ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Ανάλυση Αλγορίθμων Τριγωνοποίησης και Υλοποίηση Εφαρμογής Faceswap

Αποστολίδης Χρήστος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων:
Σταμούλης Γεώργιος

Συνεπιβλέποντες:
Καρασούλου Άννα
Δαδαλιάρης Αντώνιος

Λαμία, 2019

Περιεχόμενα

1. Εισαγωγή 3
2. Convex Hull 4
A. Ορισμοί 4
B. Αλγόριθμοι 5
B1. Gift wrapping, γνωστός και ως Jarvis march 5
B2. Graham scan 6
B3. Διαίρει και βασίλευε (Divide and conquer) 8
B4. Quickhull 8
B5. Monotone chain, a.k.a. Andrew's algorithm 8
B6. The ultimate planar convex hull algorithm 9
B7. Αλγόριθμος του Chan 9
3. Τριγωνοποίηση Delaunay 10
A. Delaunay 10
B. Διαγράμματα Voronoi 11
Γ. Σχέση μεταξύ τριγωνοποίησης Delaunay και διαγραμμάτων Voronoi 12
Δ. Ιδιότητες Delaunay 12
E. Flipping 13
ΣΤ. Αλγόριθμοι 15
ΣΤ1. Αλγόριθμοι Flip 16
ΣΤ2. Αυξητικός 16
ΣΤ3. Διαίρει και βασίλευε 17
ΣΤ4. Sweephull 17
4. Μια εφαρμογή Faceswapping 18
A. Η Βιβλιοθήκη OpenCV 18
B. Χρήση της OpenCV, numpy, dlib, στην εφαρμογή μας 18
Γ. Αναλυτικά 19
Δ. Γραφικό περιβάλλον και τελική εφαρμογή 27
Παράρτημα 1: Βιβλιογραφία & Σύνδεσμοι 44
Παράρτημα 2: Εικόνες 47

1. Εισαγωγή

Στόχος της πτυχιακής αυτής εργασίας είναι η ανάπτυξη μιας εφαρμογής που επιτυγχάνει αλλαγή προσώπου (faceswap) ανάμεσα σε δύο φωτογραφίες, καθώς και σε βίντεο. Στα κεφάλαια που ακολουθούν αναλύονται οι τεχνολογίες και τα μαθηματικά πίσω από τον τρόπο με τον οποίο επιτυγχάνεται η εναλλαγή αυτή.

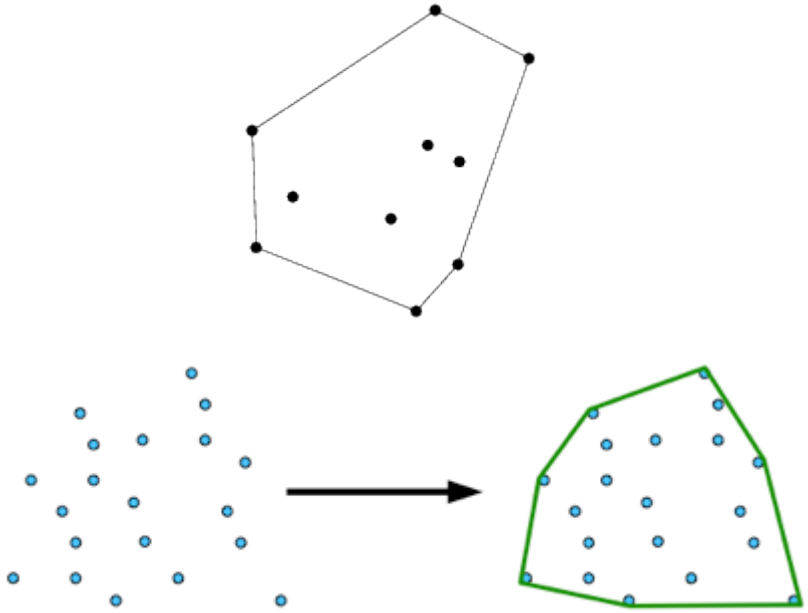
Πιο συγκεκριμένα στο κεφάλαιο 2 αναλύεται η έννοια του Convex Hull και αναφέρονται διάφορες εφαρμογές του αλλά και αλγόριθμοι, με περισσότερη έμφαση να δίνεται στους αλγόριθμους Gift Wrapping και Graham Scan.

Στο κεφάλαιο 3 αναλύονται οι έννοιες των τριγωνοποιήσεων Delaunay και των διαγραμμάτων Voronoi, καθώς και η στενή σχέση που έχουν μεταξύ τους. Αναφέρονται οι ιδιότητες που παρουσιάζουν οι τριγωνοποιήσεις αυτές και οι τεχνικές με τις οποίες μπορούμε να μετατρέψουμε τρίγωνα έτσι ώστε να εμπίπτουν σε αυτή την κατηγορία ακόμα και αν δεν παρουσιάζουν τις ιδιότητες Delaunay αρχικά. Ακόμα αναφέρονται διάφορες εφαρμογές των τριγωνοποιήσεων Delaunay και αναλύονται τέσσερις σημαντικοί αλγόριθμοι.

Στο τέταρτο κεφάλαιο βλέπουμε τις βιβλιοθήκες προγραμματισμού που μας παρέχουν υλοποιημένους του κατάλληλους αλγόριθμους Convex Hull, Delaunay κ.α., όπως η βιβλιοθήκη OpenCV και dlib. Στη συνέχεια παρουσιάζεται αναλυτικά ο κώδικας της ανεπτυγμένης εφαρμογής τόσο για το ίδιο το faceswap όσο και για την ανάπτυξη του γραφικού περιβάλλοντος με την βιβλιοθήκη Kivy.

Disclaimer: Οι ορισμοί, οι ιδιότητες και οι περιγραφές των αλγορίθμων που παρουσιάζονται σε αυτή την εργασία, έχουν παρθεί από τα αντίστοιχα paper και πηγές, τα οποία αναφέρονται αναλυτικά στο παράρτημα της βιβλιογραφίας.

2. Convex Hull



Εικόνα 2.1 & 2.2
Περιγραφή Convex Hull [2], [16]

Περιγραφικά, το convex hull (κυρτό κήτος) ή κυρτό κλείσιμο ενός συνόλου S σημείων σε ένα Ευκλείδειο επίπεδο ή σε ένα χώρο είναι το μικρότερο δυνατό κυρτό σύνολο που περιλαμβάνει όλα τα σημεία του S [1].

Α. Ορισμοί

Ένα σύνολο σημείων ονομάζεται κυρτό εάν περιέχει κάθε μία από τις γραμμές που ενώνουν ζεύγη των σημείων του. Το convex hull ενός συνόλου S μπορεί να οριστεί ως:

1. Το μοναδικό ελάχιστο κυρτό σύνολο που περιλαμβάνει το S.
2. Η τομή όλων των κυρτών συνόλων που περιλαμβάνουν το S.
3. Το σύνολο όλων των κυρτών συνδυασμών των σημείων του S.
4. Η ένωση όλων των συνιστωσών με συντεταγμένες στο S.

Για N σημεία $p_1, \ldots, p_N$, το convex hull C αποτιμάται από την μαθηματική έκφραση

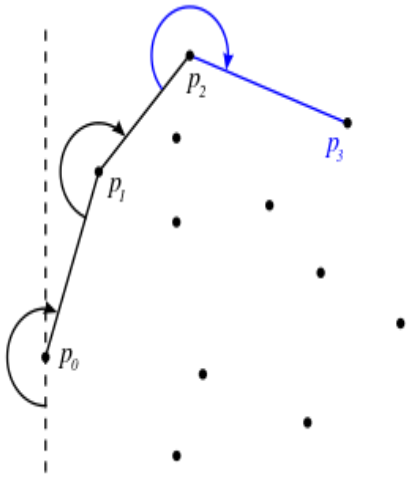
$$C \equiv \left\{ \sum_{j=1}^{N} \lambda_j \, p_j : \lambda_j \geq 0 \text{ for all } j \text{ and } \sum_{j=1}^{N} \lambda_j = 1 \right\}.$$ [2]

Β. Αλγόριθμοι

Οι αλγόριθμοι που κατασκευάζουν το convex hull διαφόρων αντικειμένων έχουν μεγάλο εύρος εφαρμογών στα μαθηματικά και την επιστήμη των υπολογιστών, όπως φαίνεται στη βιβλιογραφία [1], [8], [9]. Υπάρχουν πολλοί αλγόριθμοι για τον υπολογισμό του convex hull για ένα πεπερασμένο σύνολο σημείων, με πολλαπλές υπολογιστικές πολυπλοκότητες.

Μερικοί γνωστοί αλγόριθμοι για τον υπολογισμό του convex hull παρατίθενται στη συνέχεια. Η πολυπλοκότητα χρόνου του κάθε αλγόριθμου δίνεται σε συνάρτηση του αριθμού των σημείων εισόδου n και του αριθμού των σημείων στο πολύγωνο h. Στη χειρότερη περίπτωση το h ισούται με n.

Β1. Gift wrapping, γνωστός και ως Jarvis march



Εικόνα 2.3
Οπτικοποίηση αλγόριθμου Gift Wrapping [4]

Πρόκειται για έναν από τους απλούστερους αλγόριθμους τοποθέτησης και δεν είναι πολύ αποδοτικός σε χρόνο στην χειρότερη περίπτωση. Αναπτύχθηκε ξεχωριστά από τους Chand & Kapur το 1970 [18] και από τον R. A. Jarvis το 1973 [43]. Η μέση πολυπλοκότητα του ανέρχεται σε O(nh). Στην χειρότερη περίπτωση έχει πολυπλοκότητα $\Theta(n^2)$. Περισσότερες πληροφορίες υπάρχουν στις αναφορές [4], [5].

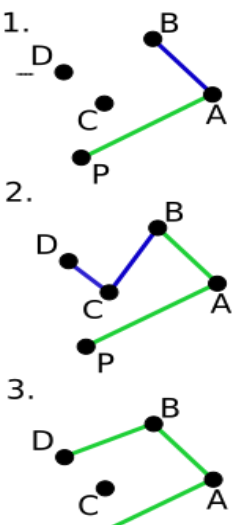
Ο αλγόριθμος gift wraping ξεκινάει με i=0 και ένα σημείο p_0 που είναι γνωστό ότι βρίσκεται εντός του convex hull, όπως για παράδειγμα το αριστερότερο σημείο, και διαλέγει ένα σημείο $p_{i+1,}$ έτσι ώστε όλα τα υπόλοιπα σημεία να βρίσκονται δεξιότερα της γραμμής $p_i\, p_{i+1}$. Το σημείο αυτό μπορεί να βρεθεί σε πολυπλοκότητα χρόνου O(n) συγκρίνοντας τις γωνίες που σχηματίζουν τα υπόλοιπα σημεία συναρτήσει του p_i , το οποίο θεωρείται ως αρχή των πολικών συντεταγμένων. Έπειτα το i γίνεται i=i+1, και επαναλαμβάνονται τα βήματα έως ότου $p_h=p_0$ όπου θα έχει βρεθεί το convex hull σε h βήματα.

```
jarvis(S)
 // S is the set of points
 pointOnHull = leftmost point in S // which is guaranteed to be part of the CH(S)
 i = 0
 repeat
   P[i] = pointOnHull
   endpoint = S[0]      // initial endpoint for a candidate edge on the hull
```

```
  for j from 1 to |S|
    if (endpoint == pointOnHull) or (S[j] is on left of line from P[i] to endpoint)
       endpoint = S[j]   // found greater left turn, update endpoint
  i = i+1
  pointOnHull = endpoint
until endpoint == P[0]      // wrapped around to first hull point
```

B2. Graham scan

Ένας ελαφρώς πιο εξελιγμένος αλλά πολύ πιο αποδοτικός αλγόριθμος, που δημοσιεύτηκε από τον Ronald Graham το 1972 [44]. Η μέση πολυπλοκότητα του ανέρχεται σε O(n log n). Αν τα σημεία είναι ήδη ταξινομημένα από μία από τις συντεταγμένες ή από τη γωνία σε ένα σταθερό διάνυσμα, τότε ο αλγόριθμος παίρνει τον χρόνο O (n), αναλυτικότερα περιγράφεται στις αναφορές [6], [7].



Εικόνα 2.4
Οπτικοποίηση αλγόριθμου Graham Scan [6]

Ως πρώτο βήμα βρίσκουμε, και ονομάζουμε P, το σημείο με τη χαμηλότερη συντεταγμένη y. Εάν η χαμηλότερη συντεταγμένη y υπάρχει σε περισσότερα από ένα σημεία του συνόλου, πρέπει να επιλεγεί το σημείο με τη χαμηλότερη συντεταγμένη x. Αυτό το βήμα έχει πολυπλοκότητα χρόνου O(n).

Έπειτα, το σύνολο των σημείων πρέπει να ταξινομηθεί με αύξουσα σειρά ως προς την γωνία που σχηματίζει το σημείο P με τον άξονα x. Οποιοσδήποτε αλγόριθμος ταξινόμησης γενικού σκοπού είναι κατάλληλος γι 'αυτό, για παράδειγμα ο heapsort (που είναι O (n log n)) [21].

Ο αλγόριθμος προχωρά εξετάζοντας κάθε ένα από τα σημεία της ταξινομημένης συστοιχίας σε σειρά. Για κάθε σημείο, καθορίζεται πρώτα κατά πόσον η μετακίνηση από τα δύο σημεία που προηγούνται αμέσως του σημείου αυτού αποτελεί μια αριστερή στροφή ή μια δεξιά στροφή. Στην περίπτωση της δεξιάς στροφής, το προ-τελευταίο σημείο δεν είναι μέρος του convex hull, αλλά βρίσκεται στο εσωτερικό του. Ο ίδιος προσδιορισμός γίνεται για το σύνολο του τελευταίου σημείου και για τα δύο σημεία που προηγούνται αμέσως του σημείου που βρέθηκε ότι ήταν μέσα στο hull και επαναλαμβάνεται μέχρις ότου συναντήσουμε ένα σύνολο "αριστερών στροφών", οπότε ο αλγόριθμος κινείται στο επόμενο σημείο στο σύνολο των σημείων στην ταξινομημένη διάταξη εκτός των σημείων που υπολογίστηκε ότι βρίσκονται μέσα στο hull. Δεν χρειάζεται να επανεξετάσουμε αυτά τα

σημεία. (Εάν σε οποιοδήποτε στάδιο τα τρία σημεία είναι στην ίδια ευθεία, μπορεί κάποιος να επιλέξει να τα απορρίψει ή να το αναφέρει.)

