SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3118

Softverska implementacija grafičkog protočnog sustava

Đuro Kliček

Zagreb, lipanj 2013.

SADRŽAJ

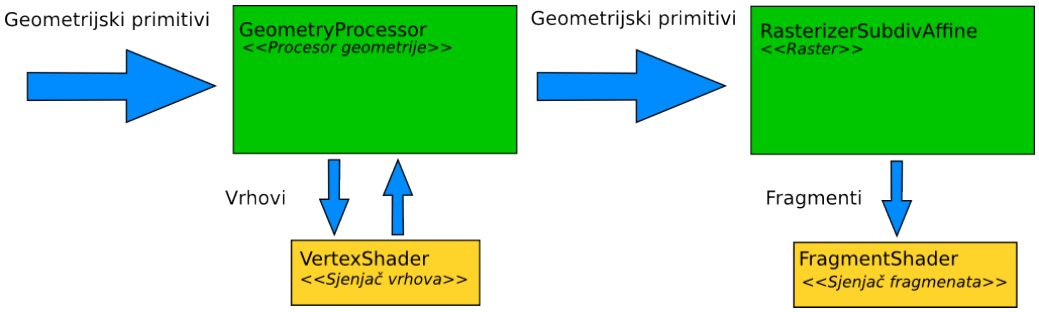
1. Uvod

U ovom radu analiziram postojeću implementaciju softverskog grafičkog protočnog sustava, te je izmijenjujem da može raditi u više dretva. Napravio sam dvije verzije izmjena, jednu sa mehanizmima sinkronizacije dretvi i jednu bez. Ovaj grafički protočni sustav dio je Visage softverskog paketa za praćenje lica. Zbog nemogučnosti rada grafičkog protočnog sustava u više dretvi, i sama detekcija lica je ograničena na jednu dretvu.

2. Grafički protočni sustav

2.1. Postojeći grafički protočni sustav

Postojeći softverski grafički protočni sustav napisao je Markus Trenkwalder. On se sastoji od procesora geometrije sa programom za sjenjčanje vrhova(engl. *Vertex Shader*) i rastera sa programom za sjenjčanje fragmenata(engl. *Fragment Shader*). Procesor geometrije uzima podatke o geometrijskim primitivima, predaje ih programu za sjenjčanje vrhova na obradu, zatim odbacuje primitive koje ne treba crtati te ih prosljeđuje rasteru. Raster mapira primitive na rasterski prikaz, odsijeca prikaz na veličinu rasterske jedinice i prosljeđuje te podatke, fragmente, programu za sjenjčanje fragmenata.

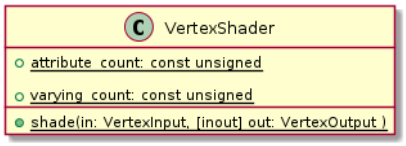


Slika 2.1: Protok podataka kroz sustav.

Za korištenje ovog protočnog sustava potrebno je implementirati programe za sjenjčanje vrhova i fragmenata. Konkretno te programe predstavlja dvoje razreda koji implementiraju određene statične metode i sadrže određene statične atribute. Dodatno program za sjenjčanje fragmenata mora naslijediti razred `GenericSpanDrawer` ili `SpanDrawer16BitColorAndDepth`.

2.1.1. Program za sjenjčanje vrhova

Opis atributa razreda koji predstavlja program za sjenjčanje vrhova:



Slika 2.2: Primjer razreda koji predstavlja program za sjenjčanje vrhova sa svim potrebnim metodama i članovima.

