SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3871

Animacija lica na osnovu slike

Andrea Gradečak

Zagreb, lipanj 2015.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

Virtualni lik predstavlja lik čovjeka na računalu u 3D obliku. Animacijom virtualnog lika prilagođavamo ga izrazima i pokretima koje želimo da simulira (na primjer, smijanje ili govor). Primjena ovakvih likova danas vrlo brzo raste - najzastupljeniji su u video igrama, no koriste se i u mnogim drugim područjima kao što su medicina, obrazovanje i marketing. Zbog toga je jedan od ciljeva računalnog vida i animacije što brža i jednostavnija izrada i animacija virtualnih likova. Jedan od načina za dobivanje virtualnog lika je iz fotografije. U ovom radu opisan je i implementiran način na koji se to može ostvariti. Pomoću detekcije karakterističnih točaka lica iz slike se generira 3D model kojeg zatim animiramo pomicanjem vrhova koji definiraju 3D model. Implementacija rada je ostvarena u Unity projektu u koji su integrirani potrebni Visage SDK alati za dobivanje 3D modela i animaciju.

2. Visage SDK

Visage SDK je skup alata tvrtke Visage Technologies koji nudi razne tehnologije za računalni vid i animaciju likova koji se mogu lako integrirati i koristiti u aplikacijama. Tehnologije koje pokriva su detekcija i praćenje lica iz kamere, videa ili slika te animacija na temelju snimke govora ili sinteze govora iz teksta [1]. U Visage SDK-u se nalazi nekoliko projekata koji služe za upoznavanje s ponuđenim tehnologijama i demonstriraju njegove mogućnosti. Kao pomoć i temelj za rad korišteni su projekti Visage Tracker Unity i Face Detector.

Visage Tracker Unity je projekt koji demonstrira integraciju Visage SDK sa skupom alata za izradu video igara Unity. U projektu se, između ostalog, koristi i pristup 3D modelu lica što je dio ovog rada te je on korišten kao temelj za Unity projekt stvoren u ovom radu.

Face Detector je projekt koji demonstrira detekciju karakterističnih točaka lica u slikama. Rezultat detekcije se prikazuje i sprema u novu slikovnu datoteku. Primjer slike i rezultata detekcije se nalazi na slikama 2.1 i 2.2.

Glavna funkcionalnost Visage SDK koja je korištena u radu je detekcija lica. Detekciju lica obavlja razred VisageFeaturesDetector, a rezultate sprema u strukturu FaceData.

2.1. Visage Features Detector

Razred VisageFeaturesDetector detektira lice i karakteristične točke lica u slikama. Za svako detektirano lice daje navedene rezultate: 3D poziciju glave, koordinate karakterističnih točaka lica i 3D model koji odgovara licu. Ovi rezultati se spremaju u jedno ili više objekata tipa FaceData, ovisno o broju detektiranih lica.

Funkcionalnosti VisageFeaturesDetectora su prikazane u već spomenutom projektu Face Detector. VisageFeaturesDetector se sastoji od sljedećih funkcija:

- `VisageFeaturesDetector ()`
  konstruktor

Slika 2.1: Slike lica za detekciju pomoću Face Detectora

- `~VisageFeaturesDetector ()`
  destruktor
- `bool Initialize (const char *path)`
  funkcija koja inicijalizira detektor pri čemu joj se kao argument mora predati kazalo koje sadrži sve potrebne datoteke za uspješnu inicijalizaciju te vraća `true` ako je inicijalizacija uspjela.
- `int detectFacialFeatures (IplImage *frame, FaceData *output, int maxFaces=1, int minFaceSize=0)`
  izvršava detekciju lica i karakterističnih točki lica iz slike zadane u argumentu `frame`. Algoritam detektira jedno ili više lica i karakteristične točke svakog detektiranog lica. Rezultati detekcije su: 3D pozicija glave, koordinate karakterističnih točaka lica i 3D model lica. Navedeni rezultati se spremaju u objekt tipa FaceData koji je naveden kao argument `output`. Argumentima `maxFaces` i `minFaceSize` može se specificirati broj lica koji se želi detektirati, odnosno minimalna veličina lica koja se želi detektirati u pikselima. Funkcija vraća broj detektiranih lica, a ako je u argumentu `maxFaces` naveden broj lica manji od broja detektiranih lica, detektira se samo prvih `maxFaces` lica i njihovi se rezultati spremaju, dok se ostatak zanemaruje. Slika `frame`

