SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3076

ISCRTAVANJE OVISNO O POGLEDU

Jana Tomljanović

Zagreb, lipanj 2013.

SADRŽAJ

1. UVOD

Uočavanje i praćenje ljudskog lica i crta lica (usta, nos, oči, obrve...) na slikama ili video zapisima omogućuje bezbroj primjena u raznim aplikacijama poput animacije likova, proširene stvarnosti, upravljanje kamerom ili avatarom u igrama, prepoznavanje i praćenje osoba, analiza izraza lica u svrhu istraživanja tržišta a naposljetku i za iscrtavanje ovisno o pogledu. [1]

Za implementaciju svih ovih primjena imamo na raspolaganju razna tehnološka rješenja a odluka koje ćemo rješenje koristiti ovisi o našim željama i zahtjevima: radimo li sa slikama ili video zapisom, trebamo li podatak o 2D ili 3D koordinatama lica, treba li nam samo pozicija lica ili želimo i podatke o izrazu lica, koja nam je ciljana platforma?

U okviru ovog rada, gdje nam je zadaća bila implementirati program za simulaciju iscrtavanja ovisno o pogledu, paket programa visage|SDK Face Track u kombinaciji s Unity game engine-om nam je bio dostatan za jednostavno i efikasno rješenje.

Kroz ovaj rad ćemo ukratko objasniti problematiku iscrtavanja ovisnog o pogledu, razraditi plan rješavanja tog problema, objasniti alate koje namjeravamo koristiti te na kraju prokomentirati dobiveno rješenje i kako smo do njega tehnološki došli.

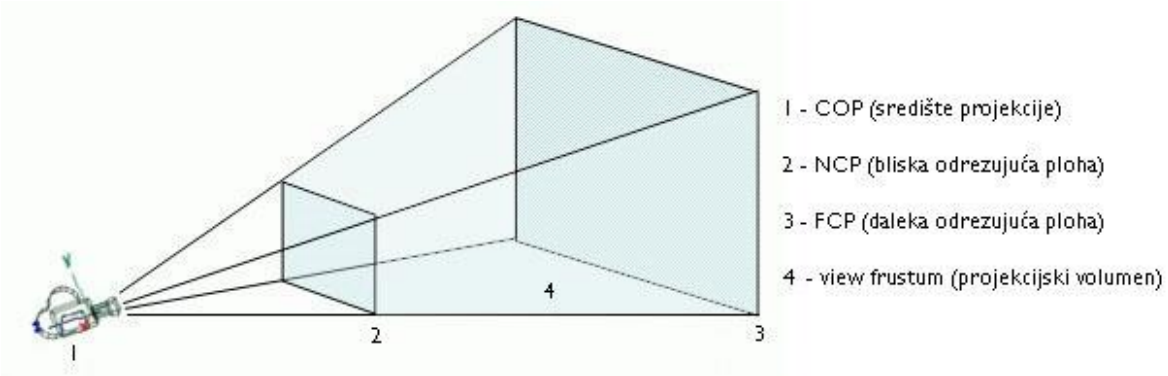
2. ISCRTAVANJE OVISNO O POGLEDU

Zadaća ovog rada je implementirati sustav za iscrtavanje ovisno o pogledu (eng. View dependent rendering). To je tehnika za 3D iscrtavanje u stvarnom vremenu uz dinamičku promjenu perspektive temeljena na rezultatima 3D praćenja pokreta glave korisnika preko obične web kamere. Time se postiže efekt da korisnik promatra iscrtane predmete s raznih strana jednostavnim pokretima glave.

U narednim potpoglavljima ćemo podrobnije objasniti problematiku iscrtavanja ovisnog o pogledu te donijeti prijedlog rješenja tog problema.

2.1 OPIS PROBLEMATIKE

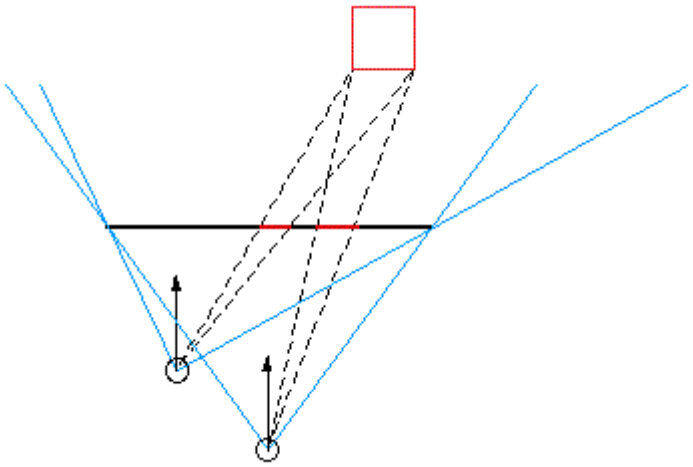
Da bi shvatili na koji način radi iscrtavanje ovisno o pogledu prvo moramo objasniti model kamere i virtualne scene. [2] Sliku jednostavnog modela kamere i njene parametre možemo vidjeti na slici 1.