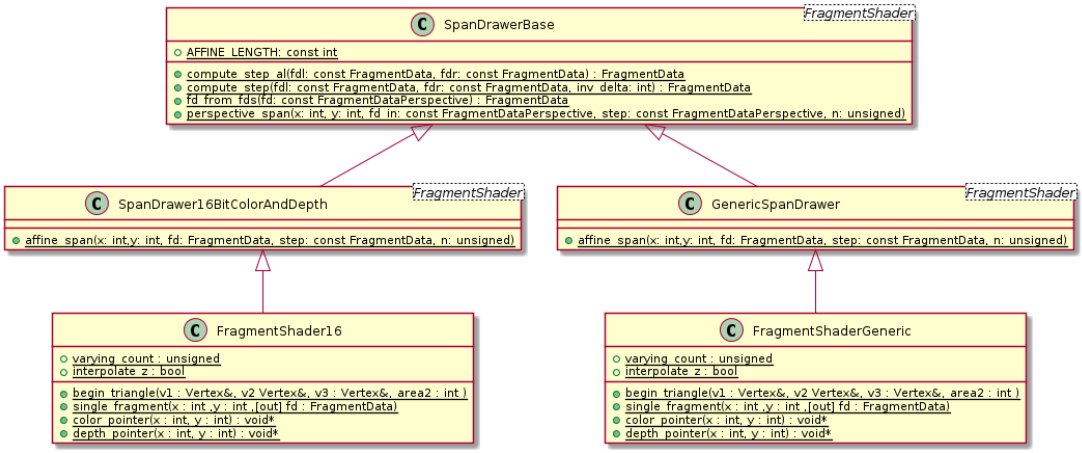
`varying_count`: u sebi sadrži broj interpoliranih varijabli koje program za sjenjčanje vrhova predaje programu za sjenjčanje fragmenata.

`attirbute_count`: u sebi sadrži broj nizova vrijednosti koji se mogu poslati programu za sjenjčanje vrhova.

Opis metoda razreda program za sjenjčanje vrhova:

`shade`: uzima podatke o jednom vrhu iz parametra `in` i vraća rezultate operacija nad tim vrhom preko izlaznog parametra `out`.

2.1.2. Program za sjenjčanje fragmenata



Slika 2.3: UML dijagram primjera implementacije programa za sjenjčanje fragmenata.

Opis atributa programa za sjenjčanje fragmenata:

`varying_count`: u sebi sadrži broj interpoliranih varijabla koje program za sjenjčanje fragmenata preuzima od programa za sjenjčanje vrhova.

`interpolate_z`: ako je sadržana vrijednost istinita, raster će interpolirati z-komponente fragmenata, što je potrebno za testiranje dubine(engl. *depth test*).

Opis metoda sjenjača fragmenata:

`triangle_begin`: sustav je poziva prije iscrtavanja trokuta.

`single_fragment`: njezin ulaz je jedan fragment i njegova pozicija na rasterskoj jedinici, nju sustav poziva za svaku poziciju u rasteru.

`color_pointer`: na ulaz dobiva koordinate pozicije u rasteru, te vraća pokazivač na piksel na tom mjestu.

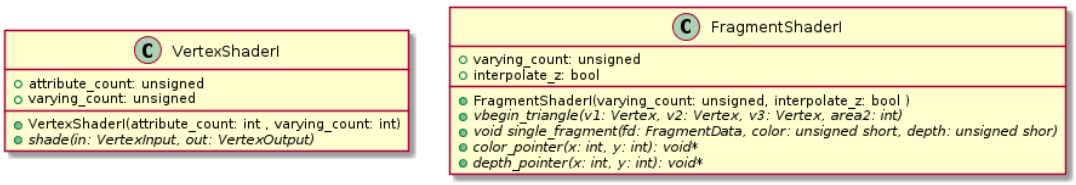
`depth_pointer`: na ulaz dobiva koordinate pozicije u rasteru, te vraća pokazivač na vrijednost z-spremnika na tom mjestu.

2.2. Izmjena grafičkog protočnog sustava

Postojeća implementacija ne može istovremeno pisati u više rasterskih jedinica, sa programom za sjenjčanje fragmenata iste funkcionalnosti. Pošto su sve metode i članovi programa za sjenjčanje `static`, programu za sjenjčanje se parametri mogu predati jedino preko statičnih varijabli ili varijabli u globalnom opsegu. Kod sekvencijalne izvedbe ovo nije problem jer se kod svakog iscrtavanja može promijeniti vrijednost tih varijabla i postaviti iscrtavanje na neku drugu jedinicu. Dok bi kod paralelne izvedbe ovakvo mijenjanje izlaza dovodilo do neželjenog međudjelovanja.

