SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3913

ANIMACIJA POKRETANA PRAĆENJEM LICA

Silvestar Badak

Zagreb, lipnja 2015.

Sadržaj

Uvod

U današnjem svijetu računalna grafika se brzo razvija. Razvijaju se novi sustavi i tehnologije kao software (računalni programi), a brzo se i sam hardware (mehanički dijelovi računala) mijenja. Primjerice grafičke kartice su povećale protočnu strukturu, procesori povećali broj jezgri i brzinu obavljanja računskih operacija, memorije su povećale brzinu prijenosa i kapacitet i tako dalje... Također s druge strane aplikacije danas obavljaju veliki rad u poslovanju poduzeća, i može se reći da bi rad bez njih bio nezamisliv. Osim toga aplikacije su prisutne i u svakodnevnom životu svakog čovjeka (komunikacija, igranje igara, planiranje obveza itd.)

Jedna od tehnologija koja se koristi u ovom radu se zasniva na pokretima lica korisnika, tj. računalo preko kamere prati položaj i pokrete lica, takozvani „face tracking“. Tako će ti pokreti biti unos podataka u aplikaciju, umjesto već dosadašnjih klasičnih načina poput tipkovnice i miša.

Za usporedbu, postoje i drugi načini za unos podataka u računalo prateći pokrete dijelova tjela, npr. nintendo wii, ali ključna razlika je u tome što wii zapravo prati položaj wii kontrolera kojeg se nosi u ruci, pa se time efektivno dobije praćenje pokreta ruke [1]. Tehnologija zasnivana na praćenju lica raspoznaje lice korisnika na slikama iz kamere bez ikakvih drugih pomagala.

U ovom radu je napravljena igrica u programskom jeziku C++, koristeći Microsoft DirectX tehnologiju crtanje i animaciju, a sama animacija je upravljana pomoću face tracking tehnologije dostupne u VisageSDK-u.

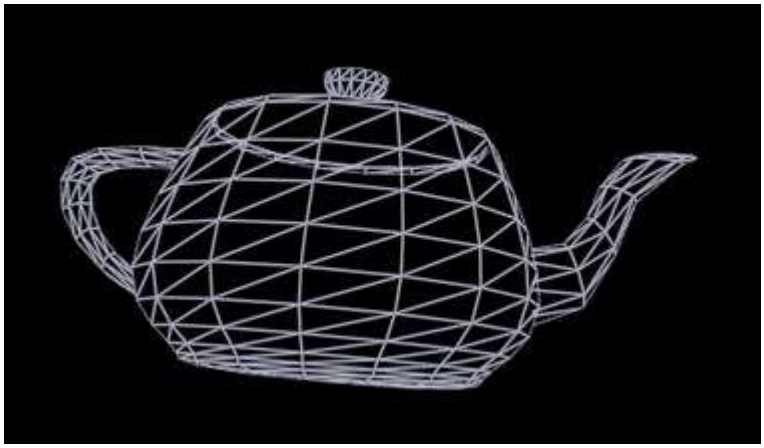
1. DirectX

DirectX je tehnologija koju razvija Microsoft i dostupna je samo uz računalni operacijski sustav Microsoft Windows. Svaka verzija DirectX-a izlazi s novom verzijom Windowsa. Trenutno je najnovija verzija DirectX 11.2 koja je dostupna uz operacijski sustav Windows 8.1. Aplikacija za ovaj rad koristi verziju 11.0 koja je objavljena uz Windows 7.Slika 1.1 prikazuje logo Windows 7 i Directx11.

Slika 1.1. Logo DirectX i Windows7

1.1 Prikaz objekata na ekran

DirectX omogućava iscrtavanje objekata na ekran. Ti objekti su predstavljeni kao skup točaka u 3D prostoru koji se povezuju u trokute, pa je zapravo cijeli objekt skup trokuta koji se projicira na 2D ravninu koja će predstavljati ekran (Slika 1.2.). Zatim se dobivene projekcije iscrtavaju i dobivamo sliku ili engleski naziv frame.



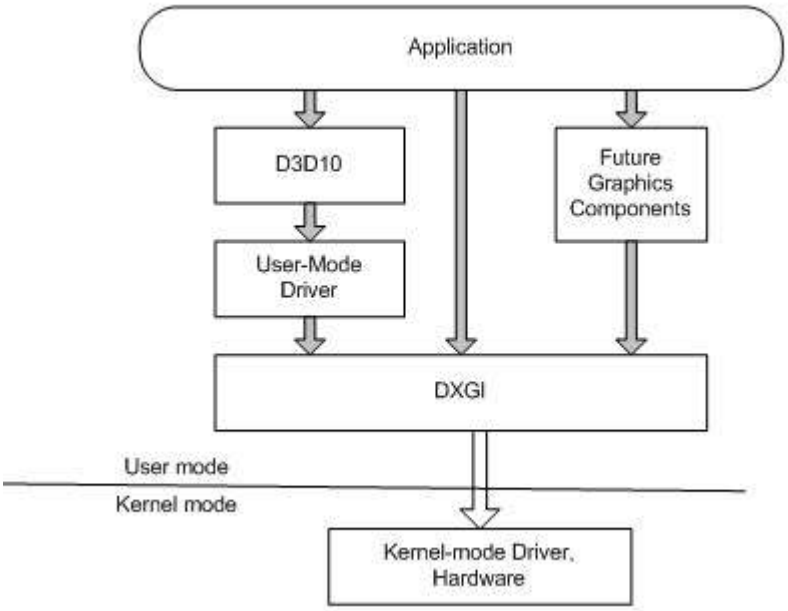
Slika 1.2. Čajnik (objekt) sastavljen od mnoštva trokuta

Objekti (tj. trokuti) se bojaju bojama koje se iščitaju iz slika. Te slike imaju i svoj poseban naziv, teksture, pa se cijeli proces takozvanog „lijepljenja“ teksture na trokute naziva teksturiranje. Kako bi se dobila prividna animacija objekata, za svaki mali dio vremena mijenjamo položaj objekta u 3D prostoru prije projekcije i iscrtavanja. Takvim

načinom se iscrtava više slika u jednoj sekundi, a mjerna jedinica za broj iscrtanih slika u sekundi je FPS(frame per second). Broj slika koji se može iscrtati na zaslon zavisi o računalu i njegovim komponentama (hardware), odnosno o procesoru i grafičkoj kartici. DirectX sadržava sučelje pomoću kojeg se upravlja rad sa grafičkom karticom.