Slika 1. Model kamere

Virtualna kamera u sceni određuje koji dio scene će se iscrtati na ekranu. Ako provučemo polupravce od središta projekcije kroz rubove projekcijskog prozora i ograničimo prostor koji ti polupravci opisuju bliskom i dalekom odrezujućom plohom dobijemo projekcijski volumen oblika krnje piramide. Samo onaj dio scene koji se nalazi unutar projekcijskog volumena će se projicirati na projekcijski prozor i možemo ga vidjeti na ekranu. Ukoliko mijenjamo položaj kamere u odnosu na projekcijski prozor mijenjamo i projekcijski volumen a tako izravno i predmete koji će se iscrtati na ekranu.

Proučimo sada model iscrtavanja ovisno o pogledu. Na slici 2. prikazan je tlocrtni pogled na dvije kamere u prostoru, projekcijski volumen koji stvaraju i projekcije predmeta na projekcijski prozor.



Slika 2. Usporedba projekcija dviju kamera

Već po veličini projekcije na slici možemo naslutiti da će se slike iscrtane ovim kamerama međusobno razlikovati no pogledajmo na slici 3. i iscrtane slike kocke iz primjera.

Slika 3. Projekcije iz kamera različito postavljenih u sceni

Kada translaciju kamere u prostoru povežemo s translacijom glave korisnika aplikacije dobijemo korisnu dodatnu mogućnost našeg programa. Mijenjanjem položaja glave mijenja se pogled u virtualnu scenu i implementirali smo iscrtavanje ovisno o pogledu. Primijetimo da nam za iscrtavanje ovisno o pogledu nije bitan podatak o rotaciji glave korisnika. Rotacija nam nije potrebno zbog činjenice da se ona automatski postavlja tako da kamera gleda u smjeru polupravca od središta projekcije kroz središte projekcijskog prozora kako bi osigurali da u svakom trenutku gledamo u projekcijski volumen a ne izvan njega.

2.2 ŽELJENA FUNKCIONALNOST I PRIJEDLOG RJEŠENJA

Naš je zadatak kontinuirano mijenjati poziciju kamere u prostoru i time postići dojam kontinuirane promjene perspektive pogleda. Radi što efektnijeg rezultata mijenjanje položaja kamere nećemo postići konvencionalnim ulaznim jedinicama poput miša i tipkovnice već ćemo kameru upravljati pokretima vlastite glave koristeći samo web kameru i program za praćenje pokreta lica. Pri tome moramo paziti da na kameru ne želimo prenositi i rotacije glave već samo njenu translaciju.

Iscrtavanje ovisno o pogledu bi mogli postići korištenjem podataka o poziciji glave koje dobivamo iz obične web kamere i postavljanjem jedne virtualne scene u kojoj ćemo pomicati kameru. Prvo inicijaliziramo elemente virtualne scene ovisno o početnoj poziciji glave. Zatim za svaki odmah glave od svoje inicijalne pozicije odmičemo i virtualnu kameru za isti korak. To radimo tako da za svaki period osvježavanja prikaza ekrana provjerimo stanje korisnika ispred kamere i ovisno o tome ažuriramo stanje virtualnih očiju korisnika.

Koristeći sustav za praćenje pokreta glave dobiven na zavodu i Unity game engine za kreiranje virtualne scene napravit ćemo simulaciju promatranja predmeta iz raznih perspektiva pomakom glave.

3. KORIŠTENI ALATI

U izradi programa koji simulira mijenjanje slike na ekranu ovisno o poziciji naše glave koristili smo dva osnovna programska paketa: Visage|SDK Face Tracker i Unity game engine. Koristeći Head Tracking (praćenje pokreta glave) mogućnost Visage|SDK programskog paketa pomoću obične web kamere smo dobili poziciju i rotaciju glave u 3D koordinatama te smo zatim te informacije koristili u Unity projektu kako bi pomicali kameru postavljenu u jednoj jednostavnoj sceni i tako dobili dojam promjene perspektive pogleda ovisno o promjeni položaja vlastite glave.