Ο προσδιορισμός του αν τα τρία σημεία αποτελούν "αριστερή στροφή" ή "δεξιά στροφή" δεν απαιτεί υπολογισμό της πραγματικής γωνίας μεταξύ των δύο γραμμικών τμημάτων και μπορεί να υπολογιστεί εύκολα. Για τρία σημεία

$$P_1 = (x_1, y_1)\ ,\ P_2 = (x_2, y_2) \text{ και } P_3 = (x_3, y_3)$$

υπολογίζεται η συντεταγμένη z του εξωτερικού γινομένου των δύο διανυσμάτων $\overrightarrow{P_1P_3}$, η οποία αποτιμάται από την έκφραση

$$(x_2\text{-}x_1)(\ y_3\text{-}y_1) - (y_2\text{-}y_1)(\ x_3\text{-}x_1)$$

Αν το αποτέλεσμα είναι 0, τα σημεία βρίσκονται στην ίδια ευθεία. Εάν είναι θετικό, τα τρία σημεία αποτελούν "αριστερή στροφή" ή αριστερόστροφη κατεύθυνση, διαφορετικά μια "δεξιά στροφή" ή δεξιόστροφη κατεύθυνση.

Αυτή η διαδικασία θα επιστρέψει τελικά στο σημείο από το οποίο ξεκίνησε, οπότε ο αλγόριθμος ολοκληρώνεται και η στοίβα περιέχει τώρα τα σημεία στο convex hull κατά την αντίθετη φορά των δεικτών του ρολογιού.

Μια ευκλείδεια διαμόρφωση της f στο $\mathbb{R}^2$ λέγεται ότι είναι κατευθυντικά-αντιστρέψιμη εάν υπάρχουν τρία μη συνευθειακά σημεία P ,Q και R στο $\mathbb{R}^2$ έτσι ώστε, παρατηρώντας το $\mathbb{R}^2$ από μία συγκεκριμένη πλευρά, μία από τις δύο σειρές PQR και $f(P)\ f(Q)\ f(R)$ να φαίνεται δεξιόστροφη (clockwise cw) και η άλλη αριστερόστροφη (counter-clockwise ccw) [29].

Ο παρακάτω κώδικας χρησιμοποιεί μια συνάρτηση ccw:

- ccw> 0 εάν τρία σημεία κάνουν μια αριστερή στροφή.
- ccw <0 εάν κάνουν μία δεξιά στροφή.
- ccw = 0 εάν βρίσκονται στην ίδια ευθεία.

Έπειτα το αποτέλεσμα αποθηκεύεται στη στοίβα.

let points **be** the list of points
let stack = empty_stack()

find the lowest y-coordinate and leftmost point, called P_0
sort points by polar angle with P_0, if several points have the same polar angle then only keep the farthest

for point **in** points:
  # pop the last point from the stack if we turn clockwise to reach this point

```
    while count stack > 1 and ccw(next_to_top(stack), top(stack), point) < 0:
        pop stack
    push point to stack
end
```

Αναφορικά μερικοί ακόμα αλγόριθμοι:

B3. Διαίρει και βασίλευε (Divide and conquer)

Δημοσιεύτηκε το 1977 από τους Preparata και Hong [34]. Η μέση πολυπλοκότητα του ανέρχεται σε O(n log n). Ο αλγόριθμος αυτός διασπά το σύνολο των σημείων σε δύο υποσύνολα και βρίσκει αναδρομικά, ξεχωριστά το convex hull για το καθένα από αυτά. Έπειτα τα συνενώνει με έναν αλγόριθμο merge.

Είναι βέλτιστος στην χειρότερη περίπτωση σε δύο ή τρείς διαστάσεις. Δεν είναι βέλτιστος στην μέση περίπτωση και σε υψηλότερες από τρείς διαστάσεις. Δεν είναι δυναμικός [33].

B4. Quickhull

Δημιουργήθηκε ανεξάρτητα το 1977 από τον W. Eddy και το 1978 από τον A. Bykat. Ο αλγόριθμος αυτός τραβάει μια γραμμή από το υψηλότερο στο χαμηλότερο σημείο του συνόλου και δημιουργεί τρίγωνα με τα σημεία που απέχουν περισσότερο από την γραμμή. Έπειτα διαλέγει τα σημεία που βρίσκονται εξωτερικά από τα τρίγωνα αυτά και επαναλαμβάνει τη διαδικασία έως ότου δεν υπάρχουν άλλα σημεία οπότε θα έχει βρεθεί και το convex hull. Ακριβώς όπως ο αλγόριθμος quicksort, έχει την αναμενόμενη πολυπλοκότητα του O (n log n), αλλά μπορεί να γίνει O (n^2) στη χειρότερη περίπτωση. Αντίθετα ωστόσο με τον quicksort, δεν υπάρχει προφανής τρόπος να μετατραπεί ο quickhull σε randomized αλγόριθμο [35]. Επομένως η μέση πολυπλοκότητα χρόνου δεν είναι εύκολο να υπολογιστεί [36].

B5. Monotone chain, a.k.a. Andrew's algorithm

Δημοσιεύθηκε το 1979 από τον A. M. Andrew [38]. Η μέση πολυπλοκότητα του ανέρχεται σε O(n log n). Ο αλγόριθμος μπορεί να θεωρηθεί ως μια παραλλαγή του Graham scan που ταξινομεί τα σημεία λεξικογραφικά με τις συντεταγμένες τους. Όταν η είσοδος είναι ήδη ταξινομημένη, ο αλγόριθμος παίρνει O (n) χρόνο [37].

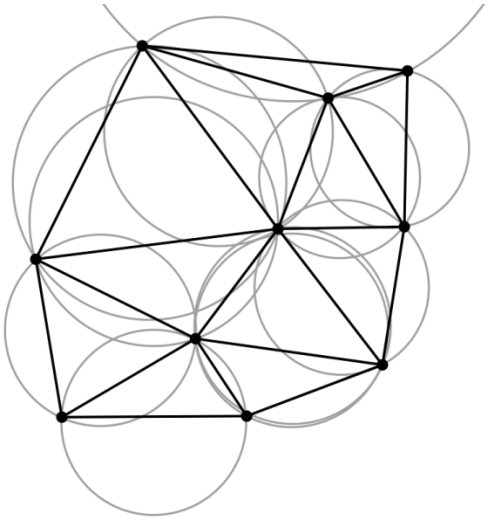
B6. The ultimate planar convex hull algorithm

Πρόκειται για τον πρώτο βέλτιστο output-sensitive αλγόριθμο, δηλαδή ο χρόνος του εξαρτάται τόσο από το μέγεθος της εισόδου όσο και από το μέγεθος της εξόδου [39]. Χρησιμοποιεί την τεχνική του γάμου-πριν-από-την-κατάκτηση (marriage-before-conquest), η οποία μοιάζει με αντίστροφη του divide and conquer. Η μέση πολυπλοκότητα του ανέρχεται σε $O(n \log h)$. Δημοσιεύθηκε από τους Kirkpatrick και Seidel το 1986 [40].

B7. Αλγόριθμος του Chan

Ένας απλούστερος βέλτιστος output-sensitive αλγόριθμος που δημιουργήθηκε από τον Chan το 1996 [42]. Η μέση πολυπλοκότητα του ανέρχεται σε $O(n \log h)$. Χρησιμοποιεί ένα συνδυασμό αλγορίθμων πολυπλοκότητας O(n log n), όπως για παράδειγμα τον Graham Scan, με τον αλγόριθμο Jarvis march (O(nh)), έτσι ώστε να μπορεί να επιτύχει την πολυπλοκότητα χρόνου O(n log h) [41].

3. Τριγωνοποίηση Delaunay



Εικόνα 3.1

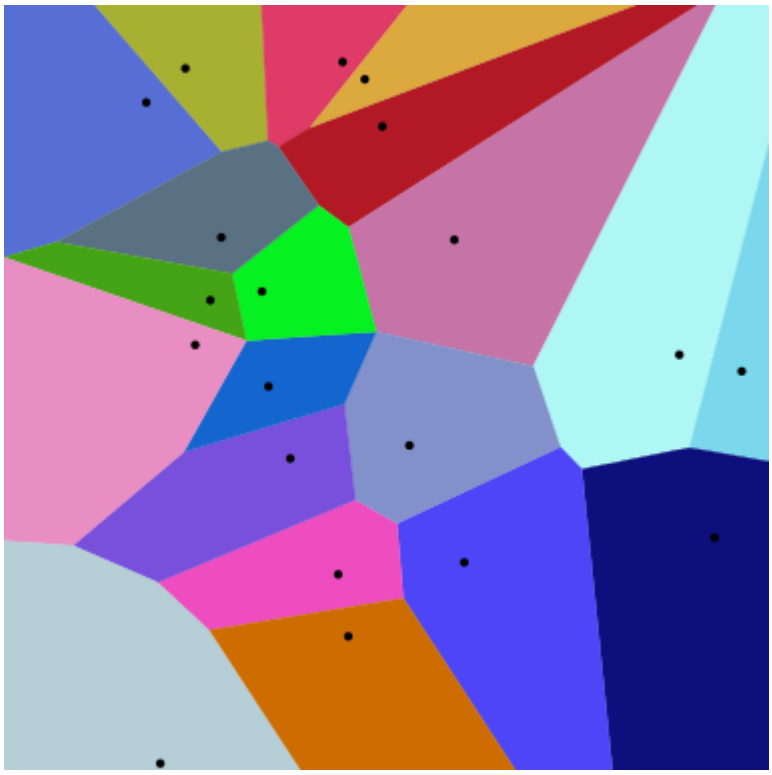
A Delaunay triangulation in the plane with circumcircles shown [10]

A. Delaunay

Στην υπολογιστική γεωμετρία, μια τριγωνοποίηση Delaunay για ένα δεδομένο σύνολο διακριτών σημείων P σε ένα επίπεδο, είναι μια τριγωνοποίηση DT(P) έτσι ώστε κανένα σημείο του συνόλου P να μην βρίσκεται εντός του κυκλικού περιγράμματος κανενός τριγώνου του DT(P), όπως φαίνεται στις αναφορές [9] και [10]. Οι τριγωνοποιήσεις Delaunay μεγιστοποιούν την ελάχιστη κλίση που μπορούν να έχουν οι γωνίες όλων των τριγώνων της τριγωνοποίησης και αποφεύγεται ο σχηματισμός τριγώνων που έχουν μία ή δύο πολύ οξείς γωνίες άρα ένα τρίγωνο μακρύ και λιγνό του οποίου οι ιδιότητες δεν βοηθάνε όταν χρειάζεται να γίνει κάποια μορφή επεξεργασίας σε αυτό.

Η τριγωνοποίηση αυτή πήρε το όνομα της από τον Boris Delaunay, Σοβιετικό μαθηματικό που το 1934 εργάστηκε και ανέπτυξε το μοντέλο.

Β. Διαγράμματα Voronoi



Εικόνα 3.2
Περιγραφή διαγράμματος Voronoi [13]

Στην υπολογιστική γεωμετρία, όπως φαίνεται στην αναφορά [13], ένα διάγραμμα Voronoi είναι ο διαχωρισμός ενός επιπέδου σε περιοχές που βασίζονται στην απόσταση από τα σημεία σε ένα συγκεκριμένο υποσύνολο του επιπέδου. Αυτό το σύνολο σημείων (που ονομάζονται σπόροι, τόποι ή γεννήτριες) καθορίζεται εκ των προτέρων και για κάθε γεννήτρια υπάρχει μια αντίστοιχη περιοχή που αποτελείται από όλα τα σημεία που βρίσκονται πιο κοντά στην εν λόγο γεννήτρια από οποιοδήποτε άλλη. Αυτές οι περιοχές ονομάζονται κελιά Voronoi.

Το όνομα Voronoi δόθηκε προς τιμήν του Georgy Voronoi, και ονομάζεται επίσης Voronoi tessellation, μια περιοχή Voronoi, ένα διαμέρισμα Voronoi, ή ένα dirichlet dessellation (μετά τον Peter Gustav Lejeune Dirichlet). Τα διαγράμματα Voronoi έχουν πρακτικές και θεωρητικές εφαρμογές σε μεγάλο αριθμό τομέων, κυρίως στην επιστήμη και την τεχνολογία, αλλά και στην εικαστική τέχνη [26] , [27]. Είναι επίσης γνωστά ως πολύγωνα Thiessen [28].

Γ. Σχέση μεταξύ τριγωνοποίησης Delaunay και διαγραμμάτων Voronoi

Η τριγωνοποίηση Delaunay ενός διακριτού συνόλου σημείων P σε γενική θέση αντιστοιχεί στο διπλό γράφημα του διαγράμματος Voronoi για το P, όπως φαίνεται στις αναφορές [10] , [11]. Οι περίκεντροι των τριγώνων Delaunay είναι οι κορυφές του διαγράμματος Voronoi. Στην περίπτωση των 2 διαστάσεων (2D), οι κορυφές Voronoi συνδέονται μέσω ακμών, που μπορούν να προκύψουν από τις σχέσεις γειτνίασης των τριγώνων Delaunay: Εάν δύο τρίγωνα μοιράζονται ένα άκρο στην τριγωνοπίηση Delaunay, οι περίκεντροί τους πρέπει να συνδεθούν με ένα άκρο στο Voronoi διάγραμμα .