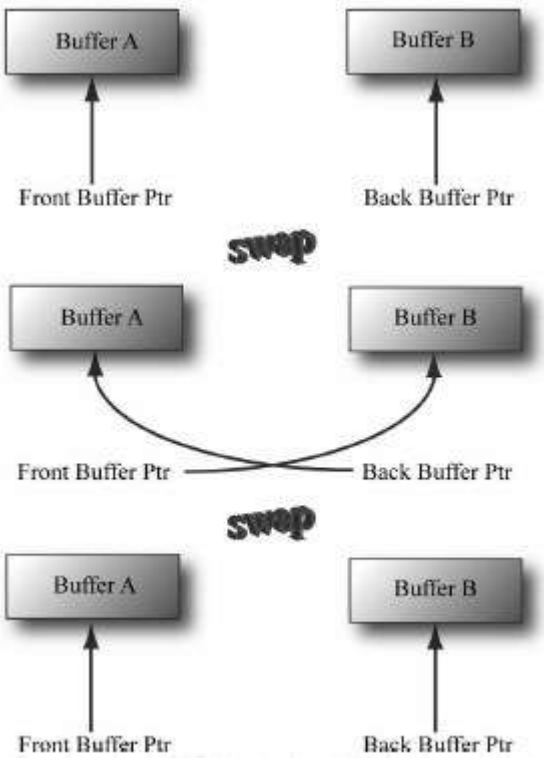
1.2 DirectX Graphics Infrastructure

DirectX Graphics Infrastructure ili kraće DXGI je biblioteka kojim se postižu određene funkcionalnosti na grafičkoj kartici. Rad DXGI biblioteke se odvija u korisničkom načinu rada i neovisan je o grafičkoj kartici koja se nalazi u računalu (Slika 3.). Jedna od funkcionalnosti koju DXGI obavlja je izmjena spremnika za iscrtavanje. [2]



Slika 1.3. Dijagram komponenti aplikacije i položaj djelovanja DXGI biblioteke. Slika preuzeta iz [2]

Slike koje se iscrtavaju na ekran se prvo iscrtavaju u jednu teksturu koja se zove stražnji spremnik ili back buffer. Slika koja se trenutno prikazuje na ekranu se nalazi u prednjem spremniku tj. front bufferu. Kada se iscrta slika u stražnji spremnik, spremnici mjenjaju uloge. Dakle, stražnji spremnik postaje prednji, a prednji postaje stražnji, i tako naizmjence (Slika 1.4.).



Slika 1.4. Izmjena spremnika. Slika preuzeta iz [3]

Struktura podataka za upravljanje radom prednjeg i stražnjeg spremnika je C++ razred IDXGISwapChain. Njega inicijaliziramo pomoću metode D3D11CreateSwapChain(). Za poziv te metode potreban je parametar koji je struktura DXGI_SWAP_CHAIN_DESC. U njoj specificiramo svojstva našeg prednjeg i stražnjeg spremnika. [3]

1.3 ID3D11Device i ID3D11DeviceContext

Osim upravljanja spremnicima na grafičkoj kartici, zadaća DirectX-a je upravljanje stanjima na grafičkoj kartici. Ta stanja određuju kojim se načinom iscrtavaju trokuti na ekran. Za to se brinu Direct3D Device i Device Context. Pored toga brinu se i o podacima koji se šalju na grafičku karticu, primjerice točke koje zelimo povezati u trokute, podatke za osvjetljenje objekta, matrice za animaciju objekta i slično. Podatkovne strukture koje šalju podatke prema grafičkoj kartici se dijele u tri grupe: spremnik vrhova (vertex buffer), spremnik indeksa (index buffer) i spremnik konstanti (constant buffer).

ID3D11Device služi za stvaranje podatkovnih struktura koje će prenositi navedene podatke i za stvaranje stanja koji upravljaju načinom crtanja na grafičkoj kartici. Slično kao i kod DXGI sučelja postoje strukture podataka koje opisuju svojstva nekog stanja kojeg želimo napraviti. Za izradu nekog stanja preduvjet su popunjene odgovarajuće strukture. [3]

Za razliku od ID3D11Device koji stvara objekte, ID3D11DeviceContext postavlja stvorena stanja u upotrebu. Dakle, ako se instancira više različitih oblika za neko stanje, sa ID3D11DeviceContext postavlja se to stanje u upotrebu. Analogno tome, ako su točke koje će se poslati prema grafičkoj kartici grupirane u više spremnika vrhova, odabirom spremnika odabire se ujedno i točke koje se šalju na grafičku.

Treba imati na umu da je mjenjanje stanja ili nekog spremnika za slanje podataka „skupa“ operacija pa je poželjno izbjegavati ih. To se može napraviti tako da se prilikom pisanja koda pazi kojim redoslijedom iscrtavamo objekte, odnosno da se grupiraju objekti koji koriste ista stanja za njihovo iscrtavanje. Tek kada se iscrta jedna grupa objekata, promijene se potrebna stanja i onda se prelazi na crtanje druge grupe objekata koja imaju ista stanja.

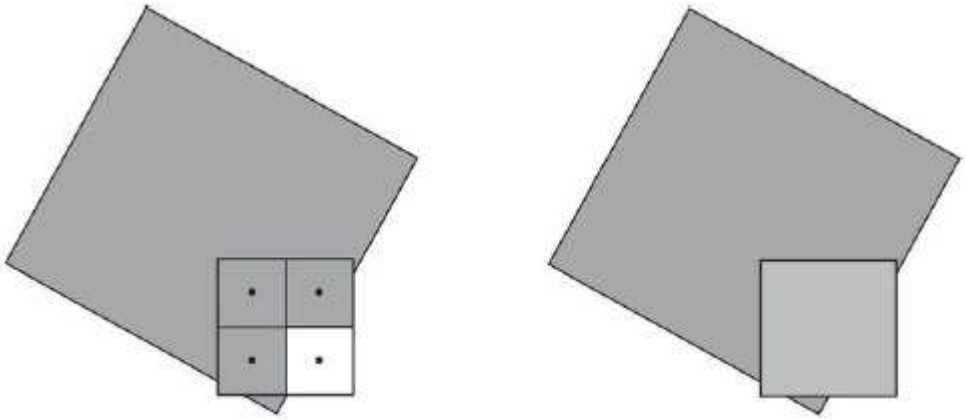
1.4 Načini crtanja objekata

Već spomenuta stanja (kojima upravljaju Device i DeviceContext) određuju način crtanja objekata. Biti će opisana samo neka od njih.