U sljedećim potpoglavljima ćemo podrobnije opisati navedene programske pakete.

3.1 VISAGE|SDK

Visage|SDK integrira opsežan broj računalnih tehnologija za animaciju likova i računalni vid u skup razvojnih alata koji se lako koriste i imaju široko područje primjene poput animacije virtualnih likova, stvaranje virtualne ili proširene stvarnosti, praćenje pokreta osoba, upravljanje avatarom u igri, analiziranje izraza lica, itd. Stoga ove alate možemo izravno koristiti i u našoj simulaciji iscrtavanja ovisno o pogledu. [3]

Visage|SDK nudi razne programe u odvojeno licenciranim paketima a neki od njih su Face Track (praćenje pokreta lica i glave u video zapisu), Head Track (praćenje pokreta glave u video zapisu), Face Detect (otkrivanje izraza lica na slikama), Visual TTS (optički pretvarač teksta u govor), Lip Sync (automatska sinkronizacija usana u stvarnom vremenu), itd. [3]

U okviru iscrtavanja ovisnog o pogledu najkorisniji nam se činio programski paket Face Track. To je izuzetno moćan i u potpunosti podesiv alat za praćenje pokreta glave i izraza lica. Njegova zadaća je pronaći i pratiti lice u video zapisima u stvarnom vremenu (može raditi brzinom od 30 sličica u sekundi) te vratiti informacije o 3D položaju glave (pomak i rotacija), usmjerenosti pogleda, koordinatama crta lica (usta, nos, oči, obrve, uši) i ostale informacije ovisne o konkretnoj primjeni. [4]

Praćenje radi s kamerama u boji, crno-bijelim kamerama, infracrvenim i termalnim kamerama, a sav software dolazi u dobro dokumentiranom C++ razvojnom programu.

Osnovni princip rada programa se temelji na podešavanju 3D modela na sliku lica te procjeni 3D pokreta glave i crta lica. [4]

Tracker za svaki obrađeni kadar video isječka vraća od korisnika tražene vrijednosti poput 3D pozicije glave, obrisa usta, pozicije brade, rotacije očiju (usmjerenost pogleda) i ostalo. Tracker radi potpuno automatski: brzo se oporavlja od bilo kojih prekida u praćenju glave uslijed okretanja glave od kamere, prekrivanja dijelova lica ili odlaska s prostora snimanja te se potpuno automatski inicijalizira svaki put kada nova osoba stupi pred kameru. [4]

Tracker može pratiti pokrete s web kamere ili iz AVI video zapisa, koristi minimalnu rezoluciju od 320x240 piksela (moguće je korištenje i većih rezolucija radi boljeg rezultata poput 800x600 piksela) od čega minimalna širina lica kojeg prati mora iznositi 80 piksela i glava ne smije biti pod kutem većim od 45° a za praćenje pokreta mu nisu potrebni dodatni markeri ili šminka. [4]

Ne zahtijeva veliku procesorsku moć pa ga možemo pokretati na osobnom računalu s osnovnim dvojezgrenim procesorom.

3.2 UNITY

Unity je pokretački sustav za razvoj računalnih igara (eng. game engine) koji podržava niz platformi uključujući Windows, MAC, Linux, iOs, Android, Nintendo Wii i Flash. Zadaća pokretačkog sustava je brinuti se o osnovnim zadacima koji se odvijaju u pozadini igre poput iscrtavanja, detekcije kolizije, umjetne inteligencije, odgovora na korisnički unos, i sl. kako programeri igre ne bi morali gubiti vrijeme na implementaciju tih osnovnih funkcionalnosti koje su zajedničke svim igrama već se mogu usredotočiti na samu implementaciju igre i modeliranje likova i scene. [5]

Unity pruža osnovne funkcionalnosti jednog game engine kao što su iscrtavanje (moguće je birati želimo li iscrtavanje izvesti koristeći Direct3D ili OpenGL), fizički engine (brine za detekciju kolizija u sceni, simulaciju gravitacije, simulaciju osnovnih sila u prirodu i sl. koristeći NVIDIA PhysX fizički engine), pisanje skripti koje određuju ponašanje predmeta u igri (skripte u Unity-u se razvijaju koristeći Mono, a mogu se pisati u programskim jezicima JavaScript, C# ili Boo), animacije, dodavanje zvuka (glazbena pozadina kroz igru ili zvukovi koje pokrene neki okidač), umjetna inteligencija (opis ponašanja likova koje ne kontrolira

igrač), mrežno igranje, upravljanje memorijom, upravljanje dretvama, stvaranje grafa scene i još nekolicina jednako korisnih ali nespomenutih funkcionalnosti. [6]

Osnovni dijelovi Unity sučelja su prozor scene, prozor projekta sa svim svojim komponentama, prozor hijerarhije scene i prozor pregledač.