Ειδικές περιπτώσεις στις οποίες η σχέση αυτή δεν ισχύει, ή είναι διφορούμενη, περιλαμβάνουν περιπτώσεις όπως:

Τρία ή περισσότερα συγγραμμικά σημεία, όπου οι περίκεντροι έχουν άπειρες ακτίνες.

Τέσσερα ή περισσότερα σημεία σε έναν τέλειο κύκλο, όπου η τριγωνοποίηση είναι διφορούμενη και όλοι οι περίκεντροι είναι πανομοιότυποι.

Δ. Ιδιότητες Delaunay

Σύμφωνα με τις αναφορές [10] , [12] , [15] μπορούμε να δούμε τις εξής ιδιότητες: Έστω n ο αριθμός των σημείων και ο αριθμός των διαστάσεων.

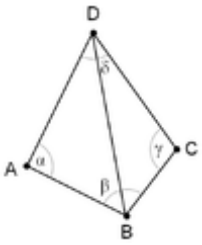
• Η ένωση όλων των εξωτερικών σημείων στην τριγωνοποίηση είναι το convex hull των σημείων.
• Μία τριγωνοποίηση Delaunay περιέχει $O(n^{\lceil d/2 \rceil})$ simplices.
• Στο επίπεδο δύο διαστάσεων (d = 2), αν υπάρχουν κορυφές b στο convex hull, τότε κάθε τριγωνισμός των σημείων έχει το πολύ 2 n - 2 - b τρίγωνα, συν μία εξωτερική όψη.
• Αν τα σημεία κατανέμονται σύμφωνα με μια διαδικασία Poisson στο επίπεδο με σταθερή ένταση, τότε κάθε κορυφή έχει κατά μέσο όρο έξι περιβάλλοντα τρίγωνα. Γενικότερα για την ίδια διαδικασία σε d διαστάσεις ο μέσος αριθμός γειτόνων είναι μια σταθερά που εξαρτάται μόνο από το d.
• Στο επίπεδο, η τριγωνοποίηση Delaunay μεγιστοποιεί την ελάχιστη γωνία. Σε σύγκριση με οποιοδήποτε άλλο τριγωνισμό των σημείων, η μικρότερη γωνία στην Delaunay είναι τουλάχιστον τόσο μεγάλη όσο η μικρότερη γωνία σε οποιαδήποτε άλλη τριγωνοποίηση.
• Ένας κύκλος που περιγράφει οποιοδήποτε τρίγωνο Delaunay δεν περιέχει άλλα σημεία εισόδου στο εσωτερικό του.
• Εάν ένας κύκλος που διέρχεται από δύο από τα σημεία εισόδου δεν περιέχει κανένα άλλο από αυτά στο εσωτερικό του, τότε το τμήμα που συνδέει τα δύο σημεία είναι μια ακμή μιας τριγωνοποίησης Delaunay των συγκεκριμένων σημείων.

• Ο πλησιέστερος γείτονας b σε οποιοδήποτε σημείο p βρίσκεται σε μια ακμή bp στη τριγωνοποίηση Delaunay, καθώς το πλησιέστερο γράφημα γείτονα αποτελεί υπογράφημα της τριγωνοποίησης Delaunay.
• Η τριγωνοποίηση Delaunay είναι ένα γράφημα t-spanner(geometric spanner). Στο επίπεδο (d = 2), η μικρότερη διαδρομή μεταξύ δύο κορυφών κατά μήκος των ακμών του Delaunay είναι γνωστό ότι δεν είναι μεγαλύτερη από $4\pi/3\sqrt{3}$ δηλαδή περίπου 2,418 φορές την Ευκλείδεια απόσταση μεταξύ τους.

E. Flipping

Από τις παραπάνω ιδιότητες, και σύμφωνα με τη βιβλιογραφία [9] , [10] ,προκύπτει ένα σημαντικό χαρακτηριστικό: Ανατρέχοντας σε δύο τρίγωνα ABD και BCD με την κοινή ακμή BD, αν το άθροισμα των γωνιών α και γ είναι μικρότερο ή ίσο με 180°, τα τρίγωνα πληρούν τη συνθήκη Delaunay .

Αυτό είναι μια σημαντική ιδιότητα επειδή επιτρέπει τη χρήση μιας τεχνικής αναστροφής. Εάν δύο τρίγωνα δεν πληρούν τη συνθήκη Delaunay, αλλάζοντας την κοινή ακμή BD για την κοινή ακμή AC παράγονται δύο τρίγωνα που πληρούν την κατάσταση Delaunay:

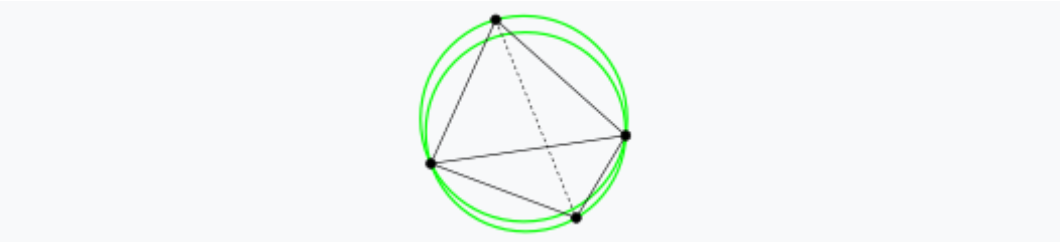


Εικόνα 3.3 *[10]*

This triangulation does not meet the Delaunay condition (the sum of α and γ is bigger than 180°).

Εικόνα 3.4 *[10]*

This pair of triangles does not meet the Delaunay condition (the circumcircle contains more than three points).



Εικόνα 3.5 *[10]*

Flipping the common edge produces a valid Delaunay triangulation for the four points.

Η πράξη αυτή ονομάζεται flip.

ΣΤ. Αλγόριθμοι

Πολλοί αλγόριθμοι για τον υπολογισμό των τριγωνισμών Delaunay βασίζονται σε γρήγορες λειτουργίες για την ανίχνευση του εάν ένα σημείο βρίσκεται μέσα στην περίκεντρο ενός τριγώνου και σε μια αποδοτική δομή δεδομένων για την αποθήκευση τριγώνων και ακμών [9] , [10].

In-circle test:
Για την ανίχνευση του εάν ένα σημείο βρίσκεται εντός του περίκεντρου που σχηματίζεται από ένα τρίγωνο τριών σημείων, αρκεί να υπολογιστεί η ορίζουσα που σχηματίζεται από τις συντεταγμένες των σημείων αυτών και του σημείου που εξετάζουμε [30] , [31].

Για παράδειγμα, σε δύο διαστάσεις, ένας τρόπος για να ανιχνευθεί εάν το σημείο D βρίσκεται στην περίκεντρο του A, B, C είναι να βρεθεί η ορίζουσα των συντεταγμένων των σημείων:

$$\begin{vmatrix} A_x & A_y & A_x^2+A_y^2 & 1 \\ B_x & B_y & B_x^2+B_y^2 & 1 \\ C_x & C_y & C_x^2+C_y^2 & 1 \\ D_x & D_y & D_x^2+D_y^2 & 1 \end{vmatrix} = \begin{vmatrix} A_x-D_x & A_y-D_y & (A_x^2-D_x^2)+(A_y^2-D_y^2) \\ B_x-D_x & B_y-D_y & (B_x^2-D_x^2)+(B_y^2-D_y^2) \\ C_x-D_x & C_y-D_y & (C_x^2-D_x^2)+(C_y^2-D_y^2) \end{vmatrix}$$

$$= \begin{vmatrix} A_x-D_x & A_y-D_y & (A_x-D_x)^2+(A_y-D_y)^2 \\ B_x-D_x & B_y-D_y & (B_x-D_x)^2+(B_y-D_y)^2 \\ C_x-D_x & C_y-D_y & (C_x-D_x)^2+(C_y-D_y)^2 \end{vmatrix} > 0$$

Όταν τα σημεία A, B και C ταξινομούνται κατά σειρά αντίθετη προς τη φορά των δεικτών του ρολογιού, η ορίζουσα που προκύπτει είναι θετική αν και μόνο εάν ο D βρίσκεται μέσα στον περίκεντρο. Στην περίπτωση που η ορίζουσα είναι αρνητική το σημείο βρίσκεται εκτός του κύκλου, ενώ στην περίπτωση που είναι ίση με 0 βρίσκεται πάνω στον περίκεντρο [10] , [31].

Επίσης, μπορούμε να λάβουμε μια εκτίμηση για την πολυπλοκότητα (bit complexity) του υπολογισμού της ορίζουσας θεωρώντας την ίδια σαν μια πολυωνυμική συνάρτηση στις συντεταγμένες εισόδου. Αν μια πολυωνυμική συνάρτηση έχει βαθμό d και οι μεταβλητές έχουν μέγεθος bit b, τότε θα χρειαστούν περίπου bd bits για τον υπολογισμό της συνάρτησης αυτής [32].

ΣΤ1. Αλγόριθμοι Flip

Αν ένα τρίγωνο δεν είναι Delaunay, μπορούμε να αναποδογυρίσουμε μία από τις ακμές του [10]. Αυτό μας οδηγεί σε έναν απλό αλγόριθμο: κατασκευάζουμε έναν οποιονδήποτε τριγωνισμό των σημείων και στη συνέχεια αναστρέφουμε τις άκρες έως ότου όλα τα τρίγωνα να είναι Delaunay. Αυτό μπορεί να πάρει $\Omega(n^2)$ περιστροφές ακμών [22]. Ο αλγόριθμος αυτός μπορεί να γενικευθεί σε τρεις και υψηλότερες διαστάσεις, όμως η σύγκλιση του δεν είναι εγγυημένη σε αυτές τις περιπτώσεις, καθώς εξαρτάται από τη συνάφεια του υποκείμενου flip γραφήματος: αυτό το γράφημα συνδέεται για δύο σειρές σημείων, αλλά μπορεί να αποσυνδεθεί σε υψηλότερες διαστάσεις.

ΣΤ2. Αυξητικός

Ο πιο απλός τρόπος για τον αποτελεσματικό υπολογισμό του τριγωνισμού Delaunay είναι να προσθέσουμε επαναληπτικά μία κορυφή κάθε φορά, επανατριγωνοποιώντας τα επηρεαζόμενα μέρη του γραφήματος [10]. Όταν προστίθεται μια κορυφή v, χωρίζουμε σε τρία τρίγωνα που περιέχουν την v και εφαρμόζουμε τον αλγόριθμο flip. Αυτό θα πάρει O(n) σε χρόνο εάν ψάξουμε όλα τα τρίγωνα ώστε να βρούμε εκείνο που περιέχει το v και στη συνέχεια να περιστρέψουμε όλα τα υπόλοιπα τρίγωνα. Τότε ο συνολικός χρόνος εκτέλεσης είναι O(n^2).

Όπως φαίνεται στις αναφορές [10] , [17] ,άν εισαγάγουμε κορυφές με τυχαία σειρά, αποδεικνύεται ότι κάθε εισαγωγή θα αναστρέψει, κατά μέσο όρο, μόνο O(1) τρίγωνα - αν και μερικές φορές μπορεί να αναστρέψει και πολλά περισσότερα [23]. Μπορούμε να αποθηκεύσουμε το ιστορικό των διαχωρισμών και των flips που εκτελούνται: κάθε τρίγωνο αποθηκεύει ένα δείκτη στα δύο ή τρία τρίγωνα που το αντικατέστησαν. Για να βρούμε το τρίγωνο που περιέχει το v, ξεκινάμε από ένα αρχικό τρίγωνο και ακολουθούμε τον δείκτη που δείχνει στο τρίγωνο που περιέχει το v, μέχρι να βρούμε κάποιο τρίγωνο που δεν έχει αντικατασταθεί ακόμα. Κατά μέσο όρο, αυτό θα πάρει χρόνο O (log n). Για όλες τις κορυφές, αυτό παίρνει O (n log n) χρόνο. Ενώ η τεχνική επεκτείνεται σε υψηλότερες διαστάσεις, ο χρόνος εκτέλεσης μπορεί να είναι εκθετικός στη διάσταση ακόμα και αν η τελική τριγωνοποίηση Delaunay είναι μικρού μεγέθους.

Δυστυχώς, οι αλγόριθμοι που βασίζονται σε αναστροφή είναι γενικά δύσκολο να παραλληλιστούν, δεδομένου ότι η προσθήκη κάποιου συγκεκριμένου σημείου να οδηγήσει σε μέχρι και O (n) διαδοχικές ανατροπές. Ο Guy Blelloch πρότεινε μια άλλη εκδοχή του αυξητικού αλγορίθμου βασισμένου σε rip-and-tent πρακτική και πολύ παραλληλισμένη με πολυγραμματοαριθμιτικό εύρος [45].

ΣΤ3. Διαίρει και βασίλευε

Ένας αλγόριθμος διαίρει και βασίλευε [10] για τριγωνισμούς σε δύο διαστάσεις αναπτύχθηκε από τους Lee και Schachter [3] και βελτιώθηκε από τους Guibas και Stolfi [24] και αργότερα από τον Dwyer [19]. Ο αλγόριθμος αυτός σχεδιάζει αναδρομικά μια γραμμή για να χωρίσει τις κορυφές σε δύο σύνολα. Ο τριγωνισμός Delaunay υπολογίζεται για κάθε σύνολο και στη συνέχεια τα δύο σύνολα συγχωνεύονται κατά μήκος της διαχωριστικής γραμμής. Χρησιμοποιώντας κάποιες έξυπνες τεχνικές, η λειτουργία συγχώνευσης μπορεί να γίνει σε χρόνο O (n), οπότε ο συνολικός χρόνος του αλγορίθμου είναι O (n log n).

