MOP kursa projekts

Arnis Priedītis

Kursa projekta ietvaros ir realizēta visa prasītā/aprakstītā funkcionalitāte. Arī loģiskās operācijas veikšana starp fona un krāsojamo krāsu atkarībā no “op” lauka vērtības.

Līnijas zīmēšanas algoritmam izmantots Brezenhama algoritms, tikai vēl papildināts ar to, ka var zīmēt līniju starp jebkādi savstarpēji novietotiem punktiem, ne tikai tādiem, ka pirmais punkts ir pa kreisi no otrā un ka līnijai ir <=45 grādu leņķis ar x asi. Ja būtu vēl laiks vai gribēšana, tad gribētos vēl ieviest līnijas apcirpšanu, lai rēķinātu tikai tos līnijas punktus, kas būs iekšā ekrānā. Bet šobrīd pixel() funkcija parūpējas, lai aprēķinātie punkti, kas ir ārpus ekrāna vienkārši netiktu zīmēti/rakstīti.
Izmantotie avoti: https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm#All_cases
https://www2.cs.sfu.ca/CourseCentral/361/inkpen/Notes/361_lecture5.pdf

Riņķa līnijas zīmēšanai izmantoju algoritmu, kas aprakstīts šeit:
https://www2.cs.sfu.ca/CourseCentral/361/inkpen/Notes/361_lecture5.pdf
Savā ziņā līdzīgs Brezenhama algoritmam. Aprēķina loku tikai 45 grādu leņķī, pārējos punktus uzzīmē ar palīgfunkcijas palīdzību, kas “spoguļo” kārtējo aprēķināto punktu uz pārējiem “oktantiem” (kvadrants vēl uz pusēm sadalīts).

Trijstūra aizpildīšanai izmantoju pieeju, kas šeit aprakstīta:
https://kristoffer-dyrkorn.github.io/triangle-rasterizer/1
Būtībā tiek noteiktas minimālās/maksimālās x un y koordinātes trijstūra punktiem, no kā iegūst taisnstūri ar potenciālajiem trijstūra pikseļiem. Tad katram pikselim aprēķina, vai tas ir iekšā trijstūrī. Tam izmanto vektoriālā reizinājuma/determinanta formulas – pārbauda, vai pikselis ir pa kreisi no visām trim malām. Varbūt šo implementāciju es gribētu izdarīt citādāk, jo te tomēr izmanto daudz reizināšanas operāciju uz katru pikseli.
Cita pieeja būtu izmantot to pašu Brezenhama algoritmu, lai inkrementētu divas trijstūra malas, piem., no augšas uz leju un kad abas pavirzās par 1 pikseli uz leju, tad aizpildīt horizontālu līniju starp šiem līniju punktiem. Tas prasītu vēl vajadzības gadījumā trijstūri sadalīt divos, lai katram būtu viena no malām horizontāla.

Viena pikseļa aizpildīšanas funkcijā no sākuma tiek pārbaudīts, vai prasītais pikselis ir iekšā FrameBuffer. Ja nav, tad ignorē šo pieprasījumu. Citādi pārbauda “op” vērtību. Ja tā nav “copy”, tad dabū šībrīža fona krāsu no FrameBuffer, izpilda attiecīgo operāciju un ieraksta rezultātu atpakaļ buferī. Ja ir “copy”, tad vienkārši ieraksta krāsu norādītājās koordinātēs.

Nedaudz piņķerīgi sākumā bija izplānot, kā FrameBuffer, tā izmēri un tekošā krāsa tiks reprezentēti programmā. Beigās paliku pie tā, ka šīm lietām ir globāli mainīgie, kas iekš agra.h tiek tikai deklarēti, izmantojot extern. Tad iekš agra_main.c tos definēju ar NULL vai 0 vērtībām. Tad main funkcijas sākumā tos inicializēju ar kaut kādām izmantojamām vērtībām.
Bet citādi grūtākais bija veikt atkļūdošanu, kad dažus no algoritmiem pirmo reizi sarakstīju asemblerā. Toties tas tāpat bija nedaudz interesanti, jo iemācījos izmantot GDB. Reizē smiekīga un bēdīga bija realizācija pēc kādas stundu ilgas atkļūdošanas, ka divi biti priekš “op” glabājas sākumā nevis beigās, jo taču tiek izmantots little endian.