2.2.1. Rješenje s mehanizmima sinkronizacije dretvi

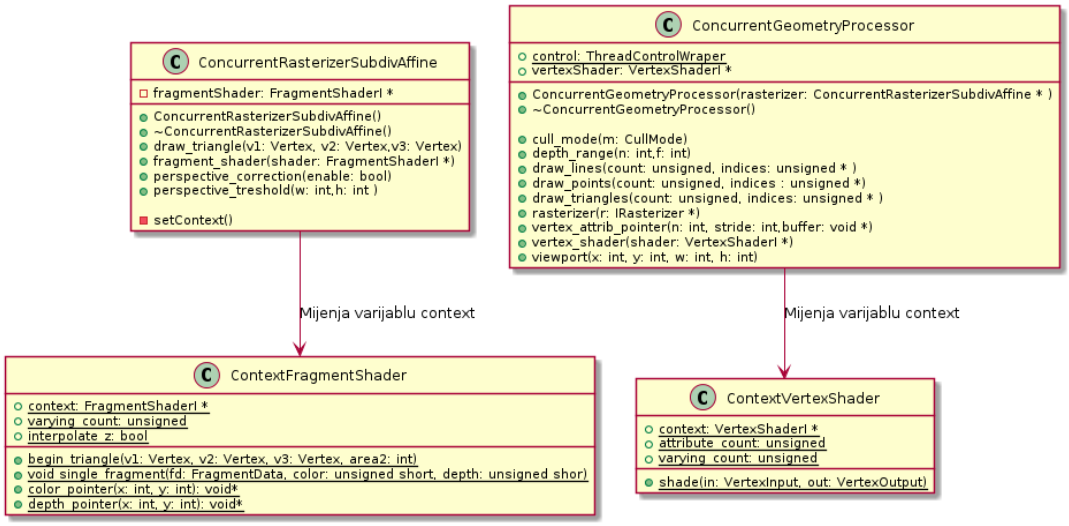
U ovom rješenju dodao sam sučelje na postojeće rješenje. Svaki poziv protočnom sustavu kroz sučelje je kritični odsječak, u kojem koristi program za sjenjčanje koji pripada trenutnoj dretvi.



Slika 2.4: Dodana sučelja koja predstavljaju programe za sjenjčanje.

Dodao sam sučelje za razrede koji predstavljaju programe za sjenčanje. Apstraktni razred `VertexShaderI` je sučelje za program za sjenčanje vrhova. Sastoji se od virtualne metode `shade` i atributa `attribute_count` i `varying_count`. Također ima konstruktor u koji se predaju vrijednosti tih atributa.

Apstraktni razred `FragmentShaderI` je sučelje na razred koji predstavlja program za sjenjčanje fragmenata. Sastoji se od virtualnih metoda `begin_triangle`, `single_fragment`, `color_pointer` i `depth_pointer`, te atributa `varying_count` i `interapolate_z`. Osim toga ima konstruktor kojem se predaju vrijednosti navedenih atributa.



Slika 2.5: Sučelja na razrede GeometryProcessor i RasterizerSubdivAffine.

Razred `ConcurrentRasterizerSubdivAffine` nasljeđuje razred postojećeg razreda `RasterizerSubdivAffine`. Implementira razred `ContextFragmentShader` koji predstavlja program za sjenčanje fragmenata i, osim svih statičnih metoda i atributa koje zahtjeva protočni sustav, sadrži i statični atribut s pokazivačem na implementaciju razreda `FragmentShaderI`. Pozivom bilo koje metode `ContextFragmentShader` poziva se istoimena metoda implementacije `FragmentShaderI` na koju pokazuje prethodno spomenuti pokazivač. Razred `ConcurrentRasterizerSubdivAffine` nadjačuje sve javne metode svog roditelja, te mijenja pokazivač na implementaciju `FragmentShaderI` kod svakog poziva tih metoda.