Για ορισμένους τύπους σημειακών συνόλων, όπως η ομοιόμορφη τυχαία κατανομή, ο αναμενόμενος χρόνος μπορεί να μειωθεί στο O(n log log n) διατηρώντας παράλληλα τη χειρότερη περίπτωση απόδοσης.

Ο αλγόριθμος διαίρει και βασίλευε έχει αποδειχθεί ότι είναι η ταχύτερη τεχνική παραγωγής Delaunay.

ΣΤ4. Sweephull

Το Sweephull [25] είναι μια υβριδική τεχνική για τριγωνισμό 2D Delaunay που χρησιμοποιεί ένα ακτινικά πολλαπλασιαστικό σάρωθρο(sweep-hull), και έναν αλγόριθμο αναστροφής [10]. Το sweep-hull δημιουργείται διαδοχικά με την μετατόπιση ενός ακτινικά ταξινομημένου συνόλου 2D σημείων και τη σύνδεση των τριγώνων με το ορατό τμήμα του convex hull, το οποίο δίνει μια μη αλληλεπικαλυπτόμενη τριγωνοποίηση. Με τον τρόπο αυτό μπορεί να δημιουργηθεί ένα convex hull, εφόσον η σειρά των σημείων εγγυάται ότι κανένα σημείο δεν θα πέσει μέσα στο τρίγωνο. Στη συνέχεια, αυτό συνδυάζεται με ένα τελικό επαναληπτικό βήμα flipping.

4. Μια εφαρμογή Faceswapping

Α. Η Βιβλιοθήκη OpenCV

Η βιβλιοθήκη OpenCV (Open Source Computer Vision Library)[1] είναι μια βιβλιοθήκη λογισμικού ανοιχτού κώδικα με δυνατότητες computer vision και μηχανικής μάθησης. Περιλαμβάνει περισσότερους από 2500 βελτιστοποιημένους αλγόριθμους που μπορούν να χρησιμοποιηθούν για την «οπτική» αναγνώριση προσώπων, αντικειμένων, την κατηγοριοποίηση ανθρώπινων πράξεων μέσα από βίντεο, τον εντοπισμό κίνησης αντικειμένων και της ίδιας της κάμερας καθώς και να εξάγει τρισδιάστατα μοντέλα αντικειμένων, να συνενώσει φωτογραφίες και πολλές άλλες ενδιαφέρουσες λειτουργίες πάνω σε εικόνα και βίντεο.

Είναι ευρέως διαδεδομένη και χρησιμοποιείται τόσο από πολλές γνωστές τεχνολογικές και μη εταιρίες στον κόσμο, όσο και σε εφαρμογές από κρατικούς μηχανισμούς όπως έλεγχο της κυκλοφορίας και την ανάλυση πληροφοριών από βίντεο παρακολούθησης.

Ενώ η OpenCV είναι γραμμένη με τη γλώσσα προγραμματισμού C++, παρέχει διασύνδεση με περισσότερες γλώσσες όπως οι Python, Java και το περιβάλλον Matlab και υποστηρίζεται από τα λειτουργικά συστήματα Windows, Linux, Android και MacOS.

Β. Χρήση της OpenCV, numpy, dlib, στην εφαρμογή μας

Για την υλοποίηση μιας faceswapping εφαρμογής χρειαζόμαστε αλγορίθμους αναγνώρισης προσώπου και συγκεκριμένων σημείων πάνω σε μια εικόνα καθώς και αλγορίθμους που να υλοποιούν τόσο εύρεση του Convex Hull ενός προσώπου όσο και τριγωνοποιήσεις Delaunay καθώς και μεθόδους για την τροποποίηση συγκεκριμένων κομματιών του εκάστοτε προσώπου έτσι ώστε να μπορεί να ενταχθεί σε μία δεύτερη φωτογραφία αλλάζοντας το πρόσωπο της δεύτερης με όσο το δυνατόν λιγότερες διαφοροποιήσεις όσο αφορά την γεωμετρία του προσώπου, το μέγεθος και το ζουμ της φωτογραφίας, την γωνία καθώς και το χρώμα του δέρματος. Όλους αυτούς τους αλγορίθμους και τις μεθόδους μας τους παρέχει η βιβλιοθήκη OpenCV σε συνδυασμό με τις βιβλιοθήκες numpy[2] και dlib[3].

[1] https://opencv.org/
[2] https://numpy.org/
[3] http://dlib.net/

Γ. Αναλυτικά

Disclaimer: Ο κώδικας για το faceswapp των φωτογραφιών έχει αναπτυχθεί βήμα-βήμα με παρακολούθηση του δωρεάν tutorial για την συγκεκριμένη εφαρμογή που βρίσκεται στην ηλεκτρονική διεύθυνση https://pysource.com/2019/04/04/face-swapping-opencv-with-python-part-1/ ενώ ο κώδικας για το faceswapp πραγματικού χρόνου έχει ληφθεί εξ ολοκλήρου από την ίδια πηγή. Και στις δύο περιπτώσεις ο κώδικας έχει τροποποιηθεί κατάλληλα και έχει γίνει object oriented ώστε να καλύπτει τις ανάγκες της εφαρμογής μας.

Αρχικά στην Python εισάγουμε την βιβλιοθήκη στον κώδικα μας αφού την εγκαταστήσουμε με την εντολή **import cv2.**

Χρησιμοποιούμε τις μεθόδους της OpenCV:

```
img = cv2.imread(img1_path)
```

Με την μέθοδο imread() διαβάζουμε μια εικόνα και την καταχωρούμε στην μεταβλητή img.

```
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Με την cvtColor() και ως ορίσματα την εικόνα μας και τον ορισμό του χρώματος σε κλίμακα του γκρι κάνουμε ένα «ασπρόμαυρο» αντίγραφο της εικόνας μας έτσι ώστε να μπορούμε να δημιουργήσουμε μια «μάσκα», προς το παρόν κενή, με την επόμενη εντολή.

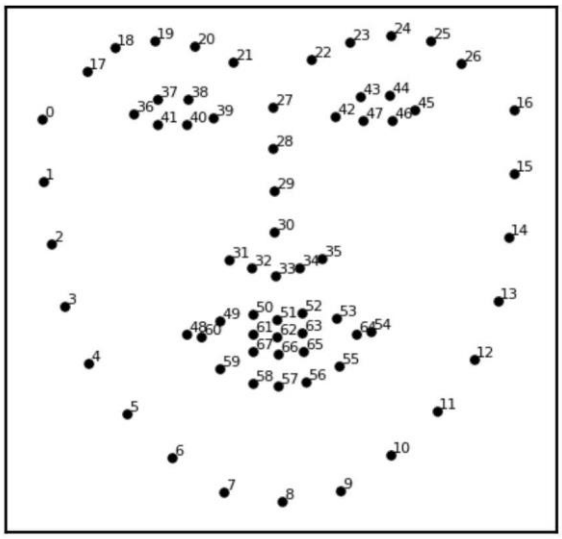
```
mask = np.zeros_like(img_gray)
```

// np η βιβλιοθήκη numpy.
//Η εντολή np.zeros() επιστρέφει έναν πίνακα δοσμένων διαστάσεων γεμάτο με μηδενικά.

```
img2 = cv2.imread(img2_path)
img2_gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
```

Έπειτα επαναλαμβάνουμε τα ίδια βήματα για μια δεύτερη εικόνα img2 και img2_grey χωρίς όμως να δημιουργήσουμε μάσκα για την δεύτερη εικόνα.

Για να φαίνεται ρεαλιστικό το αποτέλεσμα μιας εναλλαγής προσώπου θα πρέπει θα πρέπει το ένα πρόσωπο να στοιχηθεί επακριβώς πάνω στο δεύτερο έτσι ώστε να το καλύπτει πλήρως, άρα θα πρέπει να υπολογιστεί η γεωμετρία και των δύο προσώπων. Από τη στιγμή που πιθανότερα τα δύο πρόσωπα θα έχουν διαφορετική γεωμετρία θα πρέπει να τα διαμορφώσουμε κατάλληλα. Αυτό επιτυγχάνεται με τη βοήθεια της dlib και ένα αρχείο με 68 σημεία κλειδιά [14].



Εικόνα 4.1
Τα 68 σημαντικά σημεία που βρίσκονται σε κάθε πρόσωπο. [14]

Η αναγνώριση και εύρεση των 68 αυτών σημείων αποτελεί την εργασία των Vahid Kazemi και Josephine Sullivan [20].

Στο επόμενο βήμα με τη βοήθεια της dlib πρώτα εισάγουμε ένα έτοιμο αντικείμενο αναγνώρισης προσώπου και έπειτα εισάγουμε το αρχείο με τα 68 σημεία που έχουν νόημα να βρεθούν πάνω σε ένα πρόσωπο έτσι ώστε να μπορεί να γίνει η τριγωνοποίηση σε επόμενο στάδιο και δημιουργούμε μία κενή εικόνα με τα χαρακτηριστικά της πρώτης.

```
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat
")
height, width, channels = img2.shape
img2_new_face = np.zeros((height, width, channels), np.uint8)
```

// η μέθοδος img.shape επιστρέφει τις διαστάσεις τις καθώς και το channel.

//Channels are different dimensions of an image that holds value for every pixel - mainly - independently from the value of the other channels. So for example in case of an RGB image all pixels are represented as a Red, a Green, and a Blue values and there is no one summed value for a single pixel. [4]

```
faces = detector(img_gray)
        for face in faces:
            landmarks = predictor(img_gray, face)
            landmarks_points = []
            for n in range(0, 68):
```

[4] https://answers.opencv.org/question/7585/meaning-of-channels/

```
                x = landmarks.part(n).x
                y = landmarks.part(n).y
                landmarks_points.append((x, y))
```

Χρησιμοποιώντας τον detector βρίσκουμε το πρόσωπο στην γκρι φωτογραφία μας και πάνω σε αυτό τα σημεία landmark γεμίζοντας μία λίστα με τις συντεταγμένες της εικόνας όπου αυτά βρίσκονται.

```
            points = np.array(landmarks_points, np.int32)

            convexhull = cv2.convexHull(points)
            cv2.fillConvexPoly(mask, convexhull, 255)

            face_image_1 = cv2.bitwise_and(img, img, mask=mask)
```

Δημιουργούμε έναν πίνακα τύπου numpy array με στοιχεία τα σημεία που βρήκαμε και τον δίνουμε ως όρισμα στον έτοιμο αλγόριθμο της OpenCV για τον εντοπισμό του convex hull. Αφού εντοπίσουμε το convex hull, με όρισμα το ίδιο το Convex hull και τη μάσκα, αλλάζουμε το χρώμα της τελευταίας στο πολύγωνο που καλύπτεται απ το convex hull. Έπειτα με την εντολή bitwise_and και ορίσματα 2 φορές την πρώτη εικόνα και την μάσκα παίρνουμε το πρώτο πρόσωπο ακυρώνοντας απ την εικόνα κάθε τι βρίσκεται εκτός του convex hull, δηλαδή ότι δεν είναι το ίδιο το πρόσωπο.

Τριγωνοποίηση Delaunay

```
# Delaunay triangulation
            rect = cv2.boundingRect(convexhull)
            subdiv = cv2.Subdiv2D(rect)
            subdiv.insert(landmarks_points)
            triangles = subdiv.getTriangleList()
            triangles = np.array(triangles, dtype=np.int32)
            indexes_triangles = []
            for t in triangles:
                pt1 = (t[0], t[1])
                pt2 = (t[2], t[3])
                pt3 = (t[4], t[5])


                index_pt1 = np.where((points == pt1).all(axis=1))
                index_pt1 = self.extract_index_nparray(index_pt1)

                index_pt2 = np.where((points == pt2).all(axis=1))
                index_pt2 = self.extract_index_nparray(index_pt2)

                index_pt3 = np.where((points == pt3).all(axis=1))
```

```
                index_pt3 = self.extract_index_nparray(index_pt3)

                if index_pt1 is not None and index_pt2 is not None and
                   index_pt3 is not None:
                    triangle = [index_pt1, index_pt2, index_pt3]
                    indexes_triangles.append(triangle)
```

Δημιουργούμε ένα πολύγωνο με βάση το convex hull και το δίνουμε ως όρισμα στην εντολή cv2.Subdiv2D() από την οποία παίρνουμε ένα subdivision για την τριγωνοποίηση Delaunay. Αυτό μας παρέχει τις μεθόδους για να πάρουμε τα τρίγωνα που σχηματίζονται πάνω στο convex hull από τα σημεία που εισάγαμε νωρίτερα τα οποία αποθηκεύουμε και σε έναν nparray. Έπειτα για κάθε τρίγωνο βρίσκουμε τα σημεία που το σχηματίζουν και τα αποθηκεύουμε με τη μορφή δεικτών σε μία νέα λίστα indexes_triangles.

Το extract_index_nparray() είναι μία ξεχωριστή μέθοδος που λαμβάνει ως όρισμα τον nparray και επιστρέφει το index που χρειαζόμαστε:

```
    def extract_index_nparray(self, nparray):
        index = None
        for num in nparray[0]:
            index = num
            break
        return index
```

Για το πρόσωπο της δεύτερης εισάγουμε τα landmark σημεία, βρίσκουμε παρομοίως το convex hull και δημιουργούμε μια μάσκα που θα χρησιμοποιήσουμε αργότερα.