U prozoru projekta možemo vidjeti komponente koje koristimo (eng. Assets) raspoređene unutar datoteka koje pobliže opisuju njihovu kategoriju. Klikom na pojedinu kategoriju izlistaju nam se sve komponente unutar nje a klikom na samu komponentu dobijemo u prozoru pregledaču (eng. inspector) osnovne detalje o samoj komponenti. Pojedine komponente možemo koristiti kao predmet igre unutar scene ako se radi o modelu ili ih pridijeliti nekom predmetu ako se radi o materijalu, zvuku, animaciji ili skripti. Unutar prozora projekta nalaze se i sve razine igre koje smo razvili spremljene kao scene a naposljetku ih pri izgradnji projekta slažemo u željeni redoslijed.

Prozor scene nam pokazuje trenutno stanje našeg prizora u igri. Unutar tog prozora razmještamo predmete igre u prostoru, dodajemo svjetla i kamere te se možemo orijentirati po sceni i dobiti pregled konačnog izgleda igre ili dijela igre.

Kada neku komponentu dodamo u scenu ona automatski postaje predmet igre (eng. game object) i možemo joj unutar pregledača dodavati komponente poput skripte, zvuka ili teksture i mijenjati parametre poput translacije, rotacije i veličine. Svi predmeti igre su izlistani u prozoru hijerarhije scene i tamo ih možemo grupirati i stvarati roditelj-djeca veze radi lakšeg i efikasnijeg rada sa scenom.

Zahvaljujući mnoštvu Unity alata razvoj igre teče brzo i jednostavno a komponente možemo i uvoziti iz drugih programa poput Maye, 3ds Maxa i Blendera. Sve predmete u sceni možemo dodavati i mijenjati drag-and-drop metodom postepeno povećavajući složenost scene. Kada smo predmetima još dodali i skripte za ponašanje i ostale komponente te smo zadovoljni s izgledom scene možemo je odmah testirati unutar Unity okruženja i ukloniti sve neželjene pojave. Naposljetku kada smo gotovi s igrom u cijelosti možemo je izgraditi za željenu platformu u par klikova.

4. IMPLEMENTACIJA

Simulaciju iscrtavanja ovisnog o pogledu odlučili smo napraviti koristeći upravo opisane Visage|SDK Face Tracker i Unity game engine. Uz Visage|SDK programe dobili smo i niz testnih primjera za lakše razumijevanje principa rada Visage sustava pa tako između ostalih i primjer jednog Unity projekta unutar kojeg pozivamo funkcije Face Trackera. Upravo nam je taj primjer poslužio kao osnova za izgradnju programa za iscrtavanje ovisno o pogledu te ćemo rad na tom projektu opisati u potpoglavljima koji slijede. Integracija Visage|SDK Face Trackera i Unity-ja se odvija preko dodatka VisageTrackerUnityPlugin.dll koji obuhvaća sve osnovne pozive funkcija i prilagođava ih za poziv iz Unity skripti. U našem slučaju dodatak koristimo bez mijenjanja ali je moguće implementirati dodatne funkcionalnosti Trackera.

