מבוא למדעי המחשב 67101

תרגיל 7 – עיבוד תמונה: Image Morphing

שימו לב – תרגיל זה ניתן לשבועיים, כאשר את החלק הראשון שלו יש להגיש בעוד שבוע, ואת התרגיל כולו יש להגיש בעוד שבועיים.

מועדי ההגשה:

חלק ראשון: יום חמישי, 18.12.2014 בשעה 21:00

חלק ראשון+שני: יום חמישי, 25.12.2014 בשעה 21:00

הקדמה

מורפינג (Morphing) הוא אפקט מיוחד בסרטים ובאנימציות בו דמות או אובייקט משנים את צורתם באופן הדרגתי ורציף.

סרטון שמדגים בצורה יפה image morphing של נשים באומנות:

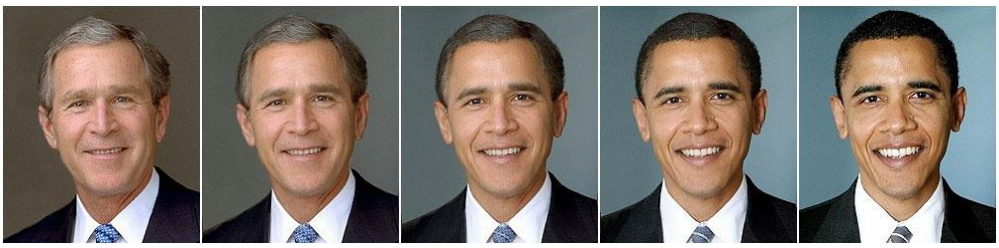
https://www.youtube.com/watch?v=nUDIoN-_Hxs

וסרטון נוסף שמדגים image morphing של כלבים חמודים:

https://www.youtube.com/watch?v=QgFX5halk6s

image morphing הוא תהליך של אינטרפולציה ("ערבוב") בין שתי תמונות, על מנת ליצור משהו שנראה כמו מעבר יפה וחלק בין תמונות הקלט השונות.

דוגמא לרצף של 5 תמונות:



בתרגיל זה נכתוב תוכנית הממשת אלגוריתם של Image Morphing, היוצר סידרה של 40 תמונות שעוברת בהדרגה מתמונה אחת לתמונה אחרת.

עליכם להוריד מהאתר את הקבצים הדרושים לתרגיל:

1. gui.py – מודול זה כבר מומש בשבילכם, והוא מכיל את כל המימוש של ה-gui הדרוש לתרגיל. כמו כן בקובץ זה נמצא ה-main, ולכן הוא זה שמריץ את כל התרגיל, וקורא לפונקציות שאתם תכתבו בקובץ ex7.py. אל תעשו שום שינוי בקובץ gui.py!
2. ex7.py – זה הקובץ שאתם תצטרכו לממש בו את הפונקציות הדרושות, ע"פ ההוראות המפורטות בהמשך. (יש למחוק את המילה pass בפונקציות אלה, ובמקומה להכניס את המימוש שלכם לפונקציות).
3. SolveLinear3.py – בקובץ זה יש פונקציה אחת שאתם צריכים לתרגיל שכבר ממומשת בשבילכם, ואתם יכולים לקרוא לה (אל תעשו בה שינוי).

4. im1.jpg – התמונה ההתחלתית (תמונת המקור):

5. im2.jpg – התמונה הסופית (תמונת היעד):

שימו לב, התרגיל מורכב בצורה מובנית ממספר משימות, שבסופו של דבר יתחברו ביחד וירכיבו את המוצר הסופי הממשש image morphing. לכן עקבו אחר ההוראות והשלבים של התרגיל, והקפידו לכתוב את הקוד שלכם במדויק על פי הנחיות התרגיל.

אתם יכולים לכתוב בקובץ ex7.py פונקציות עזר נוספות מלבד אלה הדרושות בתרגיל, ולהשתמש בהן בקוד שלכם. אבל הפונקציות הדרושות בתרגיל חייבות להיכתב בדיוק על פי הדרישות המפורטות להלן.

הקפידו לכתוב תיעוד לקוד שלכם ובפרט לכל פונקציה שאתם כותבים.

הספריה PIL

ה-gui הנתון לכם בתרגיל זה עושה שימוש בספריה PIL של פייתון, ולכן צריך אותה על מנת להריץ את התרגיל.