```
        # Face 2
        faces2 = detector(img2_gray)
        for face in faces2:
            landmarks = predictor(img2_gray, face)
            landmarks_points2 = []
            for n in range(0, 68):
                x = landmarks.part(n).x
                y = landmarks.part(n).y
                landmarks_points2.append((x, y))


            points2 = np.array(landmarks_points2, np.int32)
            convexhull2 = cv2.convexHull(points2)

        lines_space_mask = np.zeros_like(img_gray)
```

Το επόμενο κομμάτι κώδικα αφορά τον εντοπισμό των τριγώνων ένα-ένα και τον σχηματισμό τους πάνω στο εκάστοτε πρόσωπο με τη χρήση μίας μάσκας και για τα δύο πρόσωπα της πρώτης και της δεύτερης φωτογραφίας αντίστοιχα. Επίσης για κάθε τρίγωνο αποκόβεται το σημείο της εικόνας που περιέχει το τρίγωνο σε μία ξεχωριστή

εικόνα. Έτσι επιτυγχάνουμε την αφαίρεση των τριγώνων σε ξεχωριστά κομμάτια ώστε να μπορέσουν να τροποποιηθούν στο επόμενο στάδιο και στη συνέχεια να συγκολληθούν στην δεύτερη εικόνα με τρόπο που θα δούμε.

```
# Triangulation of both faces
        for triangle_index in indexes_triangles:
            # Triangulation of the first face
            tr1_pt1 = landmarks_points[triangle_index[0]]
            tr1_pt2 = landmarks_points[triangle_index[1]]
            tr1_pt3 = landmarks_points[triangle_index[2]]
            triangle1 = np.array([tr1_pt1, tr1_pt2, tr1_pt3], np.int32)


            rect1 = cv2.boundingRect(triangle1)
            (x, y, w, h) = rect1
            cropped_triangle = img[y: y + h, x: x + w]
            cropped_tr1_mask = np.zeros((h, w), np.uint8)


            points = np.array([[tr1_pt1[0] - x, tr1_pt1[1] - y],
                               [tr1_pt2[0] - x, tr1_pt2[1] - y],
                               [tr1_pt3[0] - x, tr1_pt3[1] -
 y]], np.int32)

            cv2.fillConvexPoly(cropped_tr1_mask, points, 255)

            # Lines space
            cv2.line(lines_space_mask, tr1_pt1, tr1_pt2, 255)
            cv2.line(lines_space_mask, tr1_pt2, tr1_pt3, 255)
            cv2.line(lines_space_mask, tr1_pt1, tr1_pt3, 255)
            lines_space = cv2.bitwise_and(img, img, mask=lines_space_ma
sk)

            # Triangulation of second face
            tr2_pt1 = landmarks_points2[triangle_index[0]]
            tr2_pt2 = landmarks_points2[triangle_index[1]]
            tr2_pt3 = landmarks_points2[triangle_index[2]]
            triangle2 = np.array([tr2_pt1, tr2_pt2, tr2_pt3], np.int32)


            rect2 = cv2.boundingRect(triangle2)
            (x, y, w, h) = rect2

            cropped_tr2_mask = np.zeros((h, w), np.uint8)

            points2 = np.array([[tr2_pt1[0] - x, tr2_pt1[1] - y],
                                [tr2_pt2[0] - x, tr2_pt2[1] - y],
                                [tr2_pt3[0] - x, tr2_pt3[1] -
 y]], np.int32)
```

```
            cv2.fillConvexPoly(cropped_tr2_mask, points2, 255)
```

Στο επόμενο στάδιο διαμορφώνουμε κάθε τρίγωνο έτσι ώστε να μοιάζει σε διαστάσεις με το αντίστοιχο τρίγωνο στα ίδια σημεία του προσώπου της δεύτερης φωτογραφίας. Αυτό επιτυγχάνεται με τις μεθόδους getAffineTransformation και warpAffine που μας παρέχει η OpenCV. Στη συνέχεια αντικαθιστούμε το παλιό με το πλέον ανανεωμένο τρίγωνο και το προσθέτουμε σε μία εικόνα στην οποία στο τέλος της επανάληψης θα σχηματιστεί το πρώτο πρόσωπο αλλά ανανεωμένο έτσι ώστε να ταιριάζει στο πρόσωπο της δεύτερης εικόνας.

```
# Warp triangles
            points = np.float32(points)
            points2 = np.float32(points2)
            M = cv2.getAffineTransform(points, points2)
            warped_triangle = cv2.warpAffine(cropped_triangle, M, (w, h
))
            warped_triangle = cv2.bitwise_and(warped_triangle, warped_t
riangle, mask=cropped_tr2_mask)

            # Reconstructing destination face
            img2_new_face_rect_area = img2_new_face[y: y + h, x: x + w]
            img2_new_face_rect_area_gray = cv2.cvtColor(img2_new_face_r
ect_area, cv2.COLOR_BGR2GRAY)
            _, mask_triangles_designed = cv2.threshold(img2_new_face_re
ct_area_gray, 1, 255, cv2.THRESH_BINARY_INV)
            warped_triangle = cv2.bitwise_and(warped_triangle, warped_t
riangle, mask=mask_triangles_designed)

            img2_new_face_rect_area = cv2.add(img2_new_face_rect_area,
warped_triangle)
            img2_new_face[y: y + h, x: x + w] = img2_new_face_rect_area
```

Σε τελικό στάδιο και έξω πλέον από την επανάληψη για τα τρίγωνα, αφού το νέο πρόσωπο έχει πλέον σχηματιστεί πάμε να το εφαρμόσουμε πάνω στην δεύτερη εικόνα. Αφαιρούμε πρώτα απ την δεύτερη εικόνα το αρχικό πρόσωπο και στη συνέχεια τοποθετούμε το καινούριο.

```
# Face swapped (putting 1st face into 2nd face)
        img2_face_mask = np.zeros_like(img2_gray)
        img2_head_mask = cv2.fillConvexPoly(img2_face_mask, convexhull2
, 255)
        img2_face_mask = cv2.bitwise_not(img2_head_mask)
```

```
        img2_head_noface = cv2.bitwise_and(img2, img2, mask=img2_face_m
ask)
        result = cv2.add(img2_head_noface, img2_new_face)
```

Πλέον έχει επιτευχθεί η αλλαγή προσώπου όμως υπάρχει ένα πρόβλημα. Η πρώτη φωτογραφία, άρα και το πρώτο πρόσωπο, το πιθανότερο είναι να έχει ληφθεί με διαφορετικές συνθήκες φωτισμού, σκιάς αλλά και το χρώμα του δέρματος να διαφέρει από το χρώμα του δέρματος του ατόμου της δεύτερης φωτογραφίας.

Εικόνα 4.2
Παράδειγμα μη seamless faceswap

Το πρόβλημα αυτό λύνεται εύκολα με τη χρήση της μεθόδου seamlessClone η οποία παραμετροποιεί τα χρώματα του νέου προσώπου έτσι ώστε να ταιριάζουν με το χρώμα του αρχικού προσώπου.

```
(x, y, w, h) = cv2.boundingRect(convexhull2)
        center_face2 = (int((x + x + w) / 2), int((y + y + h) / 2))

        seamlessclone = cv2.seamlessClone(result, img2, img2_head_mask,
 center_face2, cv2.NORMAL_CLONE)
```

Τέλος μπορούμε να εμφανίσουμε την εικόνα με την χρήση της εντολής cv2.imshow(“”. seamlessclone).

Εικόνα 4.3
Παράδειγμα seamless faceswap

Όλος ο παραπάνω κώδικας μπορεί να τροποποιηθεί κατάλληλα για την υλοποίηση faceswapp σε βίντεο σε πραγματικό χρόνο δεν θα αναλυθεί όμως αφού το συγκεκριμένο κομμάτι αφορά έτοιμο κώδικα που απλά έχει τροποποιηθεί κατάλληλα ώστε να είναι λειτουργικό με την εφαρμογή μας.

Εικόνα 4.4
Παράδειγμα video faceswap

Δ. Γραφικό περιβάλλον και τελική εφαρμογή

Το γραφικό περιβάλλον υλοποιήθηκε με τη χρήση της βιβλιοθήκης ανοιχτού κώδικα για την γλώσσα Python, Kivy. Η βιβλιοθήκη Kivy[5] παρέχει τις λειτουργίες για την ανάπτυξη μίας εφαρμογής όπως τη δημιουργία και παραμετροποίηση γραφικού περιβάλλοντος το οποίο μπορεί να εξαχθεί ως εκτελέσιμο σε διάφορες πλατφόρμες, όπως Windows, Linux, iOS, Android κλπ, διατηρώντας την ίδια εμφάνιση και λειτουργικότητα.

Η Kivy λειτουργεί με την object oriented λογική και χρησιμοποιεί τόσο Python για την υλοποίηση των λειτουργιών της εφαρμογής όσο και μια «γλώσσα», την Kv, με την οποία μπορούμε να γράψουμε το γραφικό μας περιβάλλον πιο απλά από το να το γράφαμε σε Python, διατηρώντας μια λογική εμφωλευμένων βρόγχων.

Το αρχείο Kv εισάγεται και τρέχει απ τον κώδικα Python ως εξής:

```
kv = Builder.load_file("faceswapptestkv.kv")
class FaceswappKivy(App):
    def build(self):
        return kv
```

Για την κλάση TestLayout() η οποία αντιπροσωπεύει την κεντρική οθόνη της εφαρμογής μας έχει γραφεί ο παρακάτω κώδικας σε Kv:

```
<TestLayout>:
    GridLayout:
        name: "main"
        cols: 1
        rows : 3
        size: root.width, root.height
        GridLayout:
            cols:3
            size_hint: 1 , .3
            Button:
                text: "?"
                size_hint: .08 , 1
                on_release: root.btnShowHint()

            Label:
                text: "FaceswApp"

            Button:
                text:"Video Faceswapp"
                background_normal : ''
                background_color: .93 , .81 , .7 , 1
```

[5] https://kivy.org/#home

```
            size_hint: .2 , 1
            on_release:
                app.root.current = "vidwindow"
                root.manager.transition.direction = "left"

    BoxLayout:
        orientation: 'horizontal'
        spacing : 10
        cols:2

        Image:
            id: img1
            source: 'bradley_cooper.jpg'
            size: root.width, root.height
        Button:
            text: "Swap Images"
            size_hint: .3 , .2
            center_y: self.parent.center_y
            on_release: root.swapImagesPlace()
        Image:
            id: img2
            source: 'jim_carrey.jpg'
            size: root.width, root.height

    GridLayout:
        cols:3
        size_hint: 1 , .3

        Button:
            text: "Insert Image 1"
            on_release: root.btnImg1()


        Button:
            text: "Insert Image 2"
            on_release: root.btnImg2()
        Button:
            text: "Faceswapp"
            on_release: root.btnFaceswapp()
```

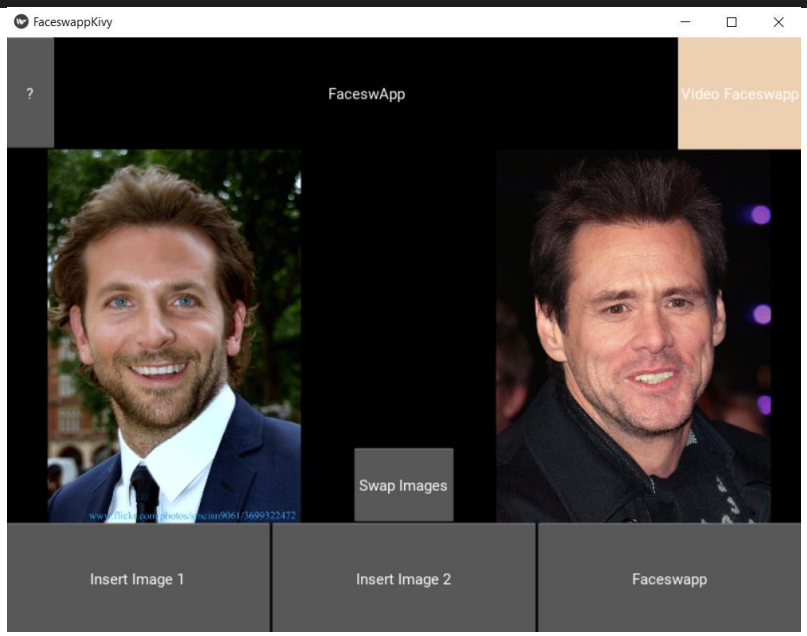
Μέσα στο αντικείμενο της TestLayout, το οποίο σε χαμηλότερες βαθμίδες ιεραρχίας μπορεί να αναφερθεί ως root, δημιουργούμε ένα GridLayout που θα συγκρατήσει όλα τα στοιχεία της οθόνης μας και ξεχωριστά για κάθε διαφορετικό κομμάτι της οθόνης διαφορετικά Layouts. Το πρώτο εξ αυτών περιέχει ένα κουμπί το οποίο καλεί την μέθοδο btnShowHint() της κλάσης μας και προβάλει ένα μήνυμα βοήθειας στον χρήστη, τον τίτλο της εφαρμογής μας καθώς και ένα κουμπί για αλλαγή οθόνης μέσω

ενός ScreenManager, για την μετάβαση στην λειτουργία faceswapp σε ζωντανό βίντεο. Το δεύτερο Layout περιλαμβάνει τα πεδία των δύο εικόνων που θα χρησιμοποιηθούν απ την εφαρμογή και ένα κουμπί για την εναλλαγή αυτών. Τέλος το τρίτο Layout περιέχει τρία κουμπιά που καλούν τις εξής μεθόδους: btnImg1(), btnImg2(), που ανοίγουν τις επιλογές για λήψη ή φόρτωση της αντίστοιχης εικόνας. Ανάλογα με το αν πατήσαμε το πρώτο η το δεύτερο κουμπί εικόνας στην αρχική οθόνη, το παράθυρο επιλογών θα ανοίξει με διαφορετικό τίτλο καθώς και με διαφορετική μεταβλητή για την αποθήκευση της εικόνας. Τέλος η μέθοδος btnFaceswapp() που καλεί τον κώδικα για την υλοποίηση του faceswapp.

Παρακάτω ο αντίστοιχος κώδικας Python της κλάσης TestLayout():

```
class TestLayout(Screen):
    img1_path = 'bradley_cooper.jpg'
    img2_path = 'jim_carrey.jpg'
    img_selector = None