Slika 2.2: Detektirane karakteristične točke lica pomoću Face Detectora

iz koje se detektira mora sadržavati lice u frontalnom položaju i s neutralnim izrazom lica kako bi rezultati detekcije bili najispravniji. Također se preporuča da lice ne zauzima veliki dio slike, već oko 50%.

- `int detectFacialFeatures (const char *imageFileName, FaceData *output, int maxFaces=1, int minFaceSize=0)`
  izvršava detekciju kao i gore navedena funkcija istog imena s razlikom u argumentu slike `imageFileName` u kojem se predaje naziv slike u nekom od podržanih formata.
- `void drawResults (IplImage *img)`
  crta rezultate detekcije na sliku zadanu kao argument.

2.2. Face Data

Struktura FaceData služi za pohranu informacija dobivenih praćenjem preko Visage Trackera ili detekcijom preko Visage Features Detectora. S obzirom da se u ovom radu koristi Visage Features Detector, opisani su samo atributi koje on koristi i popunjava, dok ostale atribute ostavlja nedefiniranima.

Detektor vraća sljedeće podatke:

– 3D položaj glave

  sastoji se od translacije i rotacije glave i spremaju se u atributima:

  - `float faceTranslation [3]`
    translacija izražena preko 3 koordinate x, y i z
  - `float faceRotation [3]`
    rotacija izražena u radijanima također s 3 vrijednosti - rotaciju oko x, y i z koordinate