Za početak pri crtanju objekata često se (ali ne i nužno) koristi uklanjanje nazubljenih linija ili multisample anti-aliasing, kraće multisampling. Pošto je ekran na kojem se crtaju objekti diskretizirana ravnina (dijelovi ravnine su kvadratići zvani pikseli), prilikom crtanja trokuta nailazi se na probleme pri crtanju njihovih bridova (Slika 1.5.).

Slika 1.5. Nazubljena linija i ispravljena nazubljena linija multisamplingom, Slika preuzeta iz [3]

Za uklanjanje nazubljene linije multisamplingom, koristi se dijeljenje svakog piksela u N proizvoljnih manjih djelova. Zatim se kao rezultatna boja nekog piksela koristi aritmetička sredina boje njegovih djelova (slika 1.6.). Vidljivo je kako uklanjanje nazubljenih površina je zapravo zahtjevna metoda pa se na računalima s manjom procesorskom snagom izbjegavaju. [3]

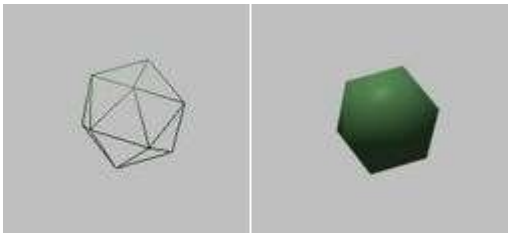


Slika 1.6. Ilustracija bojanja jednog piksela uz 4X Multisample Anti-Aliasing. Slika preuzeta iz [3].

Stanje za ovaj način crtanja se postavlja još pri stvaranju IDXGISwapChaina, ali potrebno je napomenuti da se njime dodatno upravlja i kod postavljanja stanja rasterizatora (komponente grafičke kartice koja izravno crta po stražnjem spremniku trokute na osnovu zadanih točaka). Prilikom mijenjanja stanja rasterizatora može se uključiti ili isključiti već ranije postavljeni način multisampliranja.

Sam rasterizator je stanje koje također određuje i druge stvari poput hoće li biti uključena provjera dubine ili ne (više o ovome kasnije), način popunjavanja trokuta i odsijecanje stražnjih trokuta. Za stvaranje rasterizatora koristi se funkcija (koja se nalazi u klasi ID3D11Device) CreateRasterizerState(), a za postavljanje (nalazi se u klasi ID3D11DeviceContext) RSSetState(). [2]

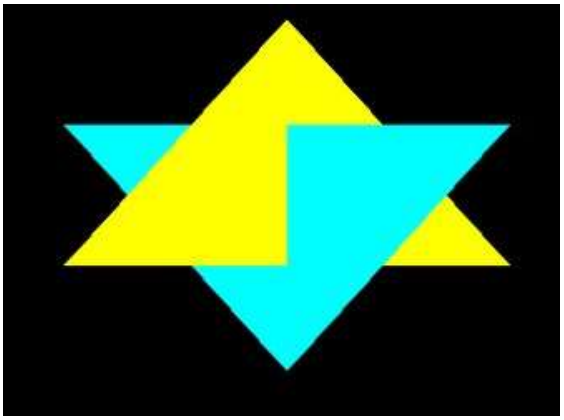
Postoje dva načina popunjavanja trokuta: žična forma (wireframe) i puni trokut (solid) prikazano na slici 1.7. Obično se koristi puni trokut, a žična forma samo kod debuggiranja aplikacije.



Slika 1.7. Žični i puni prikaz trokuta

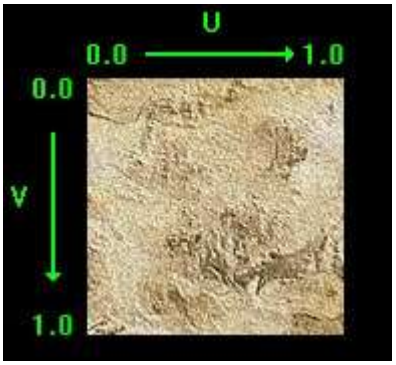
Odsijecanje stražnjih trokuta je metoda pri kojoj se ne crtaju stražnji trokuti, jer ionako neće biti vidljivi na ekranu (objekt vidimo samo s jedne strane).

Provjera dubine je metoda kojom se provjerava dubina nacrtanog piksela, tj. prilikom projekcije trokuta na ekran zapisujemo i dubinu svakog piksela, odnosno udaljenost točke na trokutu od ravnine projekcije. Pa tako ako postoji više točaka iz dva različita trokuta koje konkuriraju za neki piksel, za boju piksela se uzima onaj koji je bliži ravnini projekcije (slika 1.8.).



Slika 1.8. Dva trokuta koja se sijeku, na mjestima gdje se preklapaju trokuti urađen je depth testing

Lijepljenje tekstura je metoda kod koje se pikselima koje trokut zauzima određuje boja učitavanjem boje iz teksture. Projiciranim vrhovima trokuta se odrede vrijednosti u UV koordinatnom sustavu (slika 1.9), a pikseli između vrhova se zatim interpoliraju ovisno o udaljenosti piksela od vrha trokuta.



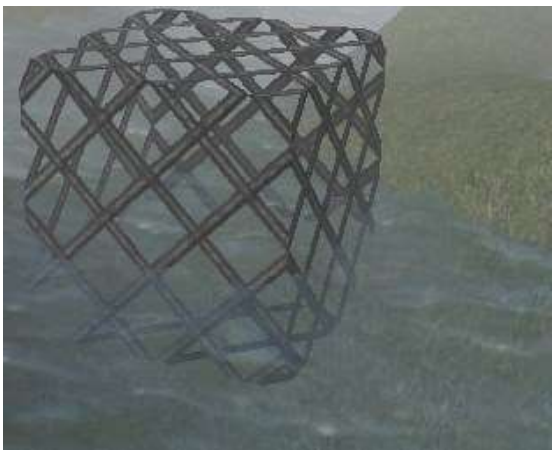
Slika 1.9 Lijepljenje teksture u UV koordinatnom sustavu. Slika preuzeta sa [4].