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        global tL
        tL= self



    def swapImagesPlace(self):
        temp = self.img1_path
        self.img1_path = self.img2_path
        self.img2_path = temp
        self.ids['img1'].source = self.img1_path
        self.ids['img2'].source = self.img2_path
        self.ids.img1.reload()
        self.ids.img2.reload()

    def getImgSelector(self):
        return self.img_selector

    def setImgSelector(self,num):
        self.img_selector = num

    def updateImage(self):

        if self.img_selector == 1:
            self.ids['img1'].source = self.img1_path
            self.ids.img1.reload()
        elif self.img_selector ==2:
            self.ids['img2'].source = self.img2_path
            self.ids.img2.reload()
        pass
```

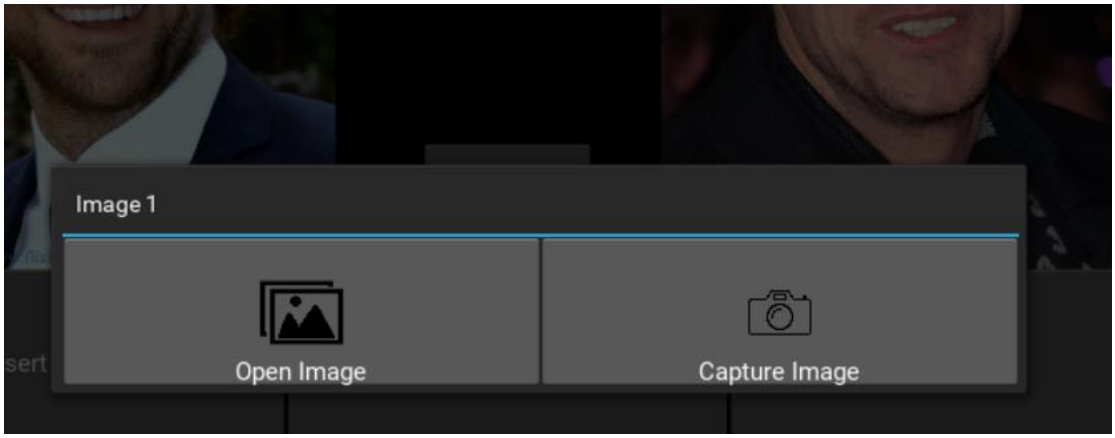
```
    def btnShowHint(self):
        HintPopup().openPP()

    def btnImg1(self):
        show =  PI()
        self.img_selector = 1
        pupWindow1 = Popup(title= "Image 1", content = show, size_hint=
(.75,.25), pos_hint={"x":0.15 , "top":.30})
        pupWindow1.open()


    def btnImg2(self):
        show =  PI()
        self.img_selector = 2
        pupWindow2 = Popup(title= "Image 2", content = show, size_hint=
(.75,.25), pos_hint={"x":0.15 , "top":.30})
        pupWindow2.open()


    def btnFaceswapp(self):

        f = fst()

        try:
            swappedImg = f.faceswapp(self.img1_path, self.img2_path)
            FaceswappedImage().openFsPp(swappedImg)
        except:
            pass
```



Η κλάση PI() μας παρέχει το αναδυόμενο παράθυρο με τις επιλογές λήψης ή φόρτωσης εικόνας, όπου απλά καλούνται είτε ο επιλογέας αρχείων TheFileChooser() ή ένα νέο αναδυόμενο παράθυρο με την κάμερα για την λήψη και προσωρινή αποθήκευση της φωτογραφίας.

```
<PI>:
    auto_dismiss: True
    title: ""
    separator_height: 0
    cols:1
    BoxLayout:
        orientation: "horizontal"
        cols:2
        Button:
            text: "Open Image"
            text_size: self.size
            halign: "center"
            valign: 'bottom'
            size_hint: 1 , 1
            on_release: root.openFcPopup()

            Image:
                source: "gallery.png"
                center_x: self.parent.center_x
                center_y: self.parent.center_y
                size: 50, 50
                allow_strech: True


        Button:
            text: "Capture Image"
            text_size: self.size
            halign: "center"
            valign: 'bottom'
            size_hint: 1 , 1
            on_release: root.openCamPopup()


            Image:
                source: "camera.png"
                center_x: self.parent.center_x
                center_y: self.parent.center_y
                size: 50, 50
                allow_strech: True
```

```
class PI(GridLayout):

    def openFcPopup(self):
        TheFileChooser().openPP("load",None)

    def openCamPopup(self):
        CameraClick().openCamPP()
```



Η κλάση TheFilechooser() καλεί τον έτοιμο επιλογέα αρχείων FileChooserIconView της Kivy και εκτελεί της λειτουργίες τόσο της φόρτωσης αρχείου όσο και της αποθήκευσης που θα δούμε σε επόμενο επίπεδο. Την επιλογή λειτουργίας την λαμβάνει ως όρισμα η μέθοδος που ανοίγει το παράθυρο, ανάλογα από το αν ανοίγει από την PI() ή από το αναδυόμενο παράθυρο που θα δούμε στην πορεία.

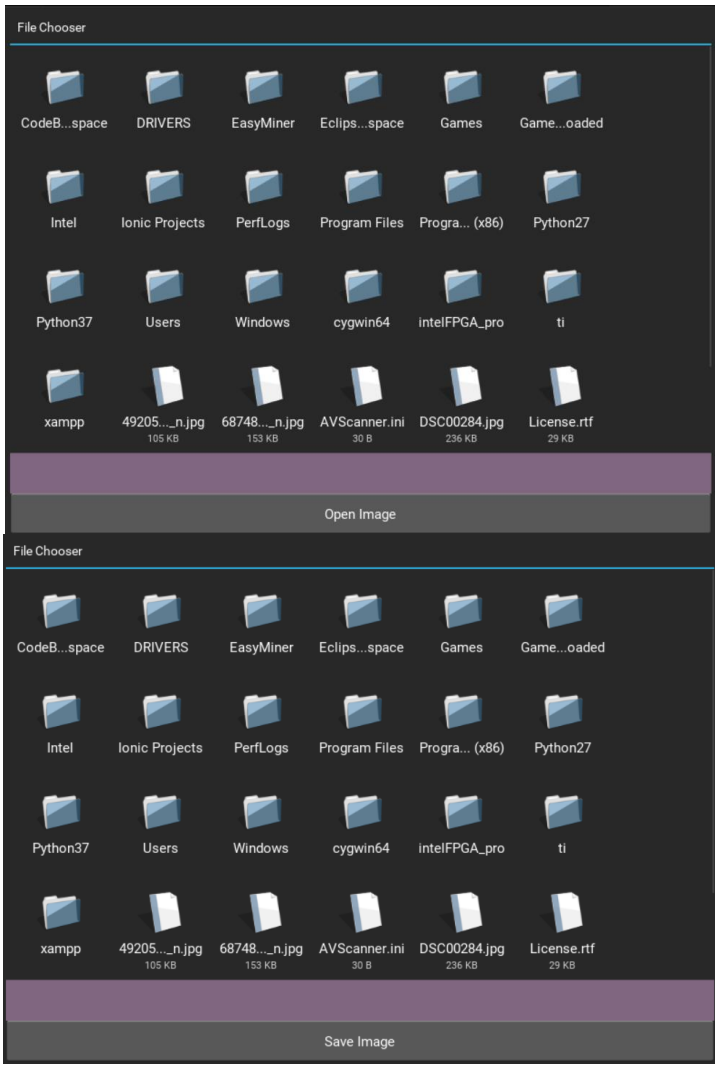
```
<TheFileChooser>:
    fciv: fciv
    label: label
    cl_fc: cl_fc

    name: "fchooser"
    orientation: 'vertical'
    FileChooserIconView:
        id: fciv
        on_selection: root.select(*args)

    Label:
        id: label
        size_hint_y: .1
        canvas.before:
            Color:
                rgb: .5,.4,.5
            Rectangle:
                pos: self.pos
                size: self.size
    Button:
```

```
        id: cl_fc
        size_hint_y: .1
        text: "Open Image"
        on_release:
            root.doStuff()
```

```
class TheFileChooser(BoxLayout):
    label = ObjectProperty(None)
    fciv = ObjectProperty(None)
    cl_fc = ObjectProperty(None)

    def openPP(self,function,img):
        if function == "load":
            self.cl_fc.text = "Open Image"
        elif function == "save":
            self.cl_fc.text = "Save Image"
            self.final_image = img
        self.show = self
        self.popUpFch = Popup(title="File Chooser", content = self.show
)
        self.popUpFch.open()
    def closePP(self):
        self.popUpFch.dismiss()


    def select(self, *args):
        try:
            self.label = args[1][0]
            self.label.text = self.label

        except: pass
    def doStuff(self):
        if self.cl_fc.text == "Open Image":
            try:
                if tL.getImgSelector() == 1:
                    tL.img1_path= self.label
                    tL.updateImage()
                elif tL.getImgSelector() == 2:
                    tL.img2_path= self.label
                    tL.updateImage()
                elif tL.getImgSelector() == 3:
                    vW.img_path = self.label
                    vW.updateImage()
                self.closePP()
            except:
                self.closePP()
```

```
        elif self.cl_fc.text == "Save Image":
            try:
                timestr = time.strftime("%Y%m%d_%H%M%S")
                shutil.copy2('swapped.jpg',self.fciv.path + "/IMG_{}.jp
g".format(timestr))
                print("Succesfully Saved")
                self.closePP()
            except:
                print("Something went wrong")
                self.closePP()
```

Ανάλογα με την επιλογή φόρτωση ή αποθήκευση λαμβάνουμε το path της εικόνας που θα φορτώσουμε από την μεταβλητή label ή αποθηκεύουμε την εικόνα μας στο path που λαμβάνουμε από τη μέθοδο path που παρέχει ο filechooser.

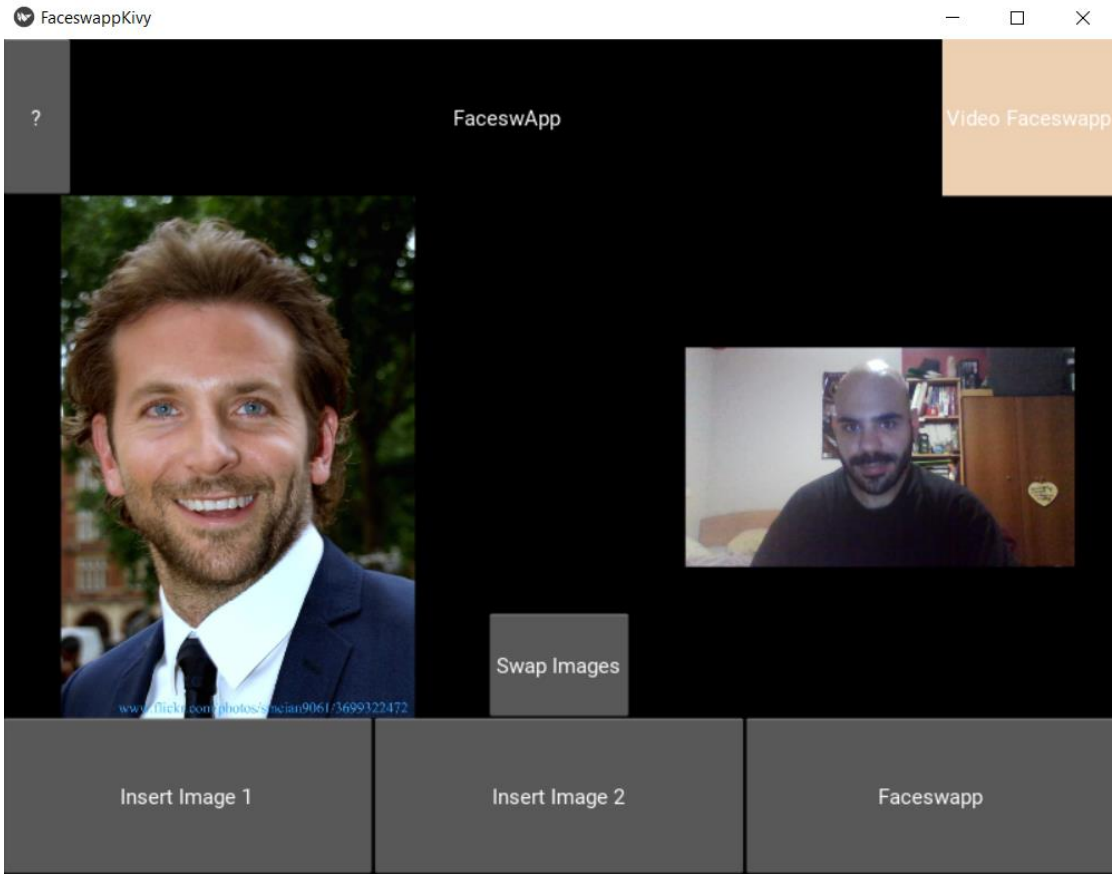


Η κλάση CameraClick() είναι η κλάση η οποία ανοίγει ένα αναδυόμενο παράθυρο και μέσα σε αυτή περιέχεται εικόνα από την προεπιλεγμένη κάμερα της εκάστοτε συσκευής και ένα κουμπί το οποίο καλεί την μέθοδο capture() η οποία απαθανατίζει και αποθηκεύει την εικόνα της κάμερας ανάλογα με το αν έχουμε επιλέξει να προσθέσουμε την πρώτη, τη δεύτερη ή την εικόνα στην οθόνη του βίντεο faceswap, και στην συνέχεια ανανεώνει το γραφικό περιβάλλον της εκάστοτε οθόνης με την νέα εικόνα.