– karakteristične točke lica

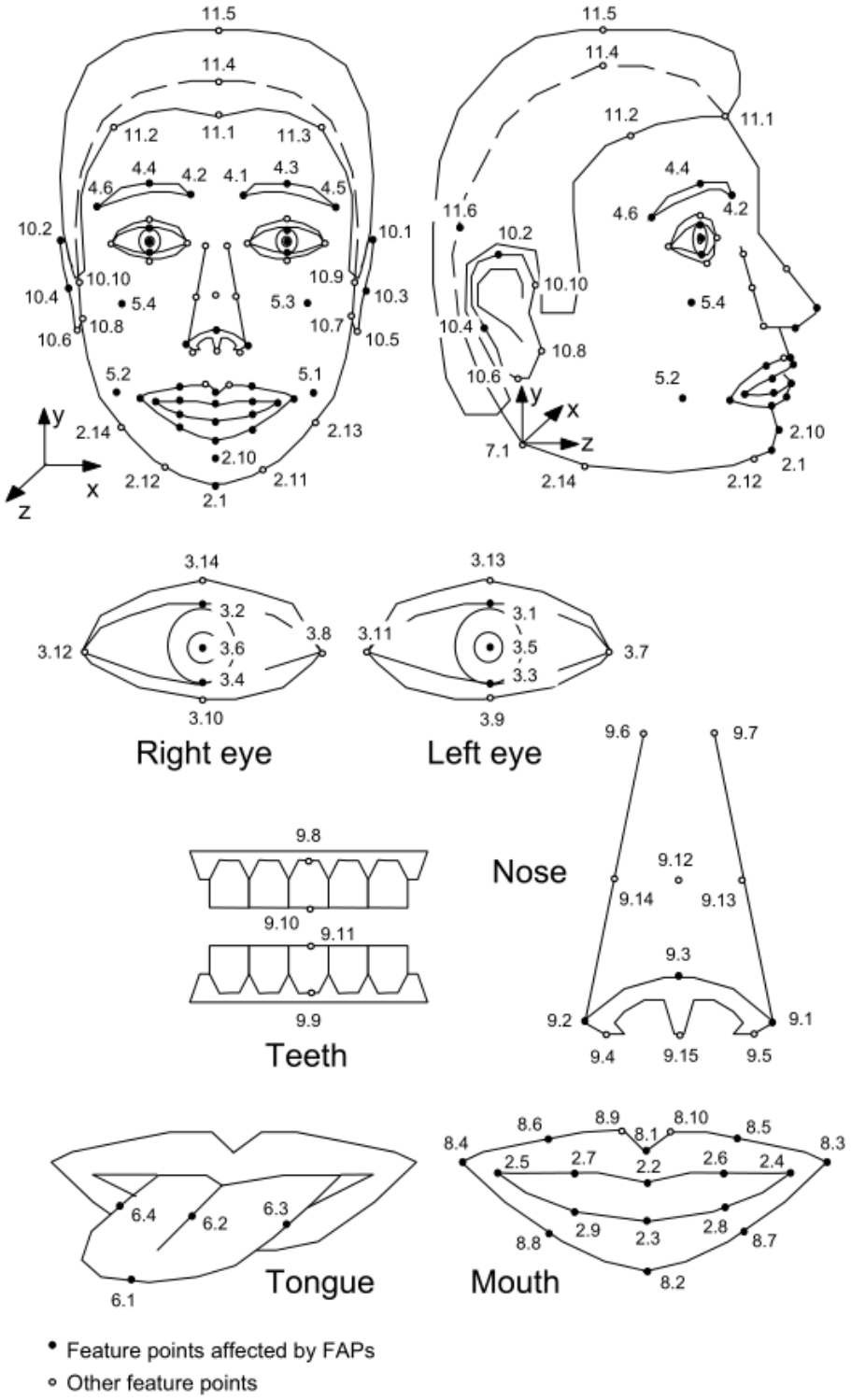
  sastoji se od 2D i 3D koordinata karakterističnih točaka lica. Spremaju se u atributima koji su tipa FDP (Face Definition Parameters) - strukture koja služi za pohranu karakterističnih točaka prema MPEG-4 FBA (Face and Body Animation) standardu. Karakteristične točke se identificiraju brojem grupe i indeksom, a prikazane su na slici 2.3. Na primjer, vrh brade pripada grupi 2 i ima indeks 1 te je identificiran kao točka 2.1. Atributi koji sadrže ove točke su:

  - `FDP * featurePoints3D`
    globalne 3D koordinate karakterističnih točaka lica
  - `FDP * featurePoints3DRelative`
    3D koordinate u odnosu na centar lica smješten između očiju
  - `FDP * featurePoints2D`
    2D koordinate karakterističnih točaka lica. Ove točke su normalizirane s obzirom na sliku pri čemu lijevi kut slike ima koordinate (0,0) a gornji desni (1,1). Također ova struktura postavlja određene točke lica kao što su točke na jeziku i zubima na 0 jer ih ne može pouzdano detektirati, a neke samo otprilike odredi zbog njihove specifične lokacije (na primjer, točka 11.4).

– 3D model lica

  3D model koji odgovara licu detektiranom na slici. Sastoji se od skupa poligona (trokuta), pripadnih vrhova i koordinata teksture spremljenih u sljedeće atribute:

  - `int faceModelVertexCount`
    broj vrhova u 3D modelu
  - `float * faceModelVertices`
    3D koordinate vrhova u obliku x, y, z. Koordinate su smještene u lokalni koordinatni sustav lica, gdje je ishodište između očiju. Kako bi



Slika 2.3: Karakteristične točke lica prema MPEG-4 FBA standardu

se transformirale u globalni koordinatni sustav mogu se koristiti atributi `faceTranslation` i `faceRotation`.