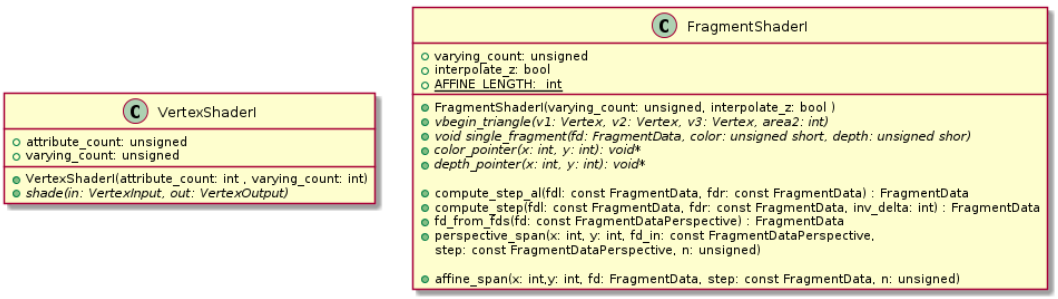
Razred `ConcurrentGeometryProcessor` nasljeđuje postojeći procesor geometrije `GeometryProcessor`. Ima implementaciju razreda `ContextVertexShader` koji predstavlja program za sjenčanje vrhova, osim statičkih metoda i atributa koje zahtjeva sustav, ima statički atribut s pokazivačem na implementaciju razreda `VertexShaderI`. Kao i prije, pozivom bilo koje metode `ConcurrentGeometryProcessor`, poziva se istoimena metoda implementacije razreda `VertexShaderI` na koju pokazuje spomenuti pokazivač. Razred `ConcurrentGeometryProcessor` nadjačuje sve javne metode postojećeg procesora geometrije. Kod poziva bilo koje

naslijeđene javne metode, ulazi u kritični odsječak, zatim mijenja pokazivač na svoju instancu `VertexShaderI`, poziva metodu svog roditelja, te izlazi iz kritičnog odsječka.

Da bi se koristio sustav s ovim promjenama potrebno je implementirati `VertexShaderI` i `FragmentShaderI`. Instancirati razrede `ConcurrentGeometryProcessor` i `ConcurrentRasterizerSubdivAffine`. Predati geometrijskom procesoru raster, instancirati implementacije programa za sjenjčanje, te ih predati rasteru i procesoru geometrije. Daljnje korištenje sustava je isto kao kod neizmijenjenog sustava.

2.2.2. Rješenje bez mehanizama sinkronizacije dretvi

U ovom rješenju ideja je promijeniti sam protočni sustav tako da ne koristi mehanizme sinkronizacije dretvi. Ovdje sam također napravio nove apstraktne razrede koji predstavljaju sučelja za programe sjenčanja.



Slika 2.6: Sučelja za programe za sjenjčanje.

Za program sjenjčanja vrhova napravio sam apstraktni razred koji služi kao sučelje, `VertexShaderI`. Izmijenio sam `GeometryProcessor` tako da radi sa instancom implementacije razreda `VertexShaderI`. Našao sam sve dijelove koda koji koriste statične metode programa za sjenjčanje vrhova i zamijenio ih pozivom metoda implementacije `VertexShaderI`.

Za program za sjenjčanje fragmenata, pošto mi nije trebala tolika generalizacija, izbacio sam razrede `GenericSpanDrawer`, te sam spojio razrede `SpanDrawer16BitColorAndDepth` i `SpanDrawerBase` u razred `FragmentShaderI`. Sve metode koje su bile statičke sam zamijenio sa normalnim metodama i sve pozive na statičke metode zamijenio sam pozivima metoda instance, te dodao virtualne metode `begin_triangle`, `single_fragment`, `color_pointer` i `depth_pointer`, te atribute `varying_count` i `interapolate_z`.

