Heightmap generator in software renderer v C-ju

Ta projekt predstavlja osnovni generator heightmapa znotraj software rendererja, napisanega v
C-ju. Gre za nekoliko nestandarden renderer, saj uporablja lastne metode in pristope, ki se razlikujejo od sodobnih grafičnih API-jev, Ravno zaradi tega ponuja enostavno in intuitivno prilagodljivost programa.

Vrednosti (resolucija, amplituda terena, velikost terena...) se spreminjajo v source-kodi, sicer pa so default opcije že podane in jih ni treba spreminjati. Kontrole so sledeče:

'ESC'-konča program;
'Q'-premakni navzdol po Y osi;
'E'-premakni navzgor po Y osi;
'F'- rotiraj po Y osi (+smer);
'G'- rotiraj po Y osi (-smer);
'W'-premakni naprej po Z osi;
'S'-premakni nazaj po Z osi;
'R'-generiraj novi teren).

V osnovi ne uporabljam matričnih operacij (kot je standard), saj v mojem primeru vzporednost ni potrebna in obenem metode niso abstrahirane od dejanskega dogajanja, kar je boljše za razumevanje rendererov. Tudi nekateri ostali algoritmi (npr za clippanje) so rahlo posebni. Zaradi tega sem se moral včasih sam znajti, ker splošnih rešitev ni bilo. Vso delovanje je opisano v komentarih kode ali pa je trivialno.

Limitacije so tukaj tudi pomembne, namreč začetna ideja je bila dodati logiko za barvo, svetlost itd. vendar windowsov API podpira le 16 barv v terminalu, zato to ni bilo mogoče.

Glavna dela programa:

Generacija heightmapa:

```
globalneTransformacije=(transformacije){{0, 0, 0}, 1.0f, {0, 0, 450}};
model* objekt=initModel((sirinaHeightmap-1)*(dolzinaHeightmap-1)*2);
generirajTeren();
razcleniHeightmap(objekt);
```

Glavni rendering loop:

```
preverjanjeInputa(objekt);
rasterizacija(objekt);
pocistiBuffer();
renderajBuffer();
```

Generacija terena*:

Najprej generiram naključne višine za vse točke v 'heightmapu', kar ustvari grobo in neskladen teren. Nato uporabim metodo drsnega okna (ang. sliding window; tukaj sem se zgledoval po CNN-ih), ki za vsako točko izračuna povprečje vseh njenih sosednjih vrednosti. Na ta način zgladim prehode med višinami in ustvarim bolj naraven, valovit teren.

Osnovni pregled rendering loopa*:

Vsak objekt se premika po »svetu«, zato uporabljam struct *globalneTransformacije*, ki hrani osnovne informacije o transformacijah glede na kamero (kot okoli posamezne osi; skalar velikosti; pozicija v »svetu«), vsak objekt pa ima spet svoje lokalne transformacije. Ker me pri tej nalogi samo zanimajo lokalne rotacije in globalni premiki, te transformacije tudi izvedem, nakar pošljem vsak posamezen trikotnik v funkcijo, ki jih *clippa* (torej poskrbi da je vsak trikotnik pred *nearClip* ravnjo ali pa jih odreže če gredo preko te ravni. Glede na pozicijo originalnega trikotnika (podrobjejši opis je v source kodi), nariše 0, 1 ali 2 trikotnika (če je ena točka za *nearClip*, pri preseku ravni nastaneta 2 trikotnika. Pomembno je tudi omeniti, da preden trikotnike sploh »clippa«, izračuna skalarni produkt normale trikotnika in krajiščnega vektorja trikotnika in izloči (preskoči) trikotnike, ki so obrnjeni vstran od kamere (angl. backface culling) ali pa so predaleč od kamere. To je zelo pomembna optimizacija, zaradi katere lahko renderam tudi zelo velike terene.

Link do videa

* podrbonejši opis delovanja je v source kodi

Viri:

Večinoma sem se zgledoval po svojih prejšnjih projektih, vendar sta mi dve spletni strani zelo pomagali (predvsem pri barvanju trikotnikov):
https://sl.wikipedia.org/wiki/Te%C5%BEi%C5%A1%C4%8Dni_koordinatni_sistem (pridobljeno 10. 6. 2025)
https://www.sunshine2k.de/coding/java/TriangleRasterization/TriangleRasterization.html (pridobljeno 8. 6. 2025)

Dokumentacija za operacije v Windowsovem consolu:
https://learn.microsoft.com/en-us/windows/console/writeconsoleoutput (pridobljeno 7. 5. 2025)

* podrbonejši opis delovanja je v source kodi