ספריה זו אינה חלק מהספריות שהן built-in בפייתון, ולכן יש צורך להתקין אותה בנוסף לפייתון.

במעבדת המחשבים של האוניברסיטה (האקוריום) כבר מותקנת ספריה זו, ואין צורך להתקין שום דבר.

על מנת לעבוד על המחשבים שלכם, עליכם להתקין את הספריה הזו. הוראות ההתקנה מפורטות בסוף המסמך.

האלגוריתם

האלגוריתם שנממש בתרגיל זה מבוסס על התאמות של נקודות בין תמונת המקור (source) לתמונת היעד (target).

האלגוריתם שלנו יוצר רצף של תמונות, שכל אחת מהן היא ערבוב קצת שונה בין תמונת המקור לבין תמונת היעד. כלומר בכל תמונה ברצף יש מידה מסוימת של דמיון לתמונת המקור, ומידה מסוימת של דמיון לתמונת היעד. מידה זו של דמיון מיוצגת בעזרת המספר אלפא α, שמקבל ערכים בין 0 ל-1. כאשר $\alpha = 0$, התמונה זהה לתמונת המקור, וכש-$\alpha = 1$ התמונה זהה לתמונת היעד. כל שאר התמונות ברצף מקבלות ערכים שונים של α בין 0 ל-1, בהתאם למידת הדמיון שלהן לתמונת המקור ולתמונת היעד. למשל התמונה שעבורה $\alpha = \frac{1}{2}$ היא בדיוק באמצע בין תמונת המקור לתמונת היעד.

האלגוריתם שלנו מתואר באופן הבא:

<u>קלט</u>: תמונת מקור (im1) ותמונת יעד (im2)

<u>פלט</u>: סדרה של 40 תמונות היוצרות morphing בין תמונת המקור לתמונת היעד.

שלבי האלגוריתם:

1. מציאת זוגות של נקודות מתאימות בין תמונת המקור לתמונת היעד, כאשר כדאי להתאים אוביקטים בתמונה אחת לאוביקטים שדומים להם בתמונה השניה (כמו עיניים, אף, תווי פנים דומים וכו').
2. חלוקת התמונות למשולשים – באופן נפרד בתמונת המקור ובתמונת היעד: כאשר הנקודות שהתקבלו בשלב 1 מהוות את קודקודי המשולשים. (כלומר מתיחת קווים בין הנקודות משלב 1 כך שמתקבלים משולשים המכסים את כל התמונה. המשולשים יהיו שונים בתמונת המקור ובתמונת היעד, בהתאם לקודקודים שהתקבלו בשלב 1).
3. יצירת morph בנפרד בין כל משולש בתמונת המקור למשולש המתאים לו בתמונת היעד, וצירוף כל המשולשים לתמונות שלמות, המהוות רצף של מעבר הדרגתי מתמונת המקור לתמונת היעד.

וכעת להסבר מפורט יותר של התרגיל:

שלב ראשון – סימון זוגות של נקודות מתאימות

לצורך התרגיל מסופק לכם gui, בעזרתו ניתן להציג את התמונות ולבצע את השלבים השונים של האלגוריתם.

הריצו את ה-gui של התרגיל מה-shell בעזרת הרצת הפקודה:

```
python3 gui.py <image_source> <image_target> <num_frames> <out_dir> <max_x> <max_y>
```

כאשר:

image_source - string שהוא השם של תמונת המקור, שהערך הדיפולטיבי שלו הוא "im1.jpg"

image_target - string שהוא השם של תמונת היעד, שהערך הדיפולטיבי שלו הוא "im2.jpg"

num_frames - מספר int חיובי שהוא מספר התמונות ברצף התמונות הסופי שאנו יוצרים בתרגיל, שהערך הדיפולטיבי שלו הוא 40.

out_dir – string שהוא השם של תיקיית היעד אליה יכנסו התמונות המיוצרות ע"י התרגיל, והערך הדיפולטיבי שלה הוא "images". כלומר באופן דיפולטיבי ה-gui מייצר תיקייה בשם "images" באותו מקום שבו שמור ה-gui, ומכניס אליה את תמונות הפלט של התרגיל.

max_x – מספר int חיובי שהוא הרוחב של התמונה (מספר העמודות), שהערך הדיפולטיבי שלו הוא 500.

max_y – מספר int חיובי שהוא האורך של התמונה (מספר השורות), שהערך הדיפולטיבי שלו הוא 350.