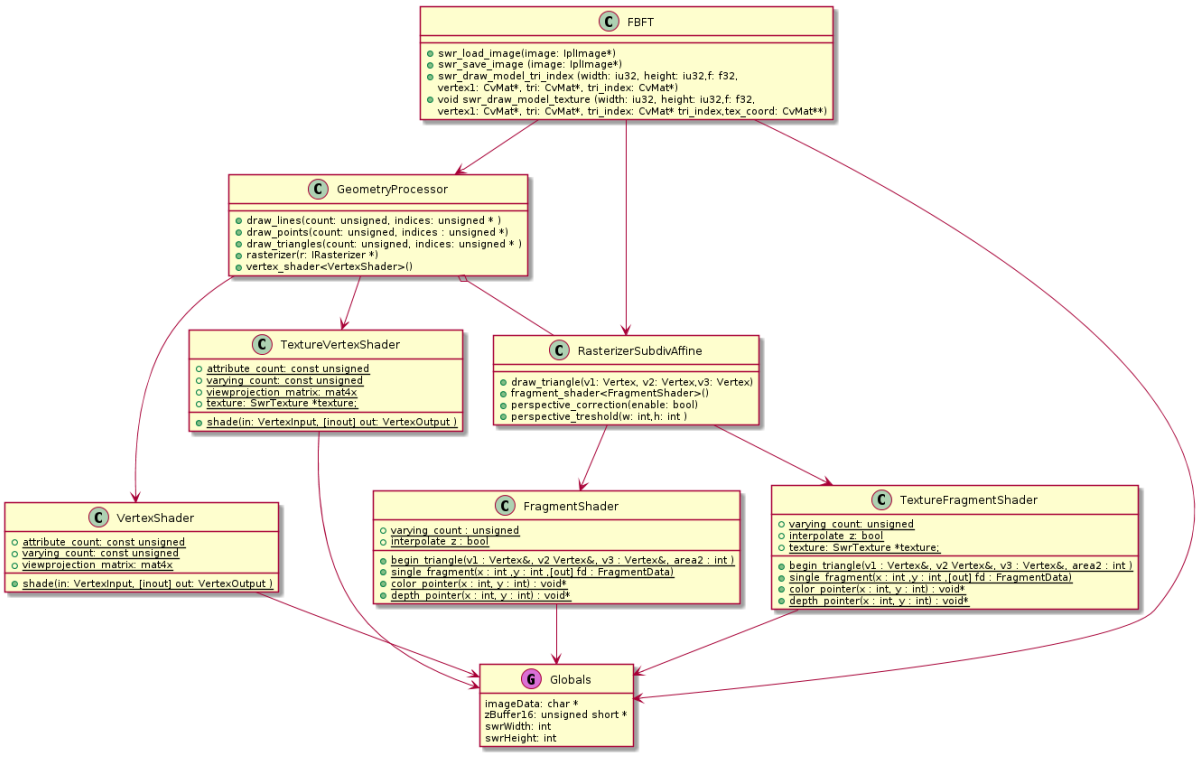
Da bi se ovako izmijenjen grafički protočni sustav koristio treba implementirati

apstraktne razrede koji su sučelja za programe za sjenjčanje. Potrebno je napraviti instance rastera i procesora geometrije. Ovdje je raster razred `RasterizerSubdivAffine`, a procesor geometrije razred `GeometryProcessor`. Njima se treba predati pokazivač na instance implementacije sučelja za programe za sjenjčanje.

3. Integracija grafičkog protočnog sustava u Visage paket

Visage paket je softverski paket koji omogućuje praćenje lica. Ovaj paket koristi softversku implementaciju grafičkog protočnog sustava za iscrtavanje lica. Potreba za grafičkim protočnim sustavom koji radi u više dretvi proizlazi iz problema kao što je praćenje više lica istovremeno.