Interpolacija koordinata za veće od 1 i manje od 0 ovisi o stanju uzorkivača (eng. sampler). Stanje samplera postavlja se sa funkcijom PSSetSamplerState(), a stvaranje samplera se obavlja sa funkcijom CreateSamplerState(). Postoji više načina ponašanja za vrijednosti, a neke od njih su zaokruživanje i zamatanje prikazani na slici 1.10.

Slika 1.10. Lijevo je zaokruživanje (clamp) i zamatanje (wrap)

Zaokruživanjem se sve vrijednosti iznad 1 zaokružuju na jedan, a sve vrijednosti ispod 0 se zaokružuju na nulu. Kod zamatanja se vrijednosti veće od 1 i manje od 0 normaliziraju na raspon od 0 do 1 (npr. 1.3 i 2.3 se preslikavaju u 0.3, a -0.5 u 0.5). [3] [5]

Miješanje boja ili blending je način crtanja prozirnih objekata ili magle, dima i slično. Mješanje boja se ostvaruje se interpolacijom najbližeg piksela i piksela iza njega. Najčešće se uzima da alpha komponenta najbližeg piksela određuje koliko se vidi piksel iza njega (Slika 1.11). Miješanje boja ovisi stanju kojem se nalazi sklop za mješanje boja (eng. blend state). Blend state kreiramo sa funkcijom CreateBlendState(), a postavljamo sa OMSetBlendState().



Slika 1.11. Rešetkasti objekt (kocka) s prozirnim dijelovima i prozirna voda. Slika preuzeta sa [3].

2. Animacija objekata

Objekt se animira tako što za svaki mali dio vremena koji protekne, obave se transformacije nad objektom. Animacija objekata se postiže matričnim transformacijama objekata.

2.1 Matrične transformacije

Transformacije se nazivaju matrične jer točke koje se transformiraju predstavljaju se kao vektor i množe se s matricama. Točke moraju imati četvrtu koordinatu postavljenu na jedan zbog rada sa 4x4 matricama, tj. transformacije se obavljaju u homogenom koordinatnom sustavu s homogenom koordinatom jedan. Pošto vrijeme između dva crtanja objekta na ekran (2 framea) može varirati, definira se brzina objekta koja se množi sa vremenom koje proteklo između ta dva crtanja, i tako se dobiva željeni pomak objekta. Time se dobiva konstantna brzina animacije i pritom ne ovisi o računalu na kojem se izvršava i njegovoj snazi. Više riječi o upravljanju vremena između dvije scene će biti kasnije. Postoje tri načina transformacije objekta: skaliranje, rotacija i translacija.

Skaliranje je operacija kojom se mjenja veličina objekta. Matrica skaliranja je prikazana na slici 2.1.

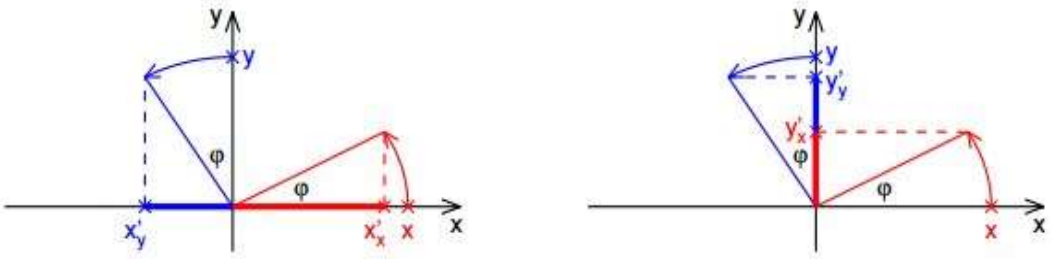
$$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Slika 2.1. Matrica skaliranja

Skaliranje se provodi tako što se množi x, y ili z koordinata objekta s odgovarajućim skalirajućim faktorom. Pa tako za vrijednosti faktora veće od jedan objekt se povećava, za vrijednosti manje od jedan i veće od nule se smanjuje, a za vrijednosti manje od nule dobivamo zrcaljenje. Kod animacije objekata transformacija skaliranja nije toliko se rijeđe koristi s obzirom na presotale dvije.

Slijedeća transformacija je rotacija objekta. Objekti se rotiraju za neki kut koji se zadaje u radijanima. Ako se rotiraobjekt, odnosno točka oko neke koordinatne osi, vrijednost koordinate za tu os ostaje ista, a vrijednost za preostale dvije koordinate se

mijenja (Slika 2.2.).



Slika 2.2. Rotacijom neke točke oko z osi za neki kut, mijenjaju se x i y komponente. Slika preuzeta sa [5].

Iznos za koliko se promjeni neka komponenta prilikom rotacije oko neke koordinatne osi ovisi o sinusu i kosinusu kuta za kojeg rotiramo. Pa će tako primjerice vrijednost za x os i rotaciju oko z osi (slika 2.3.) promjeniti se tako da zbrojimo x komponentu točke pomnoženu sa kosinusom kuta i y komponentu pomnožimo sa kosinusom tog kuta. Matrice rotacije za svaku od koordinatnih osi prikazane su na slici 2.3.