4.1 UNITY PROJEKT

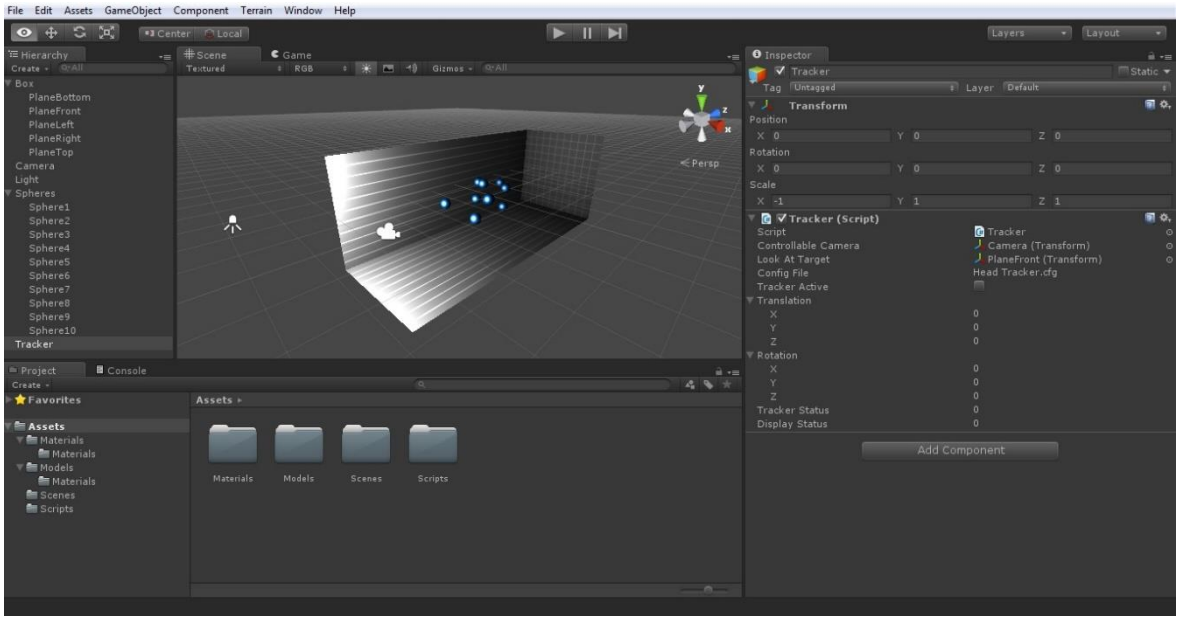
Iako nam je primjer projekta u kojemu se unutar Unity game engine-a koristi praćenje pokreta lica kako bi na lice korisnika postavili virtualne naočale poslužio kao odlična baza za shvaćanje integracije Visage|SDK sustava u Unity sam projekt se dosta razlikovao od onoga što se u našem zadatku tražilo pa smo za početak otvorili novi prazan projekt. Razmišljajući koja scena bi mogla vjerno dočarati efekt promjene perspektive uslijed promjene položaja glave odlučili smo zadržati stvari jednostavnima i sastaviti scenu s desetak nasumično raspoređenih kugli.

Prvo sam u scenu postavila kameru na globalnim koordinatama (0, 0, -350) i pozicionirala je da gleda u smjeru Z-osi. Zadala sam da je perspektivna i da joj je vidno polje 35°. Kugle sam stvorila koristeći modelatorske alate unutar Unity-ja tako što sam odabrala naredbu GameObject – Create Other – Sphere a zatim dobivenu kuglu duplicirala i naposljetku sve kugle postavila tako da se međusobno blago prekrivaju i dobro simuliraju paralaksu te ih grupirala i grupu postavila na globalne koordinate (0, 0, 200). Kuglama sam dodala spekularno osvjetljenje i plavu boju. Zatim sam u scenu dodala kvadar kojemu nedostaje prednja stranica tako što sam grupirala pet ravnina stvorenih naredbom GameObject – Create Other - Plane. Ovaj krnji kvadar mi je služio kao okolina. Unutar kvadra sam smjestila i kugle i kameru a na stranice kvadra sam postavila teksturu koja podsjeća na križaljku kako bih dobila što bolji efekt promjene perspektive pogleda pri pomicanju kamere. Kako bi se

kugle i tekstura na ravninama dobro vidjele iza kamere sam pod blagim nagibom postavila točkasti izvor svjetlosti i usmjerila ga prema stražnjoj stranici krnjeg kvadra.

Za privođenje scene kraju još mi je samo ostalo postaviti samu srž cijele scene – Tracker. To je objekt unutar igre kojemu dodajemo Tracker skriptu koja obavlja komunikaciju između Visage|SDK Face Trackera i Unity-ja. Kao što vidimo na slici 4. skripta zahtjeva od nas da odredimo kojom kamerom će upravljati Tracker i mi smo za tu poziciju odabrali jedinu kameru u sceni i zadali joj da gleda u pozadinu kvadra, no više o tome u potpoglavlju Skripta „Tracker“.

Konačan izgled scene u razvojnom okruženju možemo vidjeti na slici 4. Ovime smo završili modelatorski dio posla i preostaje nam iskodirati ponašanje kamere preko skripte Tracker.