```
<CameraClick>:

    auto_dismiss: False
    title: ""
    separator_height: 0
    BoxLayout:

        orientation: 'vertical'
        Camera:
            id: camera
            resolution: (640, 480)
            play: True


        Button:
            text: 'Capture'
            size_hint_y: None
            height: '48dp'
            on_press:
                root.capture()
```

```
class CameraClick(BoxLayout):

    def openCamPP(self):
        self.show = self
        self.popUpWindow1 =  Popup(title= "Camera", content = self.show
)
        self.popUpWindow1.open()
    def closeCamPP(self):
        self.popUpWindow1.dismiss()

    def updateImageCam(self):
        if tL.getImgSelector() == 1:
            tL.img1_path= "IMG1Captured.jpg"
            tL.updateImage()
        elif tL.getImgSelector() == 2:
            tL.img2_path= "IMG2Captured.jpg"
            tL.updateImage()
```

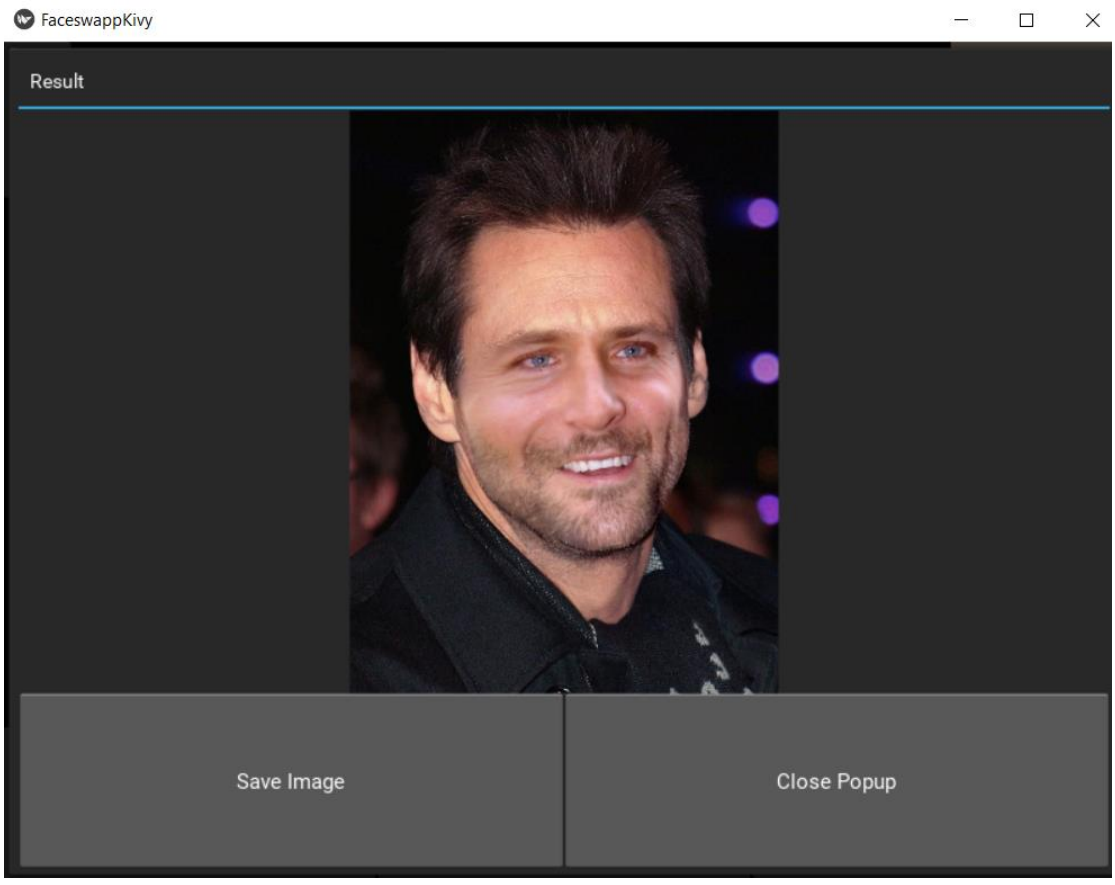
```
        elif tL.getImgSelector() == 3:
            vW.img_path= "IMG3Captured.jpg"
            vW.updateImage()

        pass


    def capture(self):
        '''
        Function to capture the image.
        '''
        camera = self.ids['camera']
        if tL.getImgSelector() == 1:
            camera.export_to_png("IMG1Captured.jpg")
        elif tL.getImgSelector() == 2:
            camera.export_to_png("IMG2Captured.jpg")
        elif tL.getImgSelector() == 3:
            camera.export_to_png("IMG3Captured.jpg")

        camera.play = False
        camera.stopped = True


        print("Captured")


        self.updateImageCam()
        self.closeCamPP()
```



Η μέθοδος FaceswappedImage() αποτελεί ένα αναδυόμενο παράθυρο το οποίο περιέχει την εικόνα-αποτέλεσμα του faceswapp και τις επιλογές αποθήκευσης της εικόνας με την επιλογή save στον επιλογέα αρχείων, ή την επιλογή να κλείσουμε απλά το παράθυρο. Η τελική εικόνα φορτώνεται ως όρισμα από την κλήση της μεθόδου openFsPp() στην TestLayout(), η οποία εικόνα έχει ληφθεί ως επιστροφή της τροποποιημένης κλάσης faceswapp του κώδικα που αναλύσαμε στο προηγούμενο υποκεφάλαιο.

```
<FaceswappedImage>:
    orientation: "vertical"
    Image:
        id: swappedImg
        source: root.fswappedImage
    BoxLayout:
        size_hint: 1 , .3
        orientation: 'horizontal'
        Button:
            text: "Save Image"
            on_release: root.saveImage()

        Button:
            text: "Close Popup"
            on_release: root.closeFsPp()
```

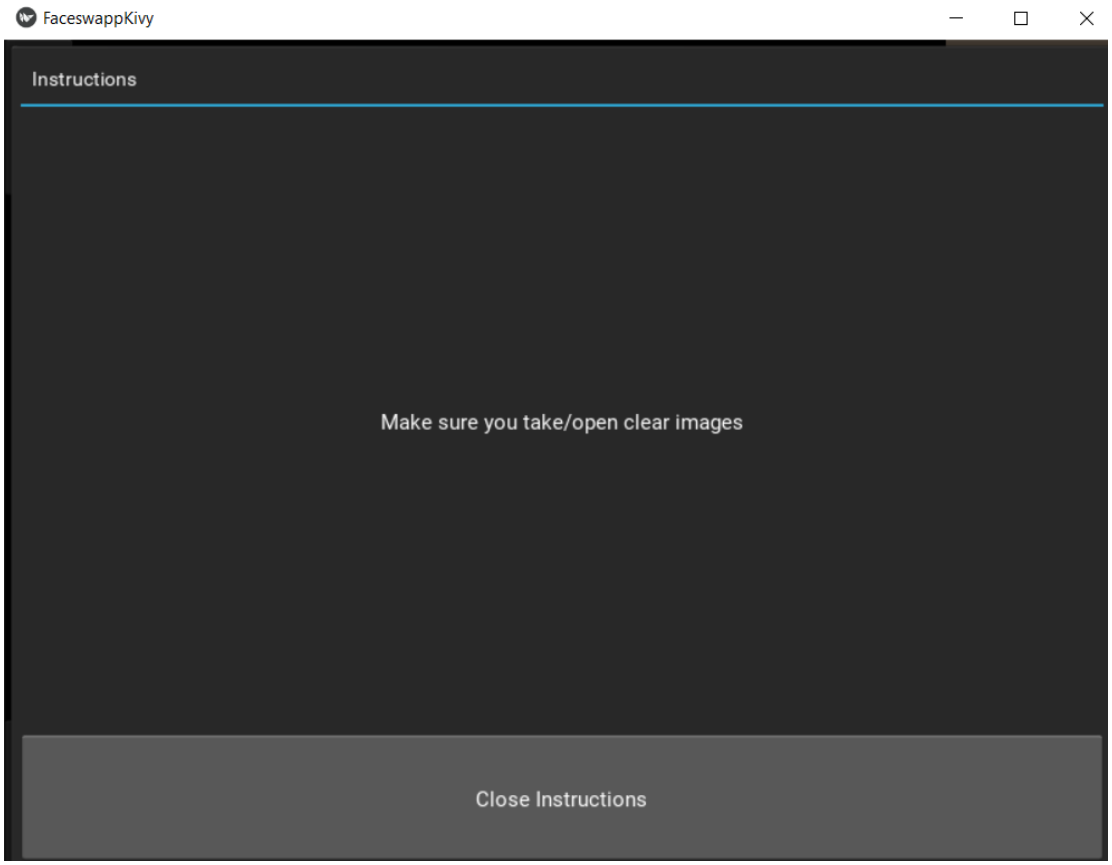
```
class FaceswappedImage(BoxLayout):
    fswappedImage = None

    def openFsPp(self, img):
        self.fswappedImage = img
        self.ids['swappedImg'].source = self.fswappedImage
        self.ids['swappedImg'].reload()

        self.pup = Popup(title="Result",content=self)
        self.pup.open()
        pass
    def closeFsPp(self):
        self.pup.dismiss()
        pass
    def saveImage(self):
        TheFileChooser().openPP("save",self.fswappedImage)
        pass
    pass
```



Η κλάση HintPopup() περιέχει το αναδυόμενο παράθυρο με την βοήθεια προς τον χρήστη και υπάρχει μόνο η επιλογή να κλείσουμε το ίδιο παράθυρο.

```
<HintPopup>:
    orientation: 'vertical'
    Label:
        text: "Make sure you take/open clear images"
    Button:
        text: "Close Instructions"
        size_hint: 1 , .2
        on_release: root.closePP()
```

```
class HintPopup(BoxLayout):
    def openPP(self):
        self.popup = Popup(title= "Instructions", content = self)
        self.popup.open()
    def closePP(self):
        self.popup.dismiss()
```



Τέλος για την μετάβαση στην οθόνη του βίντεο χρειαζόμαστε τον ScreenManager της βιβλιοθήκης Kivy καθώς και κάθε κλάση οθόνης που θέλουμε να χρησιμοποιήσουμε να κληρονομεί από την κλάση Screen.

```
ScreenManager:
    TestLayout:
        name: "main"
    VideoWindow:
        name: "vidwindow"
```

Έτσι με το πάτημα του κουμπιού VideoFaceswapp μεταφερόμαστε στην οθόνη VideoWindow και αντιστρόφως.

```
Button:
                text:"Video Faceswapp"
                background_normal : ''
                background_color: .93 , .81 , .7 , 1
                size_hint: .2 , 1
                on_release:
                    app.root.current = "vidwindow"
                    root.manager.transition.direction = "left"
```

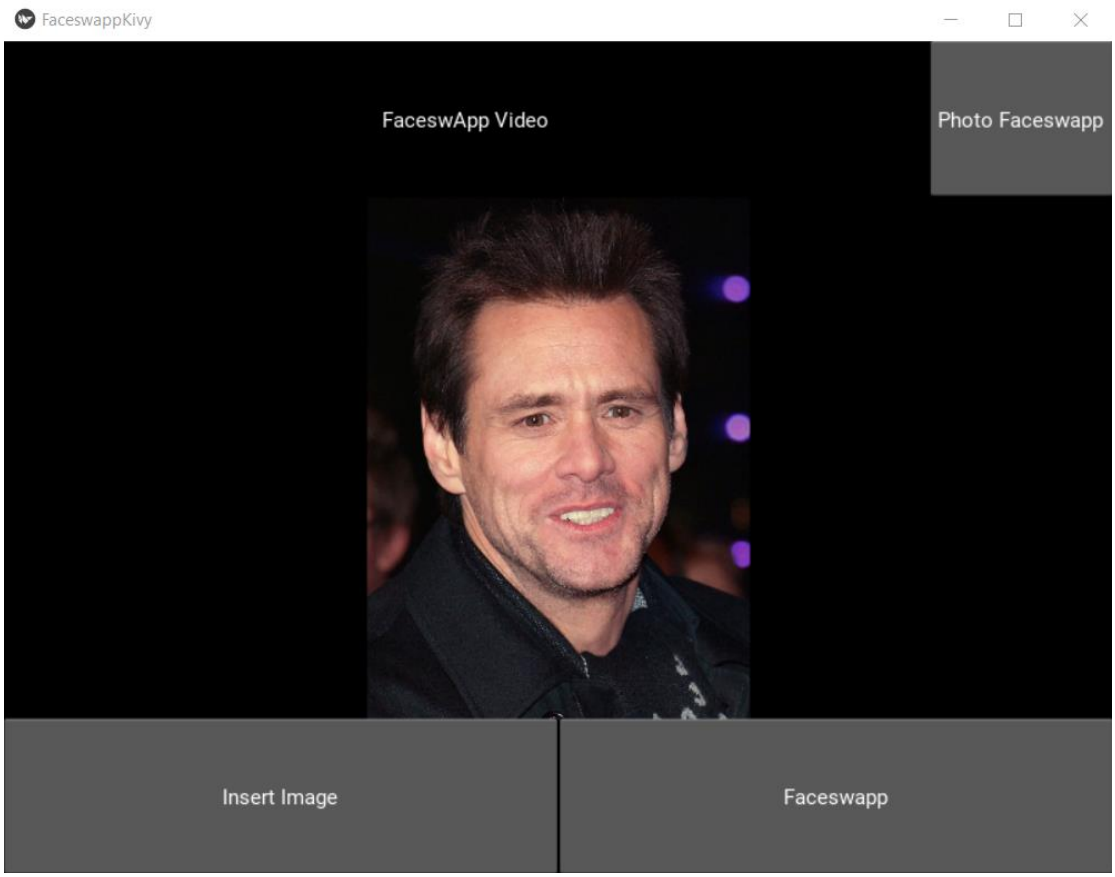
Η κλάση VideoWindow() αποτελεί την δεύτερη οθόνη μας και περιέχει αντίστοιχα τον τίτλο της εφαρμογής για βίντεο, μία εικόνα η οποία θα προστεθεί στην εικόνα πραγματικού χρόνου που θα πάρουμε από την κάμερα, ένα κουμπί για την επιλογή ή λήψη της εικόνας και το κουμπί Faceswapp που ανοίγει το τελευταίο αναδυόμενο παράθυρο το οποίο όμως τρέχει εκτός του περιβάλλοντος της Kivy μέσα από συναρτήσεις της βιβλιοθήκης OpenCV.

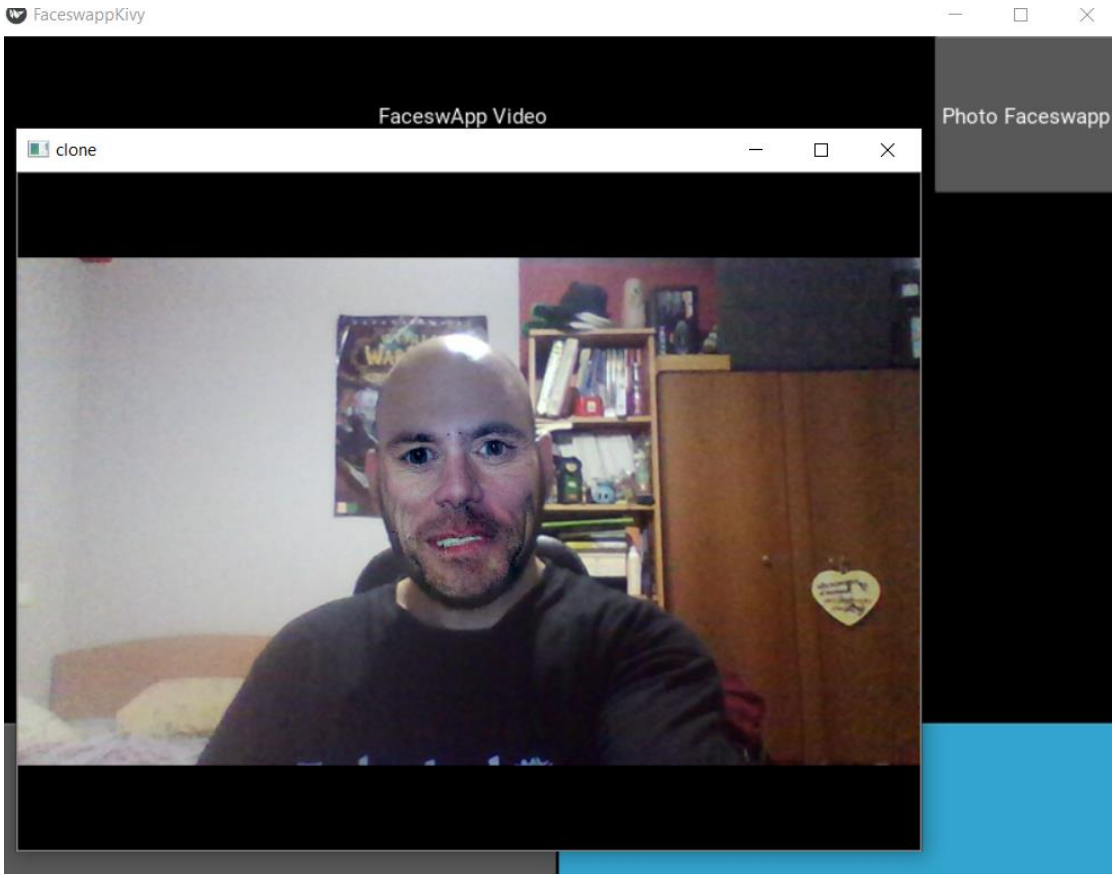
```
<VideoWindow>:
    GridLayout:
        name: "vidwindow"