X-axis

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Y-axis

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

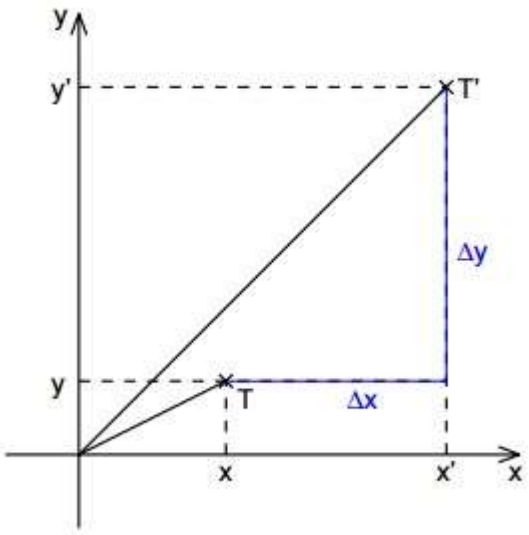
Z-axis

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Slika 2.3. Matrice rotacije za svaku od osi

Zbog jednostavnosti računanja za računalo, ove tri matrice se množe i dobiva se jedna rezultantna matrica rotacije koja se popunjava s parametrima, kutevima, koji predstavljaju koliko želimo rotirati oko koje osi. [3] [5]

Zadnja transformacija objekta koja se koristi pri animiranju objekta je translacija. Kao i kod prethodnih transformacija, translacija se obavlja nad svakom točkom objekta (Slika 2.4.).



Slika 2.4. Translacija točke u 2D prostoru (analogno ovome može se dobiti translacija i za 3D prostor, samo treba dodati z komponentu). Slika preuzeta sa [5].

Translacija se obavlja tako da jednostavno pribrojimo svakoj komponenti točke njihov odgovarajući pomak. Pri translaciji dolazi na snagu četvrta homogena koordinata točke koja iznosi jedan. Matrica translacije je prikazana na slici 2.5. [3] [5]

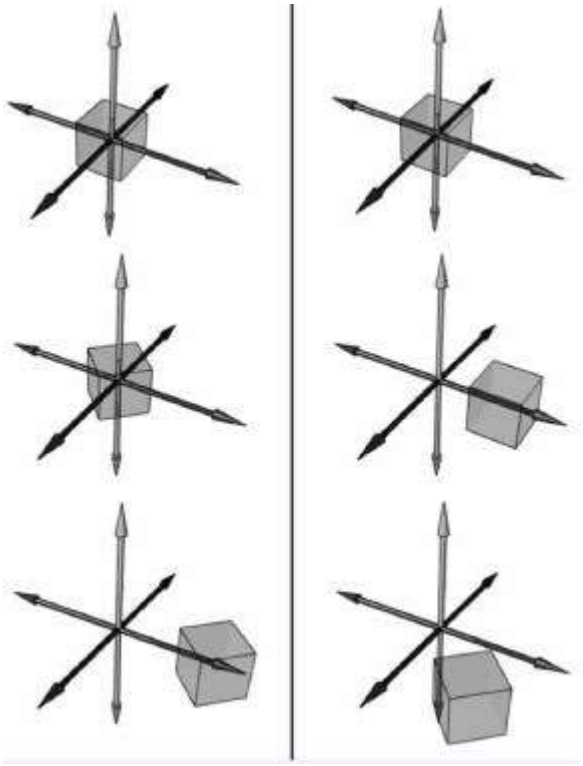
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \Delta_x & \Delta_y & \Delta_z & 1 \end{bmatrix}$$

Slika 2.5. Matrica translacije

Rezultat se dobiva jednostvnim umnoškom homogene koordinate (jedinice) i pomaka koji se zatim pribraja originalnoj točki i tako se puni rezultirajući vektor koji predstavlja translatiranu točku.

Bitno je imati na umu da kada učitamo točke objekta iz datoteke, objekt se nalazi u središtu koordinatnog sustava. Dakle čuvamo poziciju, kuteve rotacije i veličinu skaliranja u memoriji, pa onda za svako iscrtavanje slike obavljamo transformacije.

Da bi se ostvarile navedene transformacije množi se svaka točka redom s svakom matricom ili može se ići na kraći (brži) način, prvo pomnožiti te tri matrice (napraviti kompoziciju transformacija) pa onda točku pomnožiti s rezultatom. Prilikom množenja matrica mora se paziti na redoslijed množenja matrica, jer množenje matrica nije komutativno! Primjer različitih redoslijeda množenja matrica je na slici 2.6.



Slika 2.6. Kompozicija transformacija, lijevo je prikazan redoslijed rotacija pa translacija, a desno translacija pa rotacija. Slika preuzeta iz [3].

Kao što se i može pretpostaviti različiti redoslijedi transformacija neće dati isti rezultat. [3]

2.2 DirectX math podrška

Microsoft nudi DirectXMath C++ zaglavlje za olakšani rad s matricama. Ono omogućava da se ne mora „ručno“ pisati i popunjavati matrice ili pisati funkcije za množenje, invertiranje, transponiranje i slično s matricama.

Također važno je napomenuti da DirectXMath koristi posebne SIMD (single instruction multiple data) registre za operacije nad matricama kako bi se dobilo na ubrzanju rada programa. Postupak obavljanja operacija s registrima se odvija u 3 koraka: spremanje podataka u registre, obavljanje niza potrebnih operacija i iščitavanje rezultata. [3]

Podatke (točke i matrice) najprije stvaramo u posebnim tipovima varijabli. Na primjer za točku XMFLOAT3 ili XMFLOAT4, a za matrice XMFLOAT4X4 (postoje i dvodimenzionalne verzije ovih varijabli). Iz njih se može samo čitati i pisati, pa u prvom koraku se piše u njih odgovarajući. Da bi se spremili podatke u posebne registre za računanje, koriste se funkcije XMLoadFloat(). Kao povratnu vrijednost te funkcije dobiva se varijabla koja predstavlja spremljeni podatak u registar tipa XMVECTOR ili XMMMATRIX.

Zatim slijedi niz operacija s XMVECTOR i XMMATRIX varijablama. Primjerice XMMatrixTranspose() za transponiranje, operator „*“ za množenje dvaju matrica, XMMatrixInverse() za invertiranje itd.

Na kraju slijedi vraćanje podataka u XMFLOAT4X4 ili XMFLOAT3 varijable. To se obavlja pomoću funkcija XMStoreFloat(). Nakon spremanja podaci su spremni za slanje na grafičku karticu.

2.3 Vremenski sinkronizirana animacija

Već je spomenuto da vrijeme između crtanja dva objekta nije konstantno. Ono zavisi od računala do računala ili preciznije koliko je procesor „zauzet“ u nekom trentuku. Dakle potrebno je precizno izmjeriti koliko je vremena prošlo od zadnjeg ažuriranja varijabli koje bilježe poziciju i rotaciju nekog objekta.