Slika 4. Unity Projekt

4.2 SKRIPTA „TRACKER“

Da bismo kontrolirali kameru i time dobili dojam promjene pogleda ovisno o promjeni pozicije naše glave moramo napisati skriptu koja će koristeći podatke koje joj šalje Face Tracker translatirati odabranu kameru u našem projektu. Stoga sam u C# jeziku napisala skriptu „Tracker“ koja radi upravo opisano.

Na početku same skripte definiram koje ću sve funkcije koristiti iz VisageTrackerUnityPlugin.dll dinamičke knjižnice:

- funkcija za inicijalizaciju trackera:

  void _initTracker(string config)

- funkcija za pokretanje trackera:

  bool _trackFromCam( )

- funkcija za zaustavljanje trackera:

  void _stopTracker( )

- funkcija koja vraća trenutnu translaciju i rotaciju glave te status trackera:

  void _get3Ddata(out float tx, out float ty, out float tz, out foat rx, out float ry, out float rz, out int status)

Definiram još par varijabli koje ću koristiti poput Transform ControllableCamera (atribut koji biram u Unity sceni a određuje na koju će se kameru primijeniti transformacije), Transform LookAtTarget (također atribut koji biram u Unity sceni a određuje u kojem će smjeru kamera gledati), Vector3 Translation (u ovu varijablu spremam podatke o translaciji koje dobivam s web kamere), bool TrackerActive (zastavica koja daje informaciju o tome je li tracker aktivan ili ne) i string ConfigFile (u ovu varijablu spremam ime .cfg datoteke koju ću koristiti pri inicijalizaciji trackera).

Prva metoda koju moram pozvati prije svega je void Start (). Unutar nje inicijaliziram tracker i postavljam kameru kojom ću upravljati na početnu poziciju. Zatim zovem metodu koja upravlja GUI događajima i u njoj određujem da će praćenje pokreta glave i preslikavanje tih pokreta na kameru započeti klikom na gumb „Track from camera“ a završiti klikom na gumb „Stop tracking“.

Nakon svih početnih inicijalizacija dolazimo do glavnog dijela skripte – funkcije Update (). Ova funkcija se poziva jednom u svakom okviru. Svakim pozivom koristeći funkciju _get3Ddata ažurira translaciju, rotaciju i stanje trackera te ako je tracker aktivan ažurira poziciju kamere koristeći naredbu .transform.Translate. U nastavku možemo vidjeti isječak koda:

```
_get3DData(out Translation.x, out Translation.y, out Translation.z, out Rotation.x, out Rotation.y, out Rotation.z, out TrackerStatus);

TrackerActive = TrackerStatus != 0;

ControllableCamera.transform.position = CameraStartingPosition;

ControllableCamera.transform.Translate(500*Translation.x, 500*Translation.y, -500*Translation.z);
```

ControllableCamera.transform.LookAt(LookAtTarget);

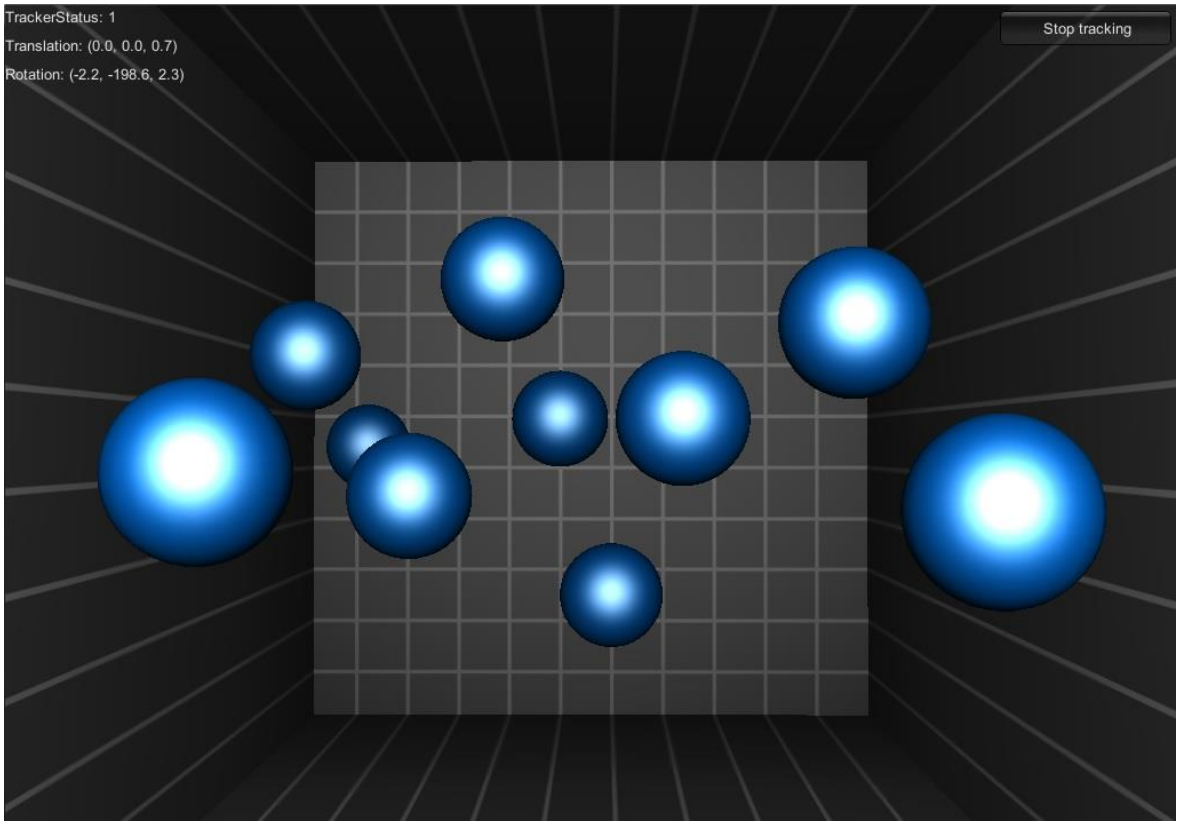
Posljednja naredba u isječku koda nam koristi za usmjeravanje pogleda prema središtu scene i zaslužna je za to da nam pri velikim translacijama pogled ne bježi izvan scene.