- `int faceModelTriangleCount`
  broj poligona (trokuta) u 3D modelu
- `int * faceModelTriangles`
  lista vrhova za svaki trokut. Svaki trokut sadrži 3 vrha koja se nalaze u atributu `faceModelVertices`, a navedeni su u smjeru obrnutom od smjera kazaljke na satu.
- `float * faceModelTextureCoords`
  koordinate teksture oblika u,v za svaki vrh sadržan u atributu `faceModelVertices`. Tekstura koja se koristi je slika iz koje se detektira lice.

3. Implementacija

Ideja ovog rada je korištenje Visage Features Detectora koji bi najprije obavio detekciju karakterističnih točaka lica. Rezultati detekcije, kako je već spomenuto, su spremljeni u strukturu FaceData iz koje možemo pristupiti vrijednostima poligona i vrhova koji su nam potrebni za dobivanje 3D modela lica. U Visage Tracker Unity projektu je korišten Visage Tracker Unity Plugin, omotač koji omogućava korištenje funkcionalnosti Visage Trackera u programskom jeziku C# (koji se koristi u skriptama Unity projekta). Pošto se ovaj rad temelji na detekciji sa slika, analogno je izrađen Face Detector Plugin koji omogućava korištenje funkcionalnosti Visage Features Detectora, a ne Visage Trackera. Funkcije Face Detector Plugina su zatim importane u Unity projekt i pozvane u skripti.

3.1. Face Detector Plugin

Face Detector Plugin izrađen je u programskom jeziku C++. Funkcije koje sadrži Face Detector Plugin su ekvivalente funkcijama Visage Features Detectora:

- `bool _initDetector(char* config)`
  stvara novi primjerak VisageFeaturesDetectora i inicijalizira ga
- `int _detectFacialFeatures(char* imageName)`
  otvara sliku zadanu imenom u argumentu, inicijalizira FaceData te navedenim parametrima poziva funkciju detectFacialFeatures iz detektora
- `bool _getFaceModel(int* vertexNumber, float* vertices, int* triangleNumber, int* triangles, float* texCoord)`
  u navedene argumente sprema određene podatke iz strukture FaceData
- `void _get3DData(float* tx, float* ty, float* tz, float* rx, float* ry, float* rz)`
  vraća parametre translacije i rotacije iz strukture FaceData

3.2. Unity projekt

Kostur Unity projekta korištenog u ovom radu preuzet je iz projekta Visage Tracker Unity. Glavni elementi projekta su Unity objekti Detector i Display Plane. Display Plane je ravnina koja nam služi kao pozadina za 3D model lica. Time dobivamo ispravan prikaz lica koje se "uklopilo" u sliku, odnosno ne primijećuje se da je 3D model iscrtan ispred slike. 3D model lica koje želimo prikazati se stvara i iscrtava u Detectoru, odnosno istoimenoj skripti koja je kao komponenta pridružena objektu Detector.

3.2.1. Detector

U Detector skripti najprije je potrebno importati funkcije iz Face Detector Plugina kako bismo ih mogli koristiti. Skripta je pisana u programskom jeziku C# pa import funkcija obavljamo na sljedeći način:

```
    [DllImport("FaceDetectorPlugin.dll"]
public static extern bool _initDetector(string config)
```

atribut DllImport označava da iz navedene .dll datoteke želimo importati funkciju. Pri tome moramo dodati oznaku `extern` koja označava da je funkcija implementirana izvana. Također moramo uskladiti tipove argumenata i povratne vrijednosti funkcije s programskim jezikom u kojem pišemo. Konkretno u primjeru, `char*` (iz programskog jezika C++ u kojem je izrađen Face Detector Plugin) je potrebno zamijeniti sa prikladnim tipom `string` u C#-u. Na isti način importamo ostale funkcije Face Detector Plugina. U skripti je bitan redoslijed pozivanja ovih funkcija. Na primjer, ako pozovemo `_detectFacialFeatures` bez da smo prethodno inicijalizirali detektor, nećemo uspjeti obaviti detekciju. Redoslijed kojim funkcije moraju biti pozvane je:

1. `bool _initDetector(string config)`
   inicijalizaciju detektora smo pozvali argumentom ".", što označava tekući direktorij Unity projekta u kojem smo spremili sve datoteke potrebne za inicijalizaciju

2. `int _detectFacialFeatures(string imageName)`
   detekciju je potrebno obaviti prije poziva dolje navedenih funkcija jer se njome popunjava struktura FaceData kojoj one pristupaju. Kao argument funkciji predajemo put do slikovne datoteke iz koje želimo detektirati lice, a one su u ovom Unity projektu smještene u Resources folder unutar Assets foldera.

3. `bool _getFaceModel( out int vertexNumber, float[] vertices, out int triangleNumber, int[] triangles, float[] texCoords)` i `void _get3DData(out float tx, out float ty, out float tz, out float rx, out float ry, out float rz)` možemo pozvati proizvoljnim redoslijedom jer ne ovise jedna o drugoj. Funkciji `_getFaceModel` predajemo varijable u koje će se spremiti broj vrhova i poligona te polja tipa `float[]` za vrhove i koordinate teksture, odnosno `int[]` za poligone. Polja je potrebno prethodno inicijalizirati sa definiranim maksimalnim brojem vrhova `MaxVertices = 256` i poligona `MaxTriangles = 256`. Tako će za vrhove biti potrebno inicijalizirati polje `[MaxVertices * 3]` jer je svaki vrh definiran s tri koordinate, za poligone `[MaxTriangles * 3]` jer je svaki poligon (trokut) definiran s tri vrha te za koordinate teksture `[MaxVertices * 2]` jer svaki vrh ima dvije koordinate teksture. Funkciji `_get3DData` predajemo varijable u koje će spremiti parametre translacije i rotacije 3D modela. One su nam potrebne kako bismo naš 3D model ispravno uklopili u sliku. To postižemo promjenom atributa `transform.position`, odnosno `transform.rotation`, nad našim Detector objektom.

Nakon što smo pozvali funkcije i spremili podatke u odgovarajuće strukture potrebno je iscrtati 3D model i postaviti odgovarajuću teksturu te animirati 3D model.

3D model

Osim skripte, komponente Detectora su i Mesh Filter i Mesh Renderer. Mesh Filter sadrži Mesh - popis vrhova i poligona koje Mesh Renderer iscrtava i prikazuje u sceni. Kako bi se iscrtao naš 3D model potrebno je meshu predati vrhove, poligone i koordinate teksture koje smo dobili pozivom funkcije `_getFaceModel`:

```
meshFilter.mesh.vertices = Vertices;
meshFilter.mesh.triangles = Triangles;
meshFilter.mesh.uv = TexCoords;
```

Kako bismo kao teksturu koristili sliku moramo je najprije učitati u varijablu tipa `Texture2D`:

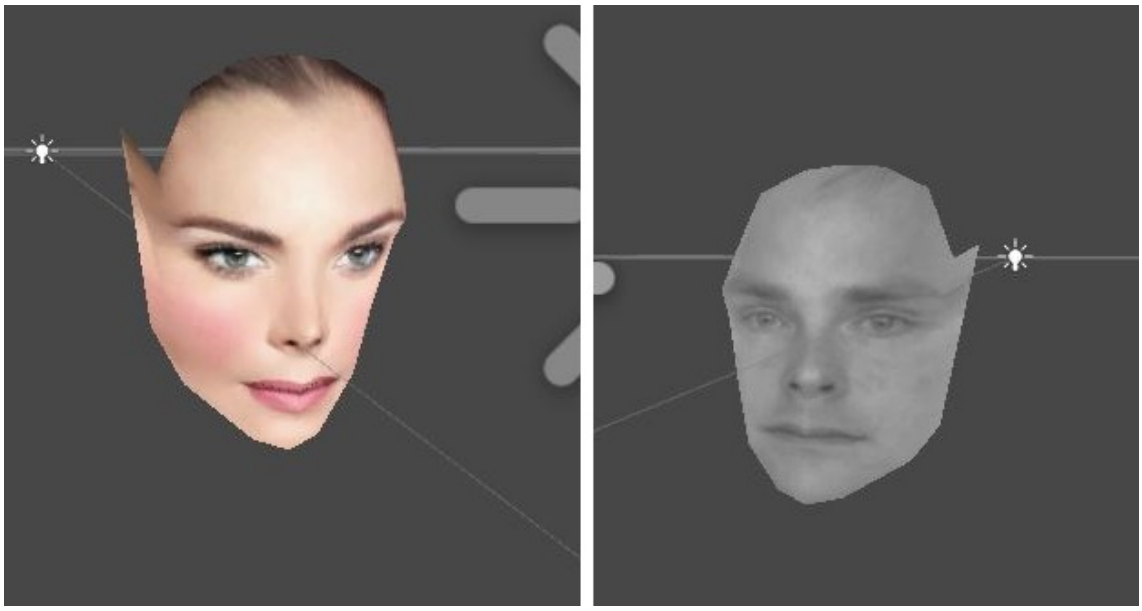
```
myTexture = Resources.Load("slika") as Texture2D;
```

pri čemu se "slika" nalazi u Resources folderu unutar foldera Assets. Još moramo u Rendereru postaviti stvorenu teksturu:

```
renderer.material.SetTexture("_MainTex", myTexture);
```

Na slikama 3.1 i 3.2 su prikazani dobiveni 3D modeli:

Slika 3.1: 3D model bez teksture



Slika 3.2: Dobiveni 3D modeli s teksturom prikazani u sceni

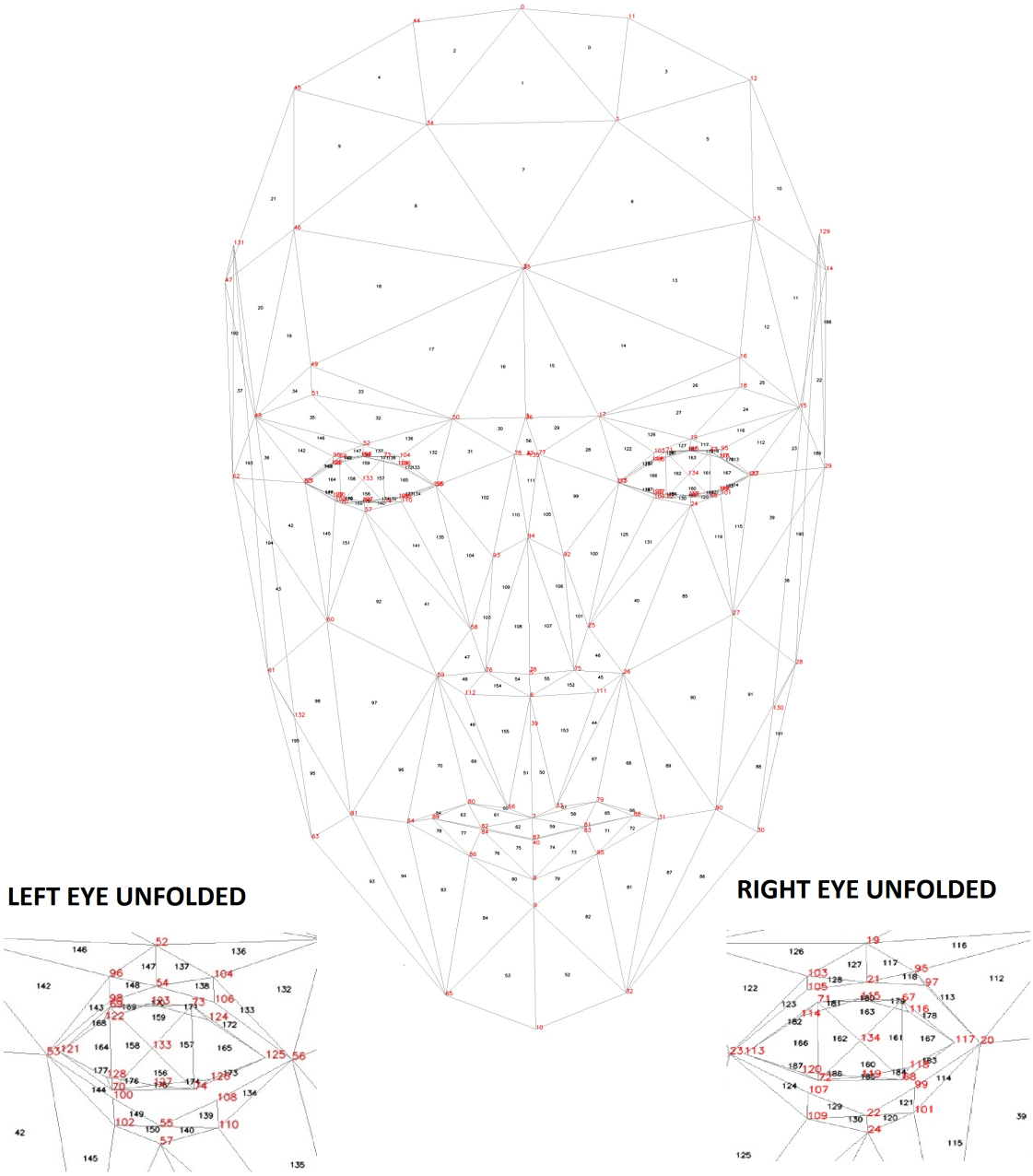
Animacija

3D model lica kojeg dobijemo pozivom funkcije `_getFaceModel` temelji se na modelu ljudskog lica Candide3 prikazanom na slici 3.3. Trenutni model se sastoji od 133 vrhova koji čine 190 poligona. Svaki izraz lica definiraju karakteristične vrijednosti vrhova. Kako bismo ostvarili animaciju potrebno je promijeniti vrijednosti u vrhovima. Od početnog do konačnog položaja dolazi se interpolacijom. Time se postepeno

mijenja početni 3D model u novi, izmijenjeni 3D model. Takav postupak se naziva morphing. U sklopu ovog rada implementirano je par jednostavnih animacija:

- pomicanjem vrhova oko usana (91, 64, 89, 88, 31 i 90) ostvaren je osmijeh
- pomicanjem vrhova koji definiraju obrve (49, 50, 51, 16, 17 i 18) ostvareno je mrštenje i ljutnja
- pomicanjem vrhova oko usana (88, 31 i 90) te vrhova obrve (16 i 18) ostvareno je podsmjehivanje s podignutom obrvom

Pomicanje od početne vrijednosti vrhova iz detekcije na slici pa do određene konačne vrijednosti ovih izraza lica ostvareno je linearnom interpolacijom pomoću funkcije `Mathf.Lerp(float from, float to, float t)`. U parametru `from` zadajemo početnu vrijednost, u parametru `to` konačnu vrijednost, a u parametru `t` korak interpolacije. Nakon svake promjene vrijednosti vrhova potrebno je osvježiti mesh ponovnim predavanjem polja izmijenjenih vrhova. Pojedini dobiveni izrazi prikazani su na slikama 3.4, 3.5 i 3.6:



Slika 3.3: Candide3 model ljudskog lica s brojevima vrhova i poligona

Slika 3.4: Neutralni izraz, osmijeh i mrštenje

Slika 3.5: Neutralni izraz, osmijeh, podsmjehivanje s podignutom obrvom

Slika 3.6: Neutralni izraz, osmijeh, ljutnja i podsmjehivanje

4. Zaključak

Postupak stvaranja virtualnih likova iz slika je vrlo kompleksan zadatak tijekom kojeg moramo u vidu imati mnoga ograničenja. Veliki broj slika nije prikladan za dobivanje 3D modela zbog loše pozicije i orijentacije glave kao i zbog udjela koje lice čini na slici. Nažalost, ovaj problem je teško rješiv jer alati detekcije nisu savršeni i ne mogu tako jednostavno procijeniti vrijednosti koje nije moguće preuzeti sa slike. No, na slikama koje odgovaraju zadanom formatu detekcija je uspješna i može vrlo precizno odrediti karakteristične točke lica. 3D model ovisi o rezultatima detekcije te njegova točnost također može biti narušena slikom koja ne odgovara zadanim uvjetima. 3D model lica još ovisi i o modelu koji se koristi - Candide3, opisan u ovom radu, je jednostavan model, ali dovoljno dobar za naše potrebe. Ako bismo koristili model s većim brojem definiranih vrhova i poligona, sigurno bismo dobili precizniji i realniji model, ali i računalu grafički zahtjevniji za prikazati i animirati. Animacija 3D modela je vjerojatno najzahtjevniji i najsloženiji dio povezan s virtualnim likovima. Simulacija ljudskih izraza lica zahtjeva detaljnu analizu i podatke o mišićima, odnosno točkama lica koje bismo trebali promijeniti za najispravnije rezultate. U ovom radu je implementirano par jednostavnih animacija koje služe kao demonstracija načina na koji se animacija ostvaruje. Pritom smo pomicali samo par najvažnijih vrhova za pojedini izraz, ali u stvarnosti je potrebno obuhvatiti puno više vrhova i pri tome voditi računa o rasponu vrijednosti koje vrhovi smiju poprimiti kako bismo izbjegli umjetni i previše deformirani prikaz koji ne nalikuje ljudskom. Naravno, u nekim prilikama je to i cilj (na primjer, karikature), što pokazuje da korištenjem animacije možemo prikazati gotovo sve što želimo.

LITERATURA

[1] Visage Technologies. *Visage SDK Documentation*. URL `http://www.visagetechnologies.com`.

Animacija lica na osnovu slike

Sažetak

Virtualni lik je 3D prikaz čovjekolikog lika na računalu. Brza i jednostavna proizvodnja virtualnih likova važna je zbog njihove sve češće primjene u mnogim područjima znanosti, ali i u svakodnevnom životu. U ovom radu prikazan je način dobivanja ovakvih likova iz fotografije pomoću alata Visage SDK. Detekcijom se prikupe podaci o karakterističnim točkama lica, vrhovima i poligonima koji se zatim koriste za prikaz odgovarajućeg 3D modela lica. Dobiveni 3D model je moguće animirati tako da prikazuje različite pokrete i izraze lica što postižemo tako da interpolacijom mijenjamo vrijednosti vrhovima 3D modela do odgovarajućih položaja vrhova.

Ključne riječi: virtualni lik, fotografija, detekcija lica, karakteristične točke lica, 3D model lica, animacija lica

Facial animation based on still image

Abstract

Virtual character is a 3D representation of humanoid character on the computer. Due to their frequent use in many branches of science, as well as in our everyday lives, it is important to have a quick and simple method for producing such characters. This thesis shows one possible way of virtual character production directly from still images by using Visage SDK. 3D data for rendering 3D face model is obtained by facial detection. That same model can be animated to display different facial expressions and movements by interpolating between specified vertex values.

Keywords: virtual character, photo, face detection, facial feature points, 3D face model, facial animation