Operacijski sustav Windows nudi C++ zaglavlje koje omogućuje da se dohvati podatke direktno iz sata računala. Prvi podatak je da se direktno očita koliko je otkucaja napravio sat na računalu, a drugi koliko otkucaja napravi sat u jednoj sekundi (frekvenciju rada). Recipročna vrijednost govori koliko jedan otkucaj predstavlja sekundi. Funkcija koja dohvati broj otkucaja je QueryPerformanceCounter(), a funkcija koja dohvaća frekvenciju rada je QueryPerformanceFrequency(). Funkciju za dohvat frekvencije rada poziva se pri inicijalizaciji aplikacije i dobiva se decimalni broj, frekvencija rada sata, koji će se koristiti pri svakom pozivu očitanja vremena. Da bi se u nekom trenutku očitalo koliko je prošlo vremena od zadnjeg očitanja, potrebno je sačuvati zadnji očitani broj otkucaja (slika 2.7.).

Slika 2.7. Očitavanje otkucaja sata

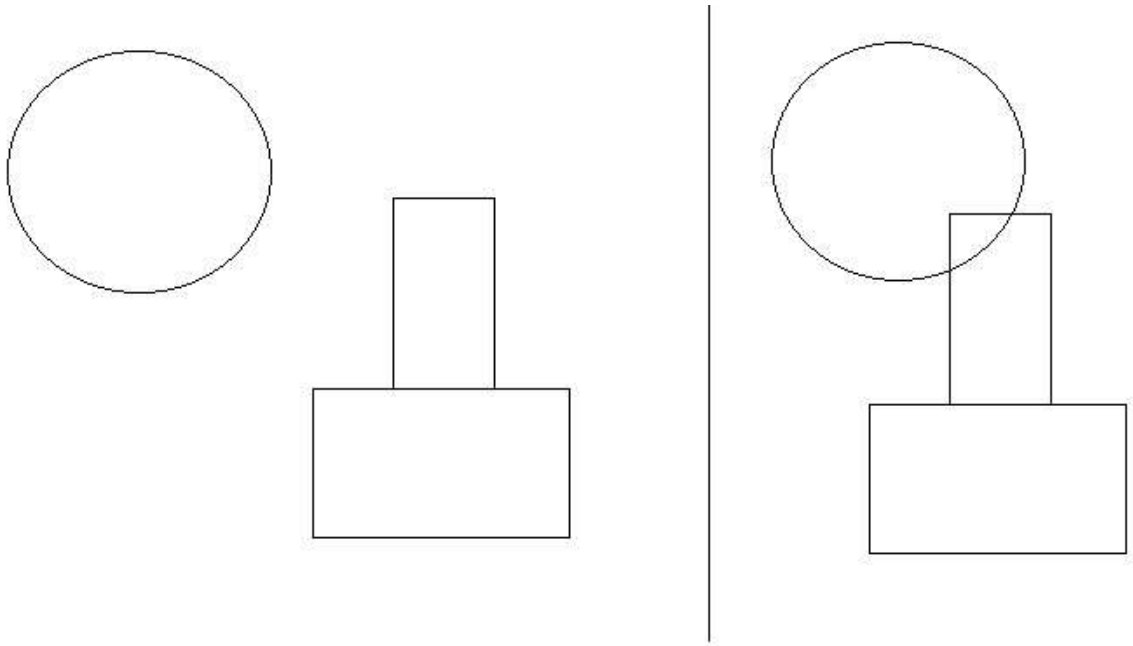
Nakon što se izmjeri trenutni broj otkucaja, oduzima se broj otkucaja od zadnjeg očitanja i dobivamo razliku, tj. deltu vremena koji je protekao od zadnjeg očitanja. Sve što je sada potrebno napraviti je pomnožiti taj delta vremena s recipročnom vrijednošću frekvencije rada i dobiti će se vrijeme koje je prošlo između dva crtanja.

2.4 Detekcija kolizija

Prilikom gibanja objekata u prostoru objekti se sudaraju. Za detekciju kolizija između objekata koriste se matematički principi. U priloženoj aplikaciji svemirski brod se

giba u svemiru i prema njemu dolaze meteori. Potrebno je detektirati da li je se brod sudario sa nekim meteorom te ako jest, završiti igru i zaključiti da je korisnik gubitnik.

Pošto je uobičajeno da su objekti kompleksne građe, računski bi bilo previše zahtjevno ići računati gledati da li se svaki trokut nekog objekta siječe s trokutom nekog drugog objekta. Zbog toga se pribjegava aproksimacijama. Svemirski brod u aplikaciji je aproksimiran s dva pravokutnika, a meteori kao kružnice (prikazano na slici 17.).



Slika 2.8. Detekcija sudara, lijevo je prikazan apkroksimirani svemirski brod u mimoilaženju s aproksimiranim meteorom, a desno u sudaru

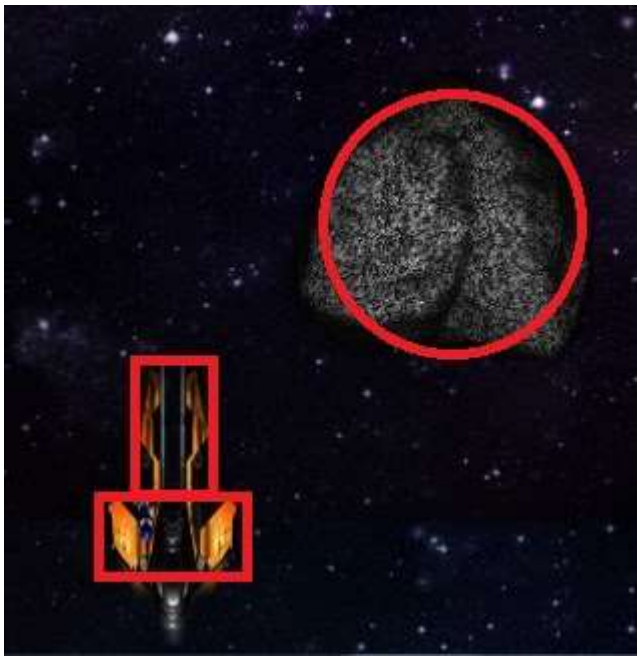
Da bi se odredilo jesu li objekti u sudaru ili ne, dobije se jednostavnom provjerom je li bilo koji vrh unutar središta kružnice. Formula izgleda ovako:

$$(Xp - Xs)^2 + (Yp - Ys)^2 < R^2$$

gdje Xp i Yp predstavljaju vrh pravokutnika, Xs i Ys središte kružnice, a R radijus kružnice. Ukoliko je ovaj uvjet zadovoljen promatrani objekti su u koliziji, a to u ovoj aplikaciji predstavlja kraj igre.