        cols: 1
        rows : 3
        size: root.width, root.height

        GridLayout:
            cols:2
            size_hint: 1 , .3

            Label:
                text: "FaceswApp Video"

            Button:
```

```
                text:"Photo Faceswapp"
                size_hint: .2 , 1
                on_release:
                    app.root.current = "main"
                    root.manager.transition.direction = "right"

        BoxLayout:
            orientation: 'horizontal'
            spacing : 10
            cols:1

            Image:
                id: img_vid
                source: root.img_path
                size: root.width, root.height



        GridLayout:
            cols:2
            size_hint: 1 , .3

            Button:
                text: "Insert Image"
                on_release: root.btnImg()



            Button:
                text: "Faceswapp"
                on_release: root.vidFaceswapp()
```

```
class VideoWindow(Screen):
    img_path = 'jim_carrey.jpg'

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        global vW
        vW= self

    def updateImage(self):
        self.ids['img_vid'].source = self.img_path
        self.ids.img_vid.reload()

    def btnImg(self):
        show =  PI()
        tL.setImgSelector(3)
```

```
        pupWindow3 = Popup(title= "Image", content = show, size_hint=(.
75,.25), pos_hint={"x":0.15 , "top":.30})
        pupWindow3.open()

    def vidFaceswapp(self):
        vf = vfs()

        try:
            swappedTexture = vf.video_faceswapp(self.img_path)
            pass
        except:
            print("Something Went Wrong")
            pass
```





Ο κώδικας της εφαρμογής βρίσκεται στην ηλεκτρονική διεύθυνση:
https://github.com/ChrisApostolidis/Faceswaping-App-Using-Kivy-And-OpenCV .

Παράρτημα 1: Βιβλιογραφία & Σύνδεσμοι

[1] Wikipedia. "Convex Hull".
https://en.wikipedia.org/wiki/Convex_hull

[2] Weisstein, E. W. "*Convex Hul"l.* From MathWorld--A Wolfram Web Resource.
http://mathworld.wolfram.com/ConvexHull.html

[3] D. T. Lee & B. J. Schachter. "Two Algorithms for Constructing a Delaunay Triangulation". International Journal of Computer & Information Sciences. June 1980, Volume 9, Issue 3.

[4] Wikipedia. "*Gift Wrapping".*
https://en.wikipedia.org/wiki/Gift_wrapping_algorithm

[5] GeeksforGeeks. "*Convex Hull | Set 1 (Jarvis's Algorithm or Wrapping)".*
https://www.geeksforgeeks.org/convex-hull-set-1-jarviss-algorithm-or-wrapping/

[6] Wikipedia. "*Graham Scan".*
https://en.wikipedia.org/wiki/Graham_scan

[7] GeeksforGeeks. "*Convex Hull | Set 2 (Graham Scan)".*
https://www.geeksforgeeks.org/convex-hull-set-2-graham-scan/

[8] *https://www.ti.inf.ethz.ch/ew/Lehre/CG13/lecture/Chapter%203.pdf*

[9] Satyan L. Devadoss, J. O. (2011). "*Discrete And Computational Geometry".* Princeton University Press.

[10] Wikipedia. "*Delaunay Triangulation".*
https://en.wikipedia.org/wiki/Delaunay_triangulation

[11] https://www.ti.inf.ethz.ch/ew/Lehre/CG13/lecture/Chapter%206.pdf

[12] Emiris Ioannis & Fisikopoulos Vissarion, (2015), "Voronoi diagram and Delaunay triangulation".
https://opencourses.uoa.gr/modules/document/file.php/DI113/%ce%94%ce%b9%ce%b4%ce%b1%ce%ba%cf%84%ce%b9%ce%ba%cf%8c%20%ce%a0%ce%b1%ce%ba%ce%ad%cf%84%ce%bf/%ce%94%ce%b9%ce%b1%cf%86%ce%ac%ce%bd%ce%b5%ce%b9%ce%b5%cf%82/2a.delaunay.pdf

[13] Wikipedia. Voronoi diagram.
https://en.wikipedia.org/wiki/Voronoi_diagram

[14] NG, K. (2018). "*Machine Learning Projects for Mobile Applications".* Packt Publishing Ltd.

[15] Glenn Eguchi, "Delaunay Triangulations".
http://web.mit.edu/alexmv/Public/6.850-lectures/lecture09.pdf

[16] “Algorithm Design and Analysis”.
https://faculty.cs.byu.edu/~farrell/courses/CS312/projects/ConvexHull.php

[17] Lischinski D. “Incremental Delaunay Triangulation”. Cornell University.
http://www.karlchenofhell.org/cppswp/lischinski.pdf

[18] D.R. Chand & S.S. Kapur, “An algorithm for convex polytopes”. J. Assoc. Comput. Mach., 17 (1970).

[19] Rex A. Dwyer. “A Faster Divide-and-Conquer Algorithm for Constructing Delaunay Triangulations”. Algorithmica (1987).

[20] Kazemi Vahid & Sullivan Josephine. “One Millisecond Face Alignment with an Ensemble of Regression Trees”.
http://www.csc.kth.se/~vahidk/papers/KazemiCVPR14.pdf

[21] Wikipedia. “Heapsort”.
https://en.wikipedia.org/wiki/Heapsort

[22] Hurtado, F. & M. Noy & J. Urrutia (1999). “Flipping Edges in Triangulations. Discrete & Computational Geometry”.
https://link.springer.com/article/10.1007%2FPL00009464

[23] Guibas, Leonidas J. & Knuth, Donald E. & Sharir, Micha (1992). “Randomized incremental construction of Delaunay and Voronoi diagrams”.
https://link.springer.com/article/10.1007%2FBF01758770

[24] “COMPUTING CONSTRAINED DELAUNAY TRIANGULATIONS IN THE PLANE”
http://www.geom.uiuc.edu/~samuelp/del_project.html

[25] ”S-hull”
http://www.s-hull.org/paper/s_hull.pdf

[26] The Irish Times. “How Voronoi diagrams help us understand our world”.
https://www.irishtimes.com/news/science/how-voronoi-diagrams-help-us-understand-our-world-1.2947681

[27] Shubhendu Trivedi. “Voronoi Art”. Onionesque Reality.
https://onionesquereality.wordpress.com/2008/12/13/voronoi-art/

[28] “Thiessen Polygon”. Geographic Information Technology Training Alliance
http://www.gitta.info/Accessibilit/en/html/UncProxAnaly_learningObject4.html

[29] Guha Sumanta. “Computer Graphics Through OpenGL®: From Theory to Experiments”. CRC Press

[30] “Delaunay Triangulations”.
https://www2.cs.duke.edu/courses/fall08/cps230/Lectures/L-21.pdf

[31] Boulos Solomon. “Introduction to Voronoi Diagrams and Delaunay Triangulations”. http://www.cs.utah.edu/~csilva/courses/cpsc7960/pdf/boulos-DT.pdf

[32] Du Dingzhu & Hwang Frank. “Computing in Euclidean Geometry”. World Scientific.

[33] Petr Felkel. “Convex Hull In 3 Dimensions”. Fel Ctu Prague. https://cw.fel.cvut.cz/old/_media/courses/ae4m39vg/lectures/05-convexhull-3d-split.pdf

[34] Preparata F. P. & Hong S. J. . “Convex hulls of finite sets of points in two and three dimensions”. Communications of the ACM Vol 20 Issue 2, Feb 1977.

[35] Wikipedia. “Randomized algorithm”. https://en.wikipedia.org/wiki/Randomized_algorithm

[36] Wikipedia. Quickhull. https://en.wikipedia.org/wiki/Quickhull

[37] Wikibooks. “Algorithm Implementation/Geometry/Convex hull/Monotone chain”. https://en.wikibooks.org/wiki/Algorithm_Implementation/Geometry/Convex_hull/Monotone_chain

[38] A. M. Andrew. "Another Efficient Algorithm for Convex Hulls in Two Dimensions", Info. Proc. Letters 9, 216-219 (1979).

[39] Wikipedia. “Kirkpatrick–Seidel algorithm”. https://en.wikipedia.org/wiki/Kirkpatrick%E2%80%93Seidel_algorithm

[40] Kirkpatrick, David G.; Seidel, Raimund (1986). "The ultimate planar convex hull algorithm". SIAM Journal on Computing.

[41] Wikipedia. “Chan's algorithm”. https://en.wikipedia.org/wiki/Chan%27s_algorithm

[42] Timothy M. Chan. "Optimal output-sensitive convex hull algorithms in two and three dimensions". Discrete and Computational Geometry, Vol. 16, pp.361–368. 1996.

[43] Jarvis, R. A. (1973). "On the identification of the convex hull of a finite set of points in the plane". Information Processing Letters.

[44] Graham, R.L. (1972). "An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set". Information Processing Letters. http://www.math.ucsd.edu/~ronspubs/72_10_convex_hull.pdf

[45] Guy Blelloch, Yan Gu, Julian Shun and Yihan Sun. “Parallelism in Randomized Incremental Algorithms”. SPAA 2016. https://www.cs.cmu.edu/~guyb/papers/BGSS16.pdf

Παράρτημα 2: Εικόνες

Εικόνα 2.1: Περιγραφή Convex Hull[2]
Εικόνα 2.2: Περιγραφή Convex Hull[16]
Εικόνα 2.3: Οπτικοποίηση αλγόριθμου Gift Wrapping[4]
Εικόνα 2.4: Οπτικοποίηση αλγόριθμου Graham Scan[6]
Εικόνα 3.1: Περιγραφή Delaunay[10]
Εικόνα 3.2: Περιγραφή διαγράμματος Voronoi[13]
Εικόνα 3.3: Περιγραφή Flipping για Delaunay[10]
Εικόνα 3.4: Περιγραφή Flipping για Delaunay[10]
Εικόνα 3.5: Περιγραφή Flipping για Delaunay[10]
Εικόνα 4.1: Τα 68 σημαντικά σημεία που βρίσκονται σε κάθε πρόσωπο. Η εικόνα δημιουργήθηκε από τον Brandon Amos της CMU[14]
Εικόνα 4.2: Παράδειγμα μη seamless faceswap
Εικόνα 4.3: Παράδειγμα seamless faceswap
Εικόνα 4.4: Παράδειγμα video faceswap

Οι εικόνες του κεφαλαίου 4Δ αποτελούν screenshot του κάθε κομματιού της εφαρμογής που αναλύεται αντίστοιχα.