu 3D na wyświetlacz 2D 9
3.3 Pokazowa grafika dla prymitywów w 3D 10
3.4 Funkcja rysująca ściany 11

4 Podsumowanie **12**
4.1 Zadania niezrealizowane 12
4.2 Zadania zrealizowane 12
4.3 Github 12

1 Opis projektu

1.1 Cele projektu

Celem projektu jest rozbudowa istniejącego projektu „Prymitywny animowany wyświetlacz” o akronimie PAW, wykonanego przez Paulinę Porczyńską i Juliusza Stanisława Tarnowskiego. Poprzez rozbudowę rozumie się wprowadzenie dodatkowych funkcjonalności, którymi będą: dodatkowy bufor oraz symulacja obrazu trójwymiarowego na wyświetlaczu LCD.

Aktualnie w projekcie zastosowany jest model z pojedynczym buforem, co powoduje widoczne i spowolnione przeładowywanie kolejnych klatek animacji. Remedium na tę niedogodność może stanowić podwójny bufor. Dodatkowy bufor (w dalszej częsci zwany drugim buforem) będzie przechowywał kolejną klatkę, która będzie miała zostać wyświetlona po klatce widocznej aktualnie na wyświetlaczu. Dzięki temu, że w tle ładowana będzie następna klatka, skrócony zostanie czas oczekiwania, w wyniku czego wyświetlany obraz będzie płynny.

Pierwotni autorzy projektu stworzyli prymitywy pozwalające na wyświetlanie obiektów w 2D. Kolejnym naturalnym rozszerzeniem tego projektu, byłoby dodanie możliwości oglądania prymitywów w 3D. Korzystając z odpowiednich przekształceń matematycznych można rzutować obiekty z *wirtualnej* przestrzeni 3D na wyświetlacz dwuwymiarowy.

1.2 Ogólne założenia

Drugi bufor zostanie zaimplementowany z wykorzystaniem RAMu, którego obsługa została już częściowo zrealizowania. Możliwe jest jedynie najprostsze zapisywanie i czytanie danych.

Do implementacji rysowania, uwzględniając obracanie prymitywów w 3D, także istnieją już pewne fundamenty. Biblioteki `PAW_Vector.h` oraz `PAW_Matrix.h` napisano z myślą o pracy w trzech wymiarach – wektory są wymiaru 4×1, a macierze 4×4.

Rozwinięcie tego aspektu PAWia będzie wymagało jedynie wprowadzenia odpowiednich operacji geometrycznych na wyższych poziomach projektu – zaczynając od `PAW_Primitive.h`.

2 Konfiguracja mikrokontrolera

Zarówno konfiguracja pinów, jak i peryferiów pozostaje niezmieniona w stosunku do poprzedniej iteracji projektu PAW.

2.1 Konfiguracja pinów

PIN nr	PIN	FUNCTION	LABEL
8	PC14/OSC32_IN*	RCC_OSC32_IN	PC14-OSC32_IN
9	PC15/OSC32_OUT*	RCC_OSC32_OUT	PC15-OSC32_OUT
10	PF0	FMC_A0	A0
11	PF1	FMC_A1	A1
12	PF2	FMC_A2	A2
13	PF3	FMC_A3	A3
14	PF4	FMC_A4	A4
15	PF5	FMC_A5	A5
19	PF7	SPI5_SCK	SPI5_SCK [L3GD20_SCL/SPC]
20	PF8	SPI5_MISO	SPI5_MISO [L3GD20_SDO]
21	PF9	SPI5_MOSI	SPI5_MOSI [L3GD20_SDA/SDI/SDO]
22	PF10	LTDC_DE	ENABLE [LCD-RGB_ENABLE]
23	PH0/OSC_IN	RCC_OSC_IN	PH0-OSC_IN
24	PH1/OSC_OUT	RCC_OSC_OUT	PH1-OSC_OUT
26	PC0	FMC_SDNWE	SDNWE
27	PC1	GPIO_Output	NCS_MEMS_SPI [L3GD20_CS_I2C/SPI]
28	PC2	GPIO_Output	CSX [LCD-RGB_CSX]
34	PA0/WKUP	GPIO_EXTI0	B1 [Blue PushButton]
35	PA1	GPIO_EXTI1	MEMS_INT1 [L3GD20_INT1]
36	PA2	GPIO_EXTI2	MEMS_INT2 [L3GD20_INT2]
37	PA3	LTDC_B5	B5
40	PA4	LTDC_VSYNC	VSYNC
42	PA6	LTDC_G2	G2
43	PA7	GPIO_Output	ACP_RST
44	PC4	GPIO_Output	OTG_FS_PSO [OTG_FS_PowerSwitchOn]
45	PC5	GPIO_EXTI5	OTG_FS_OC [OTG_FS_OverCurrent]
46	PB0	LTDC_R3	R3
47	PB1	LTDC_R6	R6
48	PB2/BOOT1	GPIO_Input	BOOT1
49	PF11	FMC_SDNRAS	SDNRAS
50	PF12	FMC_A6	A6
53	PF13	FMC_A7	A7
54	PF14	FMC_A8	A8
55	PF15	FMC_A9	A9
56	PG0	FMC_A10	A10
57	PG1	FMC_A11	A11
58	PE7	FMC_D4	D4
59	PE8	FMC_D5	D5
60	PE9	FMC_D6	D6
63	PE10	FMC_D7	D7
64	PE11	FMC_D8	D8
65	PE12	FMC_D9	D9
66	PE13	FMC_D10	D10
67	PE14	FMC_D11	D11
68	PE15	FMC_D12	D12
69	PB10	LTDC_G4	G4
70	PB11	LTDC_G5	G5