3.1. Integracija postojećeg grafičkog protočnog sustava u Visage paketu

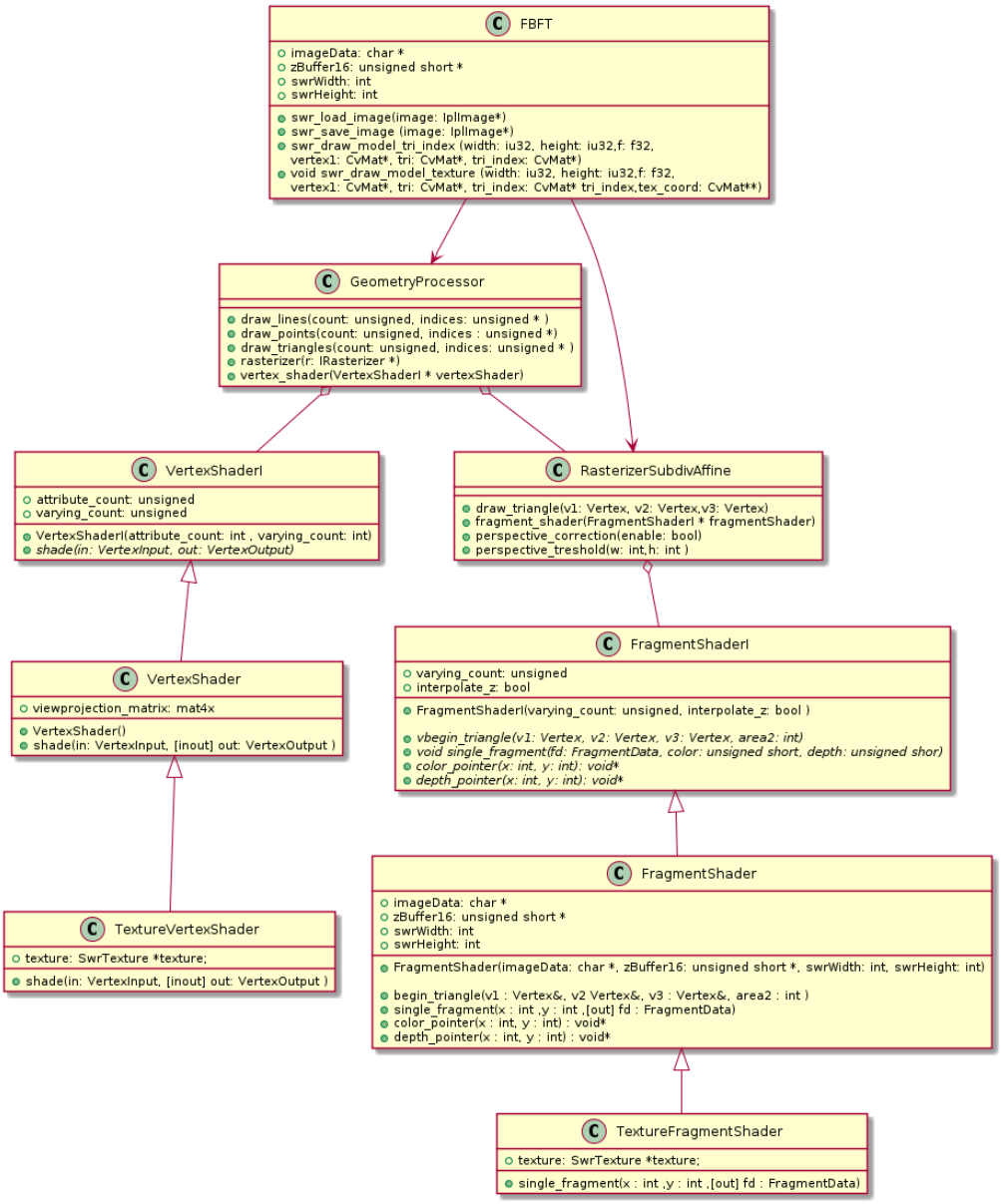


Slika 3.1: Dijagram početne integracije grafičkog protočnog sustava u Visage paketu.

Visage implementira dva para programa za sjenjčanje: `VertexShader` i `Fragment Shader`, te `TextureVertexShader` i `TextureFragmentShader`. Prvi par koristi za iscrtavanje bez tekstura, a drugi za iscrtavanje sa teksturama. Programi za sjenjčanje koriste određene globalne varijable. Oba programa za sjenjčanje fragmenata koriste pokazivač na izlazni spremnik `imageData`. Preko te varijable Visage predaje grafičkom protočnom sustavu početnu sliku i iz nje čita sliku dobivenu nakon iscrtavanja. Također, programi za sjenjčanje koriste globalnu varijablu `zBuffer16`, kao pokazivač na z-spremnik, te globalne varijable `swrHeight` i `swrWidth` širinu i visinu izlaznog spremnika. Osim potrebnih članova opisanih u sekciji 2.1, programi za sjenjčanje s teksturom imaju i statični atribut s pokazivačem na teksturu koju trenutno koriste.