Usporedimo li sa izgledom objekata u igrici i performansama igre, vidi se da je aproksimacija dovoljno dobra (slika 2.9 i 2.10)

Slika 2.9. Izgled u igrici objekata



Slika 2.10 Objekti s označenim geomtrijskim tijelima korišteni za detekciju kolizije

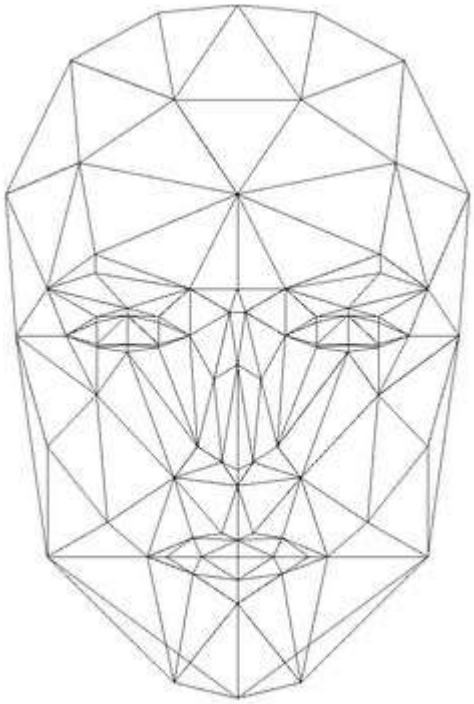
3. Tehnologija zasnivana na praćenju lica

Tehnologija zasnivana na praćenju lica (eng. face tracking) je u snažnom razvoju nad kojom se provode iscrpna istraživanja. Ta tehnologija ima široku primjenu, a neke od njih identifikacija osoba, ulazni podaci pomoću kojih se upravlja aplikacija, stvaranje virtualne proširene stvarnosti i slično. U početku face tracking je bio limitiran lošom računalnom opremom. Kamere su bile slabe ralučivosti i kvaliteta slike je bila loša, računalima je trebalo više vremena da izvedu algoritam itd. Danas se računalna oprema poboljšala pa su i rezultati koji se postižu dobili uzlet. Dostupne su mnoge biblioteke koje omogućuju praćenje lica, a jedna od njih je Visage software development kit i ona se koristi u ovom radu.

Visage software development kit ili kraće VisageSDK je programska podrška koju razvija tvrtka Visage Technologies. Ova programska podrška omogućuje praćenje lica u realnom vremenu koja je bila potrebna aplikaciji ovog rada.

3.1 Maska za praćenje lica

Za praćenje lica se koristi parametrizirana candide maska razvijena za kodiranje ljudskih lica (slika 3.1). U ovom radu se koristi najnovija verzija candide3 maska.



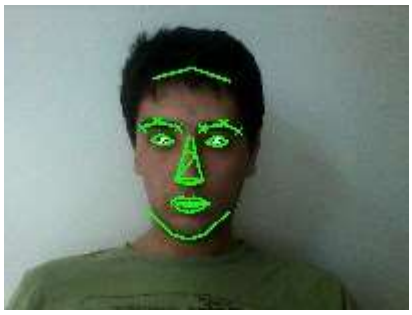
Slika 3.1 Candide maska. Slika preuzeta sa [6].

. Candide maska sadrži relativno malen broj poligona pa omogućava brzi oporavak od nestanka lica s kamere. Candide je upravljan s oblikovnim i akcijskim jedinicama.

Oblikovne jedinice nastoje model približiti obliku lica kojeg se prati, a akcijske jedinice prate promjene na licu. Akcijske jedinice se dijele na globalne i lokalne. Globalne prate rotaciju, a lokalne geste promatranog lica.

3.2 VisageTracker2

U VisageTracker2.h zaglavlju nalazi se razred VisageTracker2. Pri instanciranju razreda kao parametar proslijeđuje se datoteka u kojoj su sadržane konfiguracije za inicijalizaciju face trackinga. U konfiguracijskoj datoteci se nalaze primjerice koji candide model se koristi, koje se akcijske jedinice koriste, postavke za rad s kamerom itd. Podatci o licu kojeg se prati u razredu VisageTracker2 mogu se dobiti iz funkcijom getTrackingData(). [7] Na slici 3.2 je prikaz očitanih podataka iz visage trackera.



Slika 3.2 Prikaz očitanog lica na korsniku

Funkcija getFaceTrackingData() popunjava podatke u objektu tipa TrackingData. U TrackingData razredu mogu se naći podatci za rotaciju, translaciju, akcijske jedinice i slično. Popis akcijskih jedinica koji se mogu koristiti dan je u tablici 3.1. [7]

Akcijska jedinica	Opis
AU 1	Nose wrinkler
AU 2	Jaw z-push
AU 3	Jaw x-push
AU 4	Jaw drop
AU 5	Lower lip drop
AU 6	Upper lip drop
AU 7	Lip stretcher
AU 8	Lip corner depressor
AU 9	Lip presser
AU 10	Outer brows raiser
AU 11	Inner brows raiser
AU 12	Brow lowerer
AU 13	Eyes closed
AU 14	Lid tigthener
AU 15	Upper lid raiser
AU 16	Rotate eyes left

Tablica 3.1. Dostupne akcijske jedinice

Funkcija getTrackingData() kao povratnu vrijednost vraća kod statusa (stanja) u kojem se tracker nalazi. Statusni kodovi su dani u tablici 3.2.