Tabela 1: Konfiguracja pinów

PIN nr	PIN	FUNCTION	LABEL
73	PB12*	USB_OTG_HS_ID	OTG_FS_ID
74	PB13*	USB_OTG_HS_VBUS	VBUS_FS
75	PB14*	USB_OTG_HS_DM	OTG_F_DM
76	PB15*	USB_OTG_HS_DP	OTG_FS_DP
77	PD8	FMC_D13	D13
78	PD9	FMC_D14	D14
79	PD10	FMC_D15	D15
80	PD11	GPIO_Input	TE [LCD-RGB_TE]
81	PD12	GPIO_Output	RDX [LDC-RGB_RDX]
82	PD13	GPIO_Output	WRX_DCX [LCD-RGB_WRX_DCX]
85	PD14	FMC_D0	D0
86	PD15	FMC_D1	D1
89	PG4	FMC_BA0	BA0
90	PG5	FMC_BA1	BA1
91	PG6	LTDC_R7	R7
92	PG7	LTDC_CLK	DOTCLK [LCT-RGB_DOTCLK]
93	PG8	FMC_SDCLK	SDCLK
96	PC6	LTDC_HSYNC	HSYNC
97	PC7	LTDC_G6	G6
99	PC9	I2C3_SDA	I2C3_SDA [ACP/RF_SDA]
100	PA8	I2C3_SCL	I2C3_SCL [ACP/RF_SCL]
103	PA11	LTDC_R4	R4
104	PA12	LTDC_R5	R5
105	PA13*	SYS_JTMS-SWDIO	SWDIO
109	PA14*	SYS_JTCK-SWCLK	SWCLK
110	PA15	GPIO_EXTI15	TP_INT1 [Touch Panel]
111	PC10	LTDC_R2	R2
114	PD0	FMC_D2	D2
115	PD1	FMC_D3	D3
117	PD3	LTDC_G7	G7
122	PD6	LTDC_B2	B2
125	PG10	LTDC_G3	G3
126	PG11	LTDC_B3	B3
127	PG12	LTDC_B4	B4
128	PG13	GPIO_Output	LD3 [Green Led]
129	PG14	GPIO_Output	LD4 [Red Led]
132	PG15	FMC_SDNCAS	SDNCAS
135	PB5	FMC_SDCKE1	SDCKE1
136	PB6	FMC_SDNE1	SDNE1 [SDRAM_CS]
139	PB8	LTDC_B6	B6
140	PB9	LTDC_B7	B7
141	PE0	FMC_NBL0	NBL0 [SDRAM_LDQM]
142	PE1	FMC_NBL1	NBL1 [SDRAM_UDQM]