4.3 KONAČNO RJEŠENJE

Kada izgradimo i pokrenemo projekt dobijemo program koji vjerno dočarava promjene perspektive uslijed pomicanja naše glave. Pri pokretanju nas dočeka scena prikazana na slici 5. Vidimo samo boju pozadine, u gornjem lijevom kutu ispis stanja trackera i očitavanja trenutne translacije i rotacije a u gornjem desnom kutu se nalazi gumb Track from camera.

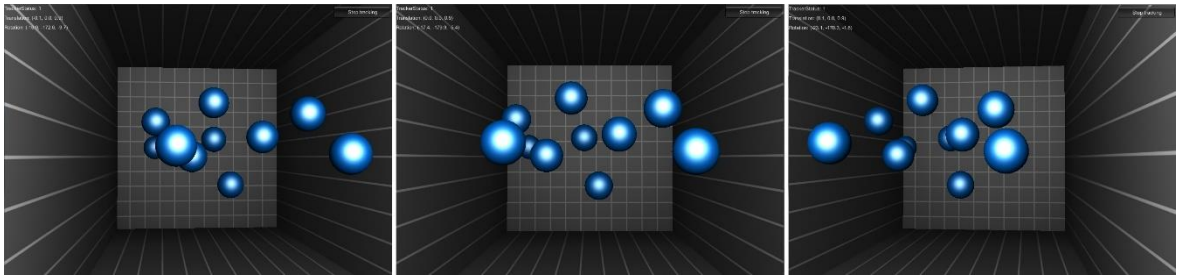
Slika 5. Pocetna scena programa

Klikom na gumb pokreće se web kamera na našem računalu, te se nakon inicijalizacije trackera počinje primjenjivati transformacija kamere u ovisnosti o pomaku naše glave i kroz par sekundi se pojavljuje scena poput one na slici 6.



Slika 6. Aktivan tracker

Kamera se postavlja unutar krnjeg kvadra ispred grupe kugli te pomakom glave u bilo kojem smjeru mijenjamo perspektivu iscrtavanja scene. Klikom na gumb Stop Tracking u gornjem desnom kutu prekida se rad kamere i trackera te se program vraća na početnu scenu, a klikom na tipku Esc zatvaramo cijeli prozor. Demonstraciju promjene perspektive u ovisnosti o promjeni položaja glave možemo vidjeti na slici 7.



Slika 7. Promjena perspektive pogleda

5. ZAKLJUČAK

Iscrtavanje ovisno o pogledu ima u budućnosti veliki potencijal postati korisna dodatna mogućnost u programima a pogotovo u području video igrica. Zahvaljujući razvoju sve boljih algoritama za uočavanje i praćenje lica te činjenice da danas većina računala i mobilnih uređaja dolaze s ugrađenom jednostavnom kamerom možemo sustavima za praćenje pokreta glave predvidjeti mjesto među korištenim načinima unosa podataka.

