Tomass Zutis tz18004

Apraksts

Izvēlētais risinājums MOP kursa projektā

1.) Kāda funkcionalitāte realizēta? Kā tā realizēta?

Projektā izstrādāta ir grafiskā bibliotēka, kas spēj "zīmēt" 2D plaknē tādas figūras, kā punkts, līnija, aplis un trijstūris. Šīs figūras tiek ierakstītas buferī (idejiski implementēts kā 2D matrica), kas vēlāk tiek izdrukātas uz ekrāna.

Realizētās funkcijas:

a.) **Krāsas uzstādīšana**. Krāsa tiek uzstādīta, izmantojot definētu struktūru, un nosūtīta uz assambler failu, kur šo krāsu izmanto pikseļu aizpildīšanai buferī, līdz tā nomainīta.
b.) **Pikseļa zīmēšana**. Saņem tā x un y koordinātas un ievieto to attiecīgajā vietā buferī. Ņem vērā krāsu un zīmēšanas operāciju. Zīmēšanas operācijas spēj mainīt bufera iznākumu, lietojot loģiskās operācijas, kas minētas aprakstā. Manā implementācijā šī funkcija arī pārbauda vai netiek zīmēts pirmo reizi, jo pirms pirmās zīmēšanas jānotīra buferis(kam izveidota atsevišķa funkcija).
c.) **Līnijas zīmēšana**. Saņem koordinātas sākumu un beigu punktiem. Pašu līniju zīmē, izmantojot uzlabotu Bresenhema līnijas zīmēšanas algoritmu. Visplašāk pieejamajam algoritmam bija jāmeklē uzlabojumi, jo pie noteikta līnijas slīpuma (lielāks y slīpums, nekā x slīpums) tika zīmēta nekorekta līnija, kas neaiziet līdz galapunktam.
d.) **Riņķa līnijas zīmēšana**. Arī izmantots Bresenhema "Midpoint circle algorithm", kas riņķi fiziski iezīmē, izmantojot apakš-funkciju, kas pēc katra aprēķina iezīmē 8 pikseļus apkārt esošajai konfigurācijai. Funkcija saņem apļa viduspunktu (x,y) un rādiusu.
e.) **Trijstūra aizpildīšana**. Šis, visgarākais algoritms, aizpilda trijstūra kontūras, zīmējot horizontālas līnijas vairākos cēlienos(konfigurācijās) līdz tiek aizpildīts viss laukums. Bieži izmanto apakš-funkciju SWAP, kas apmaina divus mainīgos (piemēram, x1 un x2) vietām. Kā parametrus saņem triju virsotņu koordinātas.
f.) **Bufera adreses atgriešana**, lai varētu tam piekļūt assambler programmā. **Bufera platuma un garuma atgriešana**, lai spētu kontrolēt bufera apstaigāšanu un **Rāmja izdrukāšana**, kas izdrukā buferi uz ekrāna.

2.) Kas nav realizēts vai nedarbojas, kāpēc?

Visas prasības uzskatu par apmierinātām, ja vien vērtēšanā veiktie testi nenorādīs savādāk. Iespējams, kods varētu būt efektīvāks un ātrāks. Saskatu problēmas ar bufera tīrīšanas pārbaudi un pikseļu operācijas pārbaudi, katru reizi, kad tiek zīmēts individuāls pikselis. Manuprāt, šādas darbības būtu jāveic funkcijās augstāk hierarhijā, taču šoreiz, pievienojot šīs funkcijas beigās, tā tehniski sanāca. Riņķa funkcijā vienā vērtību padošanas vietā no reģistra/steka uz citu, pastāvīgi radās kļūmes, kuras nespēju izsekot līdz cēlonim. Kā pagaidu risinājumu izvēlējos vērtību nodošanas veida maiņu, taču pilnīgu drošību par kļūdas neesamību tas nedod.

3.) Kas bija lielākais izaicinājums projektā?

Lielākie izaicinājumi bija:

a.) Atrast zīmēšanas algoritmus vai to implementācijas, kas tiešām strādātu visos iedomājamos gadījumos.
b.) Pārnest garās funkcijas no testēšanas vides C valodā uz assambler kodu.
c.) Kļūmju trasēšana. Neuzmanības dēļ tika pazaudēts daudz laika uz triviālām kļūmēm.

4.) Ko nākamreiz darītu citādi, ja vispār?

Diemžēl par profesora rādīto piemēru, kā tieši pārkompilēt .c failu par ARM Xscale assembly kodu atcerējos tikai tad, kad paša spēkiem jau izmisīgi centos pārrakstīt trijstūra funkciju. Likās, ka garš ģenerētais kods īsti nederēs iekļaušanai manā programmā. Tā tomēr ir ātrākā un nekļūdīgākā metode, kā pārnest šādus garus algoritmus. Nākamreiz to noteikti atcerēšos un pielietošu uzreiz, lai ietaupītu vairākas stundas darba.

5.) Citi komentāri par risinājumu.

Kopumā interesanta alternatīva eksāmenam. Noder cilvēkiem (arī man), kam ir grūtāk risināt uzdevumus ar mazu laika limitu.