Tabela 2: Konfiguracja pinów

2.2 Peryferia

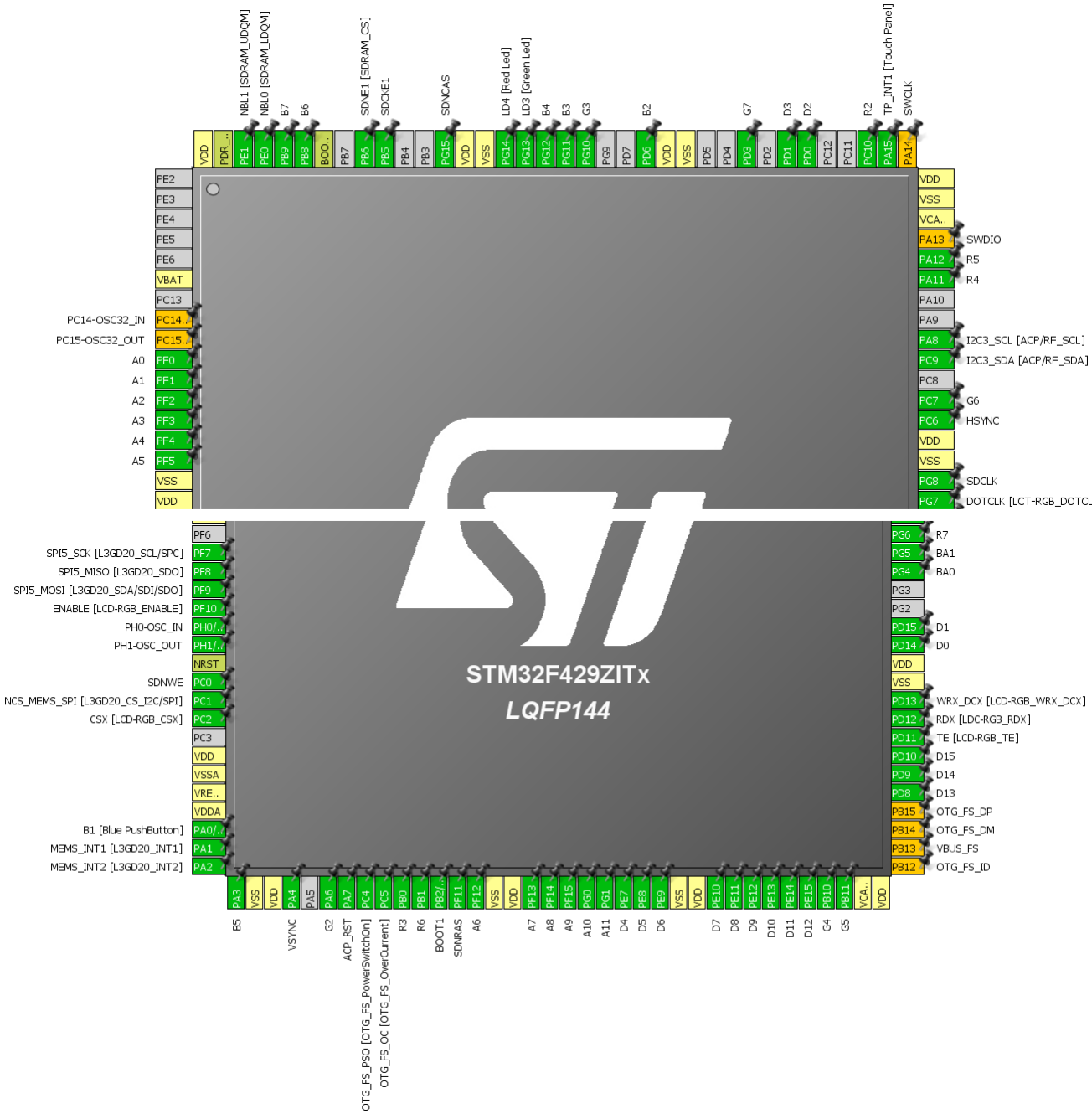
DMA2D	
mode:	Activated
Parameter Settings	
Basic Parameters:	
Transfer Mode	Memory to Memory
Color Mode	RGB565
Output Offse	0
Foreground layer Configuration:	
DMA2D Input Color Mode	RGB565
DMA2D ALPHA MODE	No modification of the alpha channel value
FMC	
Clock and chip enable:	SDCKE1+SDNE1
Internal bank number:	4 banks
Address:	12 bits
Data:	16 bits
Byte enable:	set
SDRAM 1	
Bank SDRAM	bank 2
Number of column address bits	8 bits
Number of row address bits	12 bits
CAS latency	3 memory clock cycles
Write protection	Enabled
SDRAM common clock	2 HCLK clock cycles
SDRAM common burst read	Disabled
SDRAM comon read pipe delay	2 HCLK clock cycles
SDRAM timing in memory clock cycles	
Load mode register to active delay	16
Exit self-refresh delay	16
Self-refresh time	16
SDRAM common row cycle delay	16
Write recovery time	16
SDRAM common row precharge delay	16
Row to column delay	16