Paket koristi grafički protočni sustav u metodama razreda `FBFT`. Metoda `swr_save_image` dobavlja sliku iz spremnika na koji pokazuje globalna varijabla `image Data`, `swr_load_image` kopira sliku u spremnik. `swr_draw_model_tri_index` iscrtava neteksturirane modele, a `swr_draw_model_texture` teksturirane modele.

3.2. Integracija izmijenjenog grafičkog protočnog sustava u Visage paketu



Slika 3.2: Dijagram nakon integracije izmijenjenog grafičkog protočnog sustava u Visage paketu.

Odlučio sam upotrijebiti promijenjeni grafički protočni sustav bez korištenja mehanizama sinkronizacije dretvi. Za njegovu upotrebu potrebno je implementirati `VertexShaderI` i `FragmentShaderI` umjesto razreda upisanih u sekciji 2.1. Konkretno promijenio sam `TextureFragmentShader`, `TextureVertexShader`, `VertexShader` i `FragmentShader` u razrede koji implementiraju prethodno navedene

razrede.

U razred `VertexShader` osim implementacije virtualnih metoda apstraktnog razreda `VertexShaderI` dodao sam i atribut `view_projection_matrix`, koja je bila statički atribut. Razred `TextureVertexShader` nasljeđuje razred `VertexShader`, jer osim ponašanja metode `shade` i dodatnog atributa `texture`, su ti razredi isti.

U razred `FragmentShader` osim implementacije virtualnih metoda apstraktnog razreda `FragmentShaderI`, dodao sam kao atribute `imageData`, `zBuffer16`, `swrHeight` i `swrWidth` koje koristi kao i istoimene globalne varijable. Razred `TextureFragmentShader` nasljeđuje `FragmentShader`, jer koriste iste atribute, nadjačao sam metodu `single_fragment` jer ima drugo ponašanje i dodao atribut `texture`.

U razred `FBFT` dodao sam iste atribute kao varijable koje su bile globalne. Taj razred brine da `imageData` i `zBuffer16` pokazuju na alociranu memoriju, te se njima koristi kao i s globalnim varijablama u nepromijenjenom Visage paketu.

4. Testiranje

Napravio sam tri testa, test međudjelovanja dretvi, test performansi samog grafičkog protočnog sustava i test performansi Visage paketa nakon integracije promijenjenog grafičkog protočnog sustava. U testove performansi je uključena usporedba performansi sa nepromijenjenim grafičkim protočnim sustavom i Visage paketom.

4.1. Testiranje međudjelovanja dretvi

Ovaj test ispituje dolazi li do međudjelovanja grafičkih protočnih sustava pokrenutih u više dretvi. Prvo se sto puta sekvencijalno iscrta model, svako iscrtavanje je iz drugog pogleda. Nakon toga u sto dretvi paralelno iscrtava model iz istih tih pogleda. Te se na kraju, slike modela iscrtanog iz istog pogleda paralelno i sekvencijalno uspoređuju piksel po piksel, ako nema razlike tada je test prošao. I rješenje s mehanizmima sinkronizacije dretvi, i rješenje s bez mehanizama sinkronizacije dretvi ovaj test je prošlo.

4.2. Testiranje performansi

Dva su testa performansi, jedan mjeri performanse neizmijenjenog grafičkog protočnog sustava i izmijenjenog grafičkog protočnog sustava bez mehanizama sinkronizacije dretvi. Dok drugi mjeri performanse Visage paketa s neizmijenjenim grafičkim protočnim sustavom i Visage paketa s izmijenjenim grafičkim protočnim sustavom bez mehanizama sinkronizacije dretvi.

