**ARKO**

3.7 Grafika Żółwiowa

Tomasz Dakowicz

28.04.2015

**Polecenie:** *Zrealizować program generujący na podstawie pliku tekstowego plik w formacie BMP. Plik wejściowy zawiera polecenia grafiki żółwiowej (ang. turtle graphics) według podanej w następnym punkcie składni.*

1 Założenia

-Program pobiera instrukcje grafiki żółwiowej z pliku turtle.txt

-Założona składnia realizuje założone instrukcje

-Plik wynikowy zawiera reprezentację graficzną rezultatu wykonania instrukcji, wielkość bitmapy to 160x120

2 Struktura programu

Strukturę programu pokażę w punktach, gdzie wyszczególniłem najważniejsze operacje programu:

1. Odczyt instrukcji – odczytywane są kolejno komendy, które muszą zachować zgodność z założoną składnią; plik jest odczytywany bajt po bajcie, do każdego napisu sprawdzana jest pasująca komenda; niektóre instrukcje wymagają dodatkowych liczb, które również są sprawdzane (aby zachować spójność i przejrzystość, wielkość powolnej liczby została ograniczona do 3 cyfr, czyli do 999); jeśli stwierdzono zgodność ze składnią, instrukcja jest wykonywana

(w innym przypadku jest wyłapywany błąd, co skutkuje zakończeniem programu);

1. Instrukcja ‘naprzod’ – dla danego kąta oblicza w której oktancie znajdować się będzie punkt końcowy; dla każdego przesunięcia obliczana jest różnica na osi x i y na podstawie odpowiedniej wartości sinusa, która jest pobierana z pamięci (zapisana jak typ float); operacja wyznaczania opiera się na działaniach zmiennoprzecinkowych; każdy nowy czarny piksel jest wstawiany w odpowiednie miejsce w buforze danych
2. Reprezentacja danych w postaci bitmapy – na początku bufor danych jest zapełniany samymi ‘białymi’ pikselami (3 bajty na piksel); po przetworzeniu wszystkich instrukcji bufor ten jest przepisywany do pliku ( wcześniej oczywiście zapisywany jest nagłówek (54B) ); sumaryczna wielkość pliku wynosić będzie (3\*160\*124 + 54) B
3. Spis używanych rejestrów i ich znaczeń został wyjaśniony w górnym komentarzu (w pliku z programem .asm)

3 Opis implementacji algorytmu

W celu rasteryzacji prostych użyty został algorytm Bresenhama - na początku wyznacza się oktant w której się znajduje punkt końcowy – w zależności od tego zmienia osie ‘ruchu’ algorytmu i kierunek przyrostu na x i y; dx i dy wyznacza się poprzez użycie odpowiednich wartości sinusa korzystając z operacji zmiennoprzecinkowych; wyznaczanie pozycji kolejnego, nowego piksela uzyskuje się poprzez porównywanie zmiennej decyzyjnej; w momencie przekroczenia granic obrazu (160x120) wyznaczany jest nowy punkt, który się znajduje po drugiej stronie od tego przekroczenia

( gdzie warto zauważyć, że x może przyjmować wartości z zakresu <0, 159>,

y z zakresu <0, 119> ); od tego miejsca jest kontynuowane rysowanie, z dotychczasowym nachyleniem.

4 Użyte dane

W programie zostały użyte przybliżone wartości sinusa <0,90> zapisane jako typ float; oprócz tego: bufor na komendy, bufor na dane obrazu, ciągi znaków zawierające dostępne instrukcje oraz 54 B nagłówka pliku BMP

4 Testy

Testy są zawarte w folderze ‘testy’, do każdego z nich zawarty został zbiór instrukcji grafiki żółwiowej i bitmapa wynikowa

|  |  |  |
| --- | --- | --- |
| Nazwa testu | Cel testu | Poprawność wyniku |
| co\_30\_stopni | bada rysowanie prostych linii co 30 stopni, począwszy od 0 (bada sprawność rysowania prostej o nachyleniu w każdej oktancie) | poprawny |
| co\_45\_stopni | to samo co wyżej, tylko że co 45 stopni | poprawny |
| ćwiartka\_okręgu | symuluje ćwiartkę okręgu | poprawny |
| domyślny | test z archiwum przedmiotu | poprawny |
| kształt\_zamknięty | rysuje obiekt, w którym nachylenie zmienia się o 135 stopni za każdym razem; bada spójność | poprawny |
| linia\_prosta | rysuje prostą linię o nachyleniu 0 | poprawny |
| linia\_prosta\_poza\_granice | rysuje prostą linię o nachyleniu 45, która jednocześnie ma długość przekraczającą wymiary obrazka (linia powinna zostać odbita z drugiej strony) | poprawny |