Tabela 3: Kluczowe peryferia

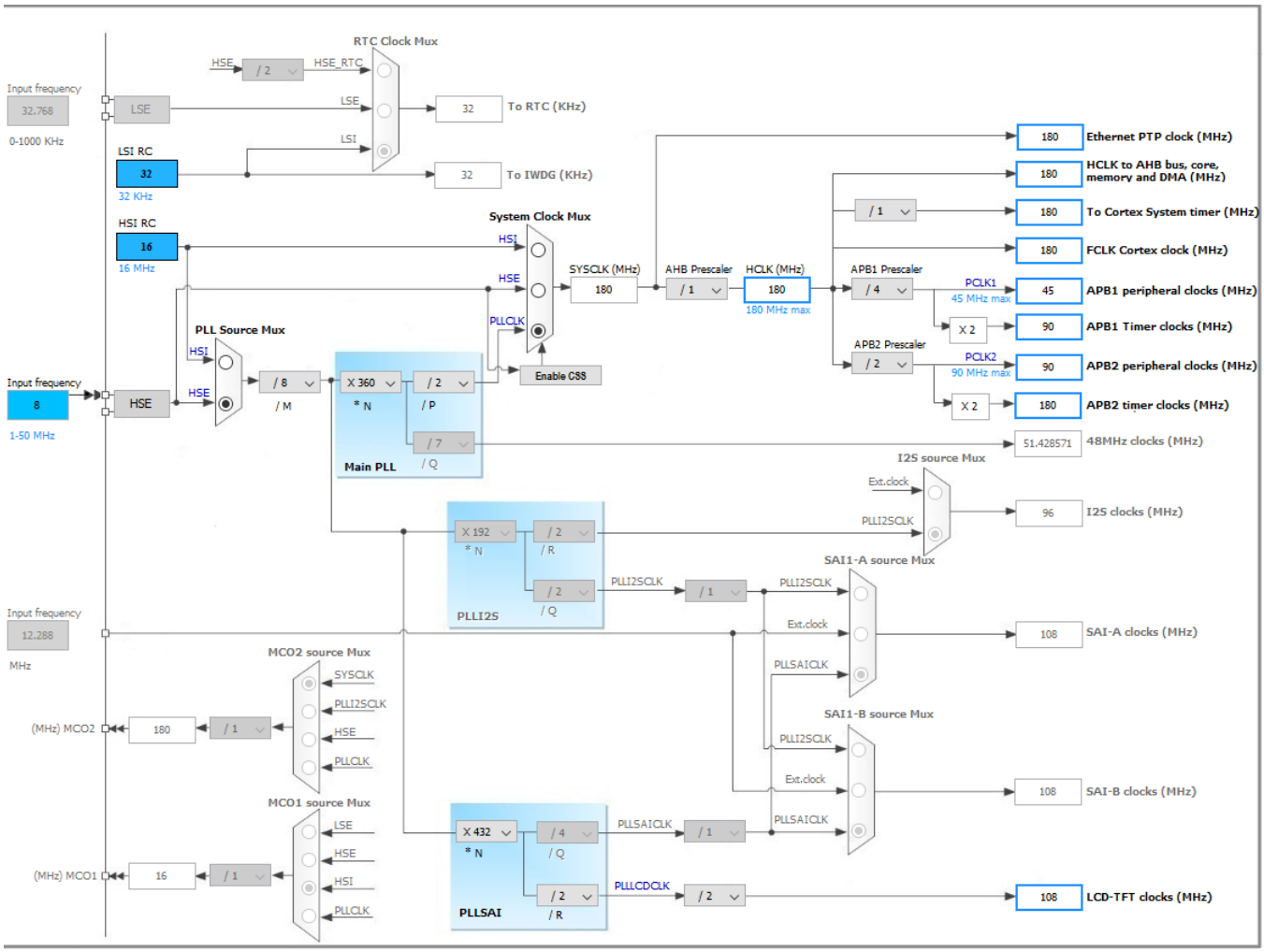
LTDC	
Display Type:	RGB666 (18 bits)
Parameter Settings	
Synchronization for Width	
Horizontal Synchronization Width	8
Horizontal Back Porch	7
Active Width	320
Horizontal Front Porch	6
HSync Width	7
Accumulated Horizontal Back Porch Width	14
Accumulated Active Width	334
Total Width	340
Synchronization for Height	
Vertical Synchronization Height	4
Vertical Back Porch	2
Active Height	480
Vertical Front Porch	2
VSync Height	3
Accumulated Vertical Back Porch Height	5
Accumulated Active Height	485
Total Height	487
Signal Polarity	
Horizontal Synchronization Polarity	Active Low
Vertical Synchronization Polarity	Active Low
Data Enable Polarity	Active Low
Pixel Clock Polarity	Normal Input
BackGround Color	
Red	0
Green	0
Blue	0