4.2.1. Performanse izmjenjenog grafičkog sustava bez mehanizma sinkronzacije dretvi

Ovaj test iscrtava model sa 3287 vrhova i 5804 poligona tisuću puta i ispsuje prosječni broj iscrtavanja u sekundi. Ovaj test pokrenuo sam deset puta za izmijenjen i neizmijenjen grafički sustav, te sam dobio ove podatke:

Tablica 4.1: Mjerenje broja iscrtavanja u sekundi

Mjerenje	Rezultat neizmijenjenog sustava	Rezultat izmijenjenog sustava
1.	146.391	154.036
2.	147.623	154.943
3.	147.601	153.186
4.	147.950	151.149
5.	147.340	154.679
6.	147.384	153.941
7.	147.449	153.610
8.	147.319	154.012
9.	147.885	154.943
10.	147.471	154.703
Prosjek:	147.440	153.958

Iz podataka u tablici 4.1 može se izračunati da je izmjenjeni grafički protočni sustav u prosjeku 4% sporiji od neizmijenjenog grafičkog protočnog sustava.

4.2.2. Performanse Visage paketa s izmijenjenim grafičkim protočnim sustavom

U ovom testu mjerio sam vrijeme potrebno da Visage paket obradi video, bez da je s njim sinkroniziran. Video korišten u testu je "chia.avi" koji dolazi sa kodom Visage paketa. Ima rezoluciju 640x480, 15 okvira u sekundi i traje 27 sekundi. U sljedećoj tablici su rezultati mjerenja sa nepromijenjenim Visage paketom i Visage paketom s promijenjenim grafičkim protočnim sustavom izraženim u sekundama.

Tablica 4.2: Mjerenje brzine obrade videa Visage paketa

Mjerenje	Rezultat neizmijenjenog paketa	Rezultat izmijenjenog paketa
1.	23.5	23.2
2.	23.1	23.4
3.	23.4	23.6
4.	23.6	23.5
5.	23.8	23.6
6.	23.2	23.9
7.	23.5	23.0
8.	23.7	23.7
9.	23.6	23.5
10.	23.4	23.5
Prosjek:	23.48	23.5

Iz podataka se vidi da su razlike skoro neprimjetne.

5. Zaključak

Napravio sam dvije izmijenjene verzije softverskog grafičkog protočnog sustava. Izmijenjen sustav s mehanizmima sinkronizacije dretvi, sam odbacio jer nema smisla u ovo doba višejezgrenih procesora. Verzija bez mehanizma sinkronizacije dretvi malo je sporija od neizmijenjenog sustava no radi zadovoljavajuće dobro. Integracija u visage sustav tražila je minimalne izmjene, pošto je grafički sustav bio dobro odvojen od ostalih funkcionalnosti sustava. Brzina rada visage paketa nakon integracije izmjena na softverskom grafičkom protočnom sustavu je skoro nepromjenjena.

Softverska implementacija grafičkog protočnog sustava

Sažetak

U ovom završnom radu radi se o prilagođavanju softverskog grafičkog protočnog sustava da radi nesmetano u više dretvi. Podloga ovog rada je grafički protočni sustav koji je napisao Markus Trenkwalder. Softverski paket visage koristi taj grafički protočni sustav. Visage je softverski paket za praćenje lica. Zbog nemogućnosti ovog softverskog protočnog sustava da radi u više dretvi, i sam softverski paket visage je ograničen na praćenje lica u jednoj dretvi. Grafički protočni sustav izmijenjen je na dva načina koji mu omogućavaju da radi u više dretvi. Prva verzija sve pozive grafičkom protočnom sustavu tretira kao kritične odsječke. Druga verzija je protočni grafički sustav izmijenjen tako da radi u više dretvi bez potrebe za takvim mehanizmima.

Ključne riječi: dretva, višedretvenost, grafika

Software implementation of a graphics rendering pipeline.

Abstract

This thesis is about modifying a software implementation of a graphics rendering pipeline so that it can work in multiple threads. The foundation of this thesis is an implementation of a graphics rendering pipeline by Markus Trenkwalder. This graphics rendering pipeline is used by Visage, a facial features tracking software. Because of the inability of the uderlying implementation of the software graphics rendering pipeline to work in multiple threads, the Visage software is limited to tracking in one thread. The graphics rendering pipeline is modified in two different ways as to allow it to work in multiple threads. The first version of the graphics rendering pipleine is modified so all calls to it are a critical section of code. The second version of the graphics rendering pipeline is modified in a way that does not requre thread syncronisation mechanisms.

Keywords: thread, renderer, multithreading