Simulacija iscrtavanja ovisnog o pogledu koju smo implementirali u ovom radu je služila samo za demonstraciju mogućnosti ove tehnike ali svakako nije jedino područje njene primjene. Iscrtavanje ovisno o pogledu bi se moglo upotrebljavati u raznim specijaliziranim programima u kojima postoji potreba za čestim promjenama perspektive pogleda ili sagledavanja nekog predmeta sa svih njegovim strana poput programa za iscrtavanje 3D modela. Također bi iscrtavanje ovisno o pogledu svakako mogli skoro vidjeti i u računalnim igricama zbog postizanja nesumnjivo dojmljivih efekata uz potpuno intuitivno i jednostavno za korištenje sučelje.

Program iz rada bi se mogao proširiti na simulaciju kompleksnijih scena od trenutno implementirane u kojima bi još više do izražaja došala tehnika iscrtavanja ovisno o pogledu ili bi se mogao prilagoditi za neke druge platforme osim Windowsa poput iOS ili Androida.

6. LITERATURA

[1] Visage Technologies, *Computer Vision applied to human faces,* http://www.visagetechnologies.com/technology/computer-vision/, 23. 05. 2013.

[2] I. S. Pandžić, T. Pejša, K. Matković, H. Benko, A. Čereković, M. Matijašević: Virtualna okruženja: Interaktivna 3D grafika i njene primjene. 1. izdanje. Zagreb: Element, 2011

[3] Visage Technologies, *Visage|SDK*, http://www.visagetechnologies.com/products/visagesdk/, 24.05.2013.

[4] Visage Technologies, *Face Track*, http://www.visagetechnologies.com/products/visagesdk/facetrack/, 24.05.2013.

[5] Unity, *Create the games you love with Unity*, http://unity3d.com/unity/, 25.05.2013.

[6] Wikipedia, *Unity(game engine)*, http://en.wikipedia.org/wiki/Unity_(game_engine), 25.05.2013.

7. SAŽETAK

Iscrtavanje ovisno o pogledu je tehnika za 3D iscrtavanje u stvarnom vremenu temeljena na rezultatima praćenja pokreta glave korisnika koje koristi za dinamičku promjenu perspektive u virtualnoj sceni tako što pomiče virtualnu kameru ovisno o pomaku glave korisnika. U ovoj tehnici pratimo samo translaciju glave korisnika i pogled usmjeravamo prema ekranu koji nam je kao prozor u virtualni svijet.

Za ostvarivanje ove tehnike smo koristili dva alata: Visage|SDK Face Tracker i Unity game engine. Koristeći Face Tracker smo dobivali podatke o položaju glave a Unity smo koristili kao razvojnu okolinu unutar koje smo definirali virtualnu scenu i koristili podatke iz Face Trackera kako bi upravljali kamerom u toj sceni. Za integraciju Visage|SDK Face Trackera i Unity-ja koristili dodatak VisageTrackerUnityPlugin.dll koji je prilagođavao funkcije Trackera Unity-ju.

Naposljetku smo dobili jednostavan primjer s kuglama raspoređenih unutar scene koje su uspješno demonstrirale efekt promjene perspektive ovisno o položaju glave korisnika.

8. KLJUČNE RIJEČI

Iscrtavanje ovisno o pogledu, Visage|SDK, Face Tracker, Unity, praćenje pokreta glave, dinamička promjena perspektive, model kamere, virtualna scena, iscrtavanje u stvarnom vremenu

9. SUMMARY

View dependent rendering is a 3D technique for real time rendering based on the results of user's head movement tracking that we use to dynamically change the perspective within the virtual scene by moving the virtual camera depending on the user's head movement. In this technique we only observe the user's head translation and we direct the view towards the screen which serves as a window into the virtual world.

In order to achieve this technique we used two tools: Visage|SDK Face Tracker and Unity game engine. Using the Face Tracker we gained information about the current position of the head and then we used Unity as a developing tool in which we defined our virtual scene and used the data from the Face Tracker to control the camera in this scene. To integrate Visage|SDK Face Tracker with Unity we added a VisageTrackerUnityPlugin.dll plugin which modified the Tracker's functions to work in Unity.

Lastly, we made a simple example with balls rearranged in the scene that successfully demonstrated the effect of changing the perspective depending on the user's head position.

10. KEY WORDS

View dependent rendering, Visage|SDK, Face Tracker, Unity, tracking head movement, dynamic change of perspective, camera model, virtual scene, real time rendering