Tabela 4: Kluczowe peryferia

2.3 Konfiguracja portów oraz zegara



Rysunek 1: Porty



Rysunek 2: Zegar

3 Rozszerzenia projektu PAW

3.1 Rekonstrukcja kodu źródłowego

Duża część programu, z którym został rozpoczęty projekt PAW 2.0 została przepisana na nowo. Prace zostały podjęte, ze względu na brak hermetyzacji i częściowy brak polimorfizmu w projekcie. Struktura projektu została zachowana, natomiast klasy matematyczne *Vector* i *Matrix* zostały napisane całkowicie od nowa, co wymagało poprawnego połączenia reszty programu. Wymagało to przepisania pozostałej części programu, aby całość była ze sobą kompatybilna. Poprawiono kilka błędów oraz usunięto zbędne linie kodu. Zastosowano lepszą ochronę przed niewłaściwym wywoływaniem metod oraz łatwiejszy sposób debugowania, związany z wysyłaniem błędów na konsolę/UART.

Kod został podzielony na pliki nagłówkowe oraz pliki źródłowe. Kod pisany był z myślą o przyszłym przepisaniu go na język C, w związku z tym w większości programu wykorzystywane były konstrukcje, które są kompatybilne z językiem C, lub wymagają niewielkiej zmiany (np. brak *range loopów*, zastosowanie przeciążenia operatorów). Funkcje matematyczne zostały wydzielone do modułu nazwanego *PAW_Math*. W fragmentach kodu, które tego wymagały zastosowano jawną konwersje zamiast niejawnej.

3.2 Projekcja obrazu 3D na wyświetlacz 2D

Korzystając z wcześniej zaimplementowanych struktur dodano możliwość projekcji obrazu 3D na wyświetlacz 2D. Wykorzystano do tego macierz projekcji, która tworząc perspektywę symuluje grafikę 3D. Zaimplementowano działania takie jak:

- rotacja,
- translacja,
- skalowanie,
- generowanie sześcianu o dowolnej długości boku.

Macierz projekcji:

$$P = \begin{bmatrix} \frac{2 \cdot near}{(right-left)} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2 \cdot near}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & \frac{-2 \cdot far \cdot near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Wartości użyte w macierzy reprezentują maksymalne wartości dla każdej osi w kierunkach ujemnych i dodatnich, zatem:

- oś x – left (-), right (+),
- oś y – top (+), bottom (-),
- oś z – near (+), far (-).

Przyjęte wartości zakładają, że środek układu współrzędnych jest w punkcie (0, 0, 0), a przestrzeń robocza jest sześcianem o długości boku 120 pikseli.