Kod	Opis
TRACK_STAT_OK	Praćenje je uključeno
TRACK_STAT_OFF	Praćenje je isključeno
TRACK_STAT_INIT	Inicijalizacija
TRACK_STAT_RECOVERING	Oporavak

Tablica 3.2. Statusni kodovi trackera

Funkcija getTrackingData popunjava podatke u TrackingData objekt samo ako je TRACK_STAT_OK kod vraćen. [7]

3.3 VisageTrackerObserver

Kako bi se omogućilo asinkrono dohvaćanje podataka u ovom radu je korišten VisageTrackerObserver. VisageTrackerObserver je apstraktni razred sa virtualnom metodom Notify(). Metoda Notify() se poziva asinkrono jednom za svaku sliku (eng. frame) s kamere koja se dohvati [7]. Primjer izgleda naslijeđene (derivirane) klase je dan u nastavku.

```
class SimpleOBserver : public VisageTrackerObserver
{
public:
	void Notify(VisageTracker2 *tracker, long timeStamp)
	{
		TrackingData data;
		int status;

		status = tracker->getTrackingData(&data);
		if(status == TRACK_STAT_OK)
		{
			//...povuci podatke...
		}
	}
}
```

U aplikaciji ovog rada je implementirano čitanje podataka u globalne varijable. Prije nego što se ažuriraju podaci za objekt koji upravljamo pokretom lica (u ovom slučaju svemirski brod), obavi se neki N broj čitanja. Pošto praćenje lica u nekim trenucima ne može odrediti položaj lica (u nekom od N čitanja), uzima se aritmetička sredina vrijednosti od tih N čitanja, umjesto samo zadnjeg čitanja. A ukoliko se ne uspije odrediti položaj lica za vrijeme nekog ažuriranja, svemirski brod ostaje na svom prethodnom mjestu.

4. Zaključak

Rezultat ovog završnog rada je uspješno napravljena igrica u kojoj se glavni objekt, svemirski brod, upravlja pokretima lica. Za iscrtavanje objekata su korištene tehnologije iz DirectX-a i programski jezik C++ kako bi se dobilo na brzini pri iscrtavanju objekata. Na poboljšanju brzine rada aplikacije je također utjecalo i strojno impelentirano množenje matrica prilikom animacije objekata. Rezultati izvođenja daju vrlo dobre rezultate, a zacrtani ciljevi u ostvareni. Jedan od većih problema pri izgradnji aplikacije je bio inicijalizacija DirectX komponenti. Za inicijalizaciju komponenti se utroši veća količina vremena pri proučavanju arhitekture DirectX-a, ali sav se trud isplati na brzini iscrtavanja objekata. Loša strana DirectX-a je portablinost aplikacije. Ovu aplikaciju će se moći pokretati samo na operacijskom sustavu Windows. Detekcija kolizija između objekata daje dobre rezulate, a prije svega je jednostavna i nije procesorski zahtjevna. Zbog jednostavnosti, odnosno aproksimaciji objekata, mogu se dogoditi greške u izvođenju programa pa da neki sudari budu nedetetirani. Tu se aplikacija može poboljšati boljom aproksimacijom objekata, ali su rizik performanse. Također može se primjetiti da aplikacija ovisi o osvijetljenosti prostorije u kojoj se korisnik nalazi i kameri koju korisnik upotrebljava. Zasigurno se može očekivati poboljšanje algoritama za detekciju lica, što bi dodatno poboljšalo rad aplikacije. Poboljšanje aplikacije može se dobiti i nadogradnjom verzije DirectX-a. U ovoj aplikaciji korišten je DirectX11, a Microsoft je najavio novu verziju DirectX12 s boljim performansama i boljim iskorištavanjem višejezgrenosti procesora.

Literatura

[1] M. Casamassina, »wii controllers: unlocking the secrets,« 2006. [Mrežno]. Available: http://www.ign.com/articles/2006/07/15/wii-controllers-unlocking-the-secrets.

[2] »DirectX Documentation,« Microsoft, 2015. [Mrežno]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/bb205075(v=vs.85).aspx.

[3] F. Luna, Introduction to 3D Game Programming with DirectX 11, 2012.

[4] »rastertek,« 2014. [Mrežno]. Available: http://www.rastertek.com/dx11tut05.html.

[5] M. Čupić i Ž. Mihajlović, 2014. [Mrežno]. Available: http://www.zemris.fer.hr/predmeti/irg/knjiga.pdf.

[6] »Candide,« 2012. [Mrežno]. Available: http://www.icg.isy.liu.se/candide/main.html.

[7] *VisageSDK Documentation*, Visage Technologies, 2014.

Animacija pokretana praćenjem lica

Sažetak

U ovom radu je prikazana jedna od mnogih mogućnosti uporabe tehnologije zasnivane na praćenju lica korisnika. Za praćenje lica korištena je tehnologija VisageSDK. Ta je tehnologija upotrebljena kao ulazni podaci u igrici. Ti ulazni podaci upravljaju animacijom objekta (svemirskog broda) s kojim se izbjegava druge objekte (meteori). Kolizija između svemirskog broda i meteora, odnosno uspješno prijeđene prepreke označavaju kraj igre. Za iscrtavanje objekata na ekran korištena je DirectX tehnologija. Objekti su predstavljeni kao skupina trokuta u prostoru koji se crtaju na ekran. Kolizije između objekata su pojednostavljeni i aproksmirani kako bi se dobilo brzini izvođenja aplikacije. Također, da bi aplikacija radila što je brže moguće, korišten je programski jezik C++.

Ključne riječi: Praćenje lica, DirectX, Animacija, Detekcija Kolizije, C++, VisageSDK

Performance-driven animation using DirectX

Summary

In this paper is shown one of many possibilities of using face tracking technology in applications. VisageSDK is used to track face of user. Data given by face tracking is used as an input to application. These datas are controlling the animation of ingame object (spaceship), with which user has to evade an incoming object (meteor). Collision between spaceship nad meteor, or successfuly passed obstacles end the game. In application DirectX technology is used to draw objects to screen. Objects are represented as a group of triangles that are rendered to screen. Collision between objects are simplified and approximated to gain better performance of application. Also, to make application as fast as possible, application has been made in C++ programming language.

Ključne riječi: Face tracking, DirectX, Animation, Collision Detection, C++, VisageSDK