Macierze rotacji:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

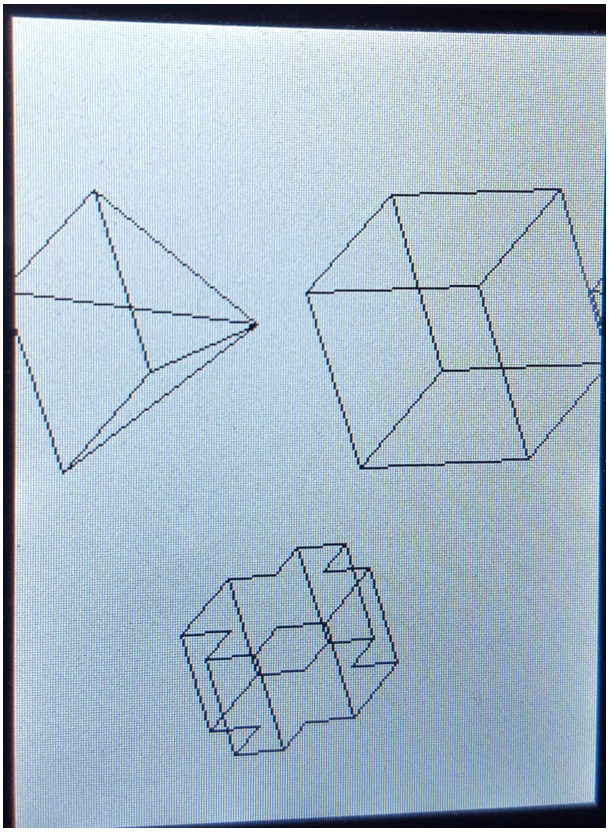
Piksele powstają w wyniku działania:

$$2D_{pixel} = P \cdot R_z(\gamma) \cdot R_y(\beta) \cdot R_x(\alpha) \cdot 3D_{pixel} + T$$

Krótki opis i ilustracje do programu na końcu dokumentu.

3.3 Pokazowa grafika dla prymitywów w 3D

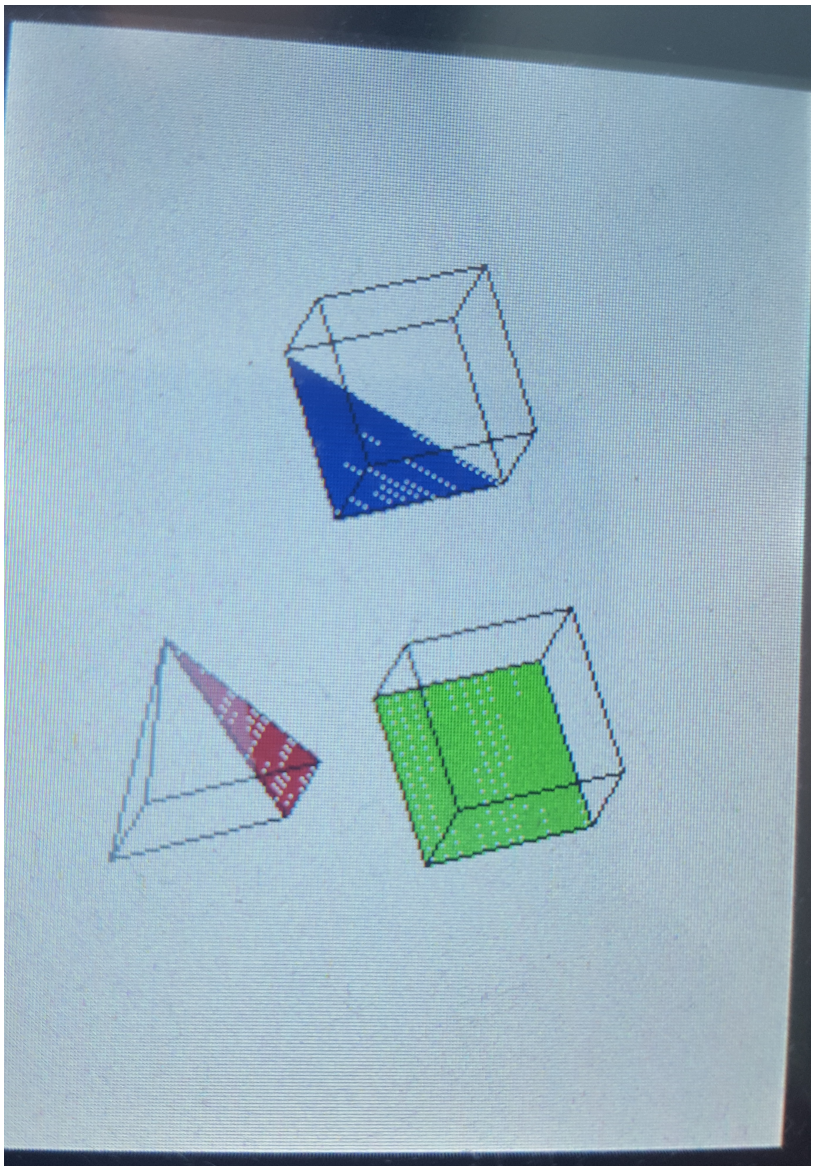
W celu zaprezentowania działania projekcji 3D na dwuwymiarowy ekran, dodano parę ruchomych kształtów. Wszystkie obiekty wyrzeźbiono za pomocą prostych linii łączących punkty.



Rysunek 3: Wygląd narysowanych ścianek

3.4 Funkcja rysująca ściany

Dodano metodę *paint*() do klasy *Figure*. Przyjmuje ona na wejściu (jako argumenty) dwie linie, pomiędzy którymi zostanie narysowana „ścianka”. Linie te muszą być równej długości. Ściana składa się z linii rysowanych w pętli.



Rysunek 4: Wyświetlacz z grafiką

4 Podsumowanie

4.1 Zadania niezrealizowane

Nie zrealizowano kluczowego aspektu projektu – podwójnego buforowania obrazu na ekran LCD. Wyświetlana animacje pozostaje rażąca dla oczu i niepłynna.

4.2 Zadania zrealizowane

Zrealizowano część projektu związaną z wyświetlaniem prostych kształtów w trzech wymiarach. Oprogramowanie umożliwia teraz manipulowanie dowolnym narysowanym obiektem w 3D, jednocześnie wiernie oddając ich wygląd w każdym momencie ruchu czy obrotu.

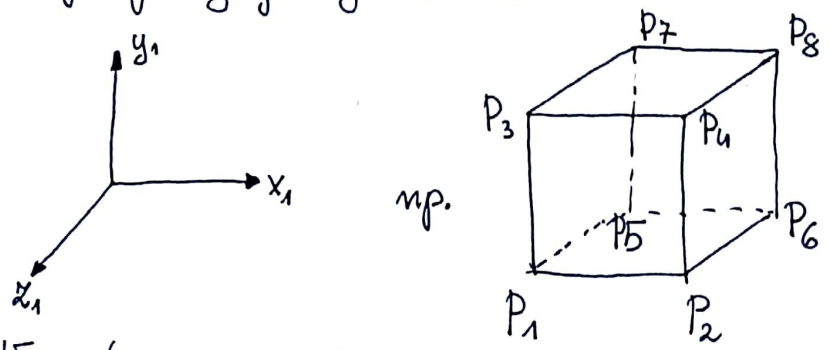
Dodatkowo, rekonstrukcja kodu źródłowego znacznie ułatwi pracę osobom potencjalnie kontynuującym projekt w przyszłości. Kod jest łatwiejszy zarówno w interpretacji i użytku, jak i w nawigowaniu.

Na koniec zrealizowano interfejs pozwalający „malować" wybrane obszary bryły lub tworzyć „ścianki" pomiędzy dwoma liniami o różnym położeniu.

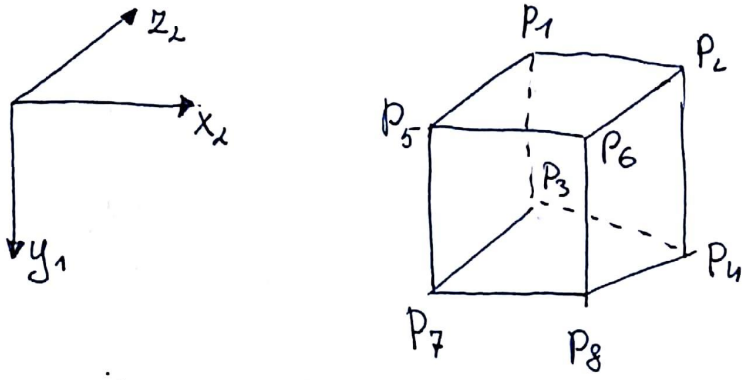
4.3 Github

Wszystkie commity dostępne pod adresem poniżej.
`https://github.com/sterowniki-robotow-projekt/main-repo/commits/master`

Z perspektywy wigłownika



Wewnętrzne wsp. wyświetlenia



Z tego wynika, że:

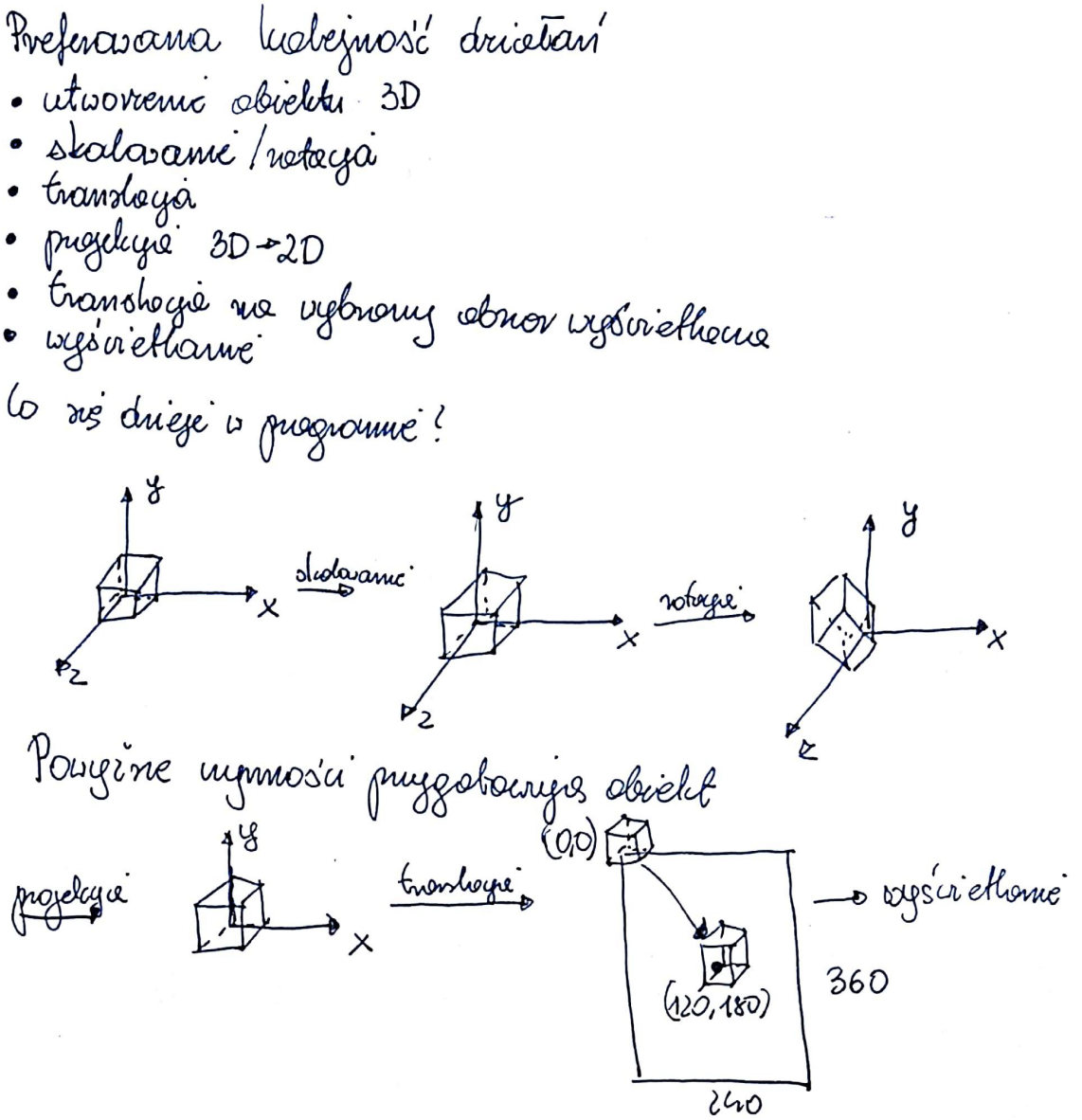
$$x_2 = x_1$$
$$y_2 = -y_1$$
$$z_2 = -z_1$$

Wsp. wigłownika zamieniane są na wsp. wyświetlenia wg. powyższej formuły.

Color coding:

P_1 - żółty	P_5 - różowy
P_2 - czerwony	P_6 - fioletowy
P_3 - zielony	P_7 - cyan
P_4 - niebieski	P_8 - pomarańczowy

Rysunek 5: Układy współrzędnych



Rysunek 6: Kolejność operacji