אם מריצים רק את השורה: python3 gui.py ללא פרמטרים, הריצה תתבצע עם הערכים הדיפולטיביים הנ"ל.

שימו לב שאם תריצו את ה-gui עם ערכים של max_x ו-max_y השונים מהערכים הדיפולטיביים של 500 ו-350, התצוגה של התמונות ב-gui תראה פחות טוב. אתם יכולים לעשות את זה כאשר אתם רוצים לדבג את התרגיל, ושהתרגיל ירוץ מהר יותר.

כמו כן על מנת שהתרגיל ירוץ מהר יותר, רצוי מאוד בזמן העבודה על התרגיל להריץ את התרגיל עם ערך קטן של num_frames (למשל 3-4).

יש להריץ את ה-gui עם 6 פרמטרים בדיוק (המפורטים למעלה), או בלי פרמטרים בכלל - ואז ילקחו הערכים הדיפולטיביים של כל אחד מהם.

דוגמא להרצת ה-gui עם פרמטרים שונים בחלקם מהערכים הדיפולטיביים:

```
python3 gui.py "im1.jpg" "im2.jpg" 3 "images" 100 50
```

ואז למשל מספר הפריימים שווה ל-num_frames = 3, רוחב התמונה שווה ל- max_x = 100 , ואורך התמונה שווה ל- max_y = 50.

אתם יכולים להניח תקינות של הקלט (כפי שמוסבר למעלה).

לאחר שה-gui נפתח, הגדילו את החלון שלו כך שיוצג על פני כל המסך.

ב-gui אתם צריכים לסמן זוגות של נקודות מתאימות על פני התמונות – נקודה אחת בתמונה העליונה ונקודה שניה בתמונה התחתונה. הנקודות ממוספרות, כך שאתם יכולים לראות שהתאמתם למשל את נקודה מספר 5 בתמונה העליונה לנקודה מספר 5 בתמונה התחתונה.

באופן דיפולטיבי כבר מסומנות תמיד בהתחלה הנקודות של הפינות בשתי התמונות, כך שכבר בהתחלה יש 4 זוגות של נקודות מתאימות בין התמונות. כדאי להוסיף עוד זוגות של נקודות מתאימות על מנת שתהליך ה-morphing יראה יותר טוב. כאמור, כדאי להתאים אוביקטים בתמונה העליונה לאוביקטים **דומים** להם בתמונה התחתונה (כמו עיניים, אף, פה, תווי פנים דומים וכו').

כמו כן יש ב-gui **כפתור של delete last point**, המבטל את הנקודה האחרונה שסומנה על התמונה.

שימו לב לא למחוק את הנקודות של הפינות שכבר מסומנות על התמונות!! (כלומר לא ללחוץ על הכפתור delete last point בהתחלה, כי אז ימחקו הנקודות של הפינות, שהן הכרחיות להצלחת האלגוריתם).

שחקו קצת עם ה-gui ותראו שאתם מבינים כיצד עובד סימון הנקודות על התמונות.

יש לסמן את אותו מספר נקודות בשתי התמונות, מכיוון שכל זוג נקודות (הממוספר באותו אינדקס ב-gui על העיגול של הנקודה) צריך להיות זוג נקודות שמהווה התאמה בין התמונות – בין תמונת המקור לתמונת היעד.

שימו לב – לאחר שתממשו את בנית המשולשים, אז לאחר כל זוג נקודות שאתם מסמנים (אחת בתמונה העליונה ושניה שמתאימה לה בתמונה התחתונה), יש ללחוץ על הכפתור **draw triangles.** לאחר שתממשו את בניית המשולשים, יופיעו משולשים כחולים על התמונות. לאחר שהופיעו הקווים הכחולים של המשולשים הקיימים, אתם יכולים לסמן עוד זוג נקודות מתאימות, וללחוץ שוב על כפתור זה.

אל תסמנו נקודות מחוץ לגבולות התמונות (למרות שזה אפשרי, זה יפגע באלגוריתם).

כמו כן **אל תסמנו נקודות על קווים כחולים (צלעות) של משולשים שכבר מופיעים על התמונות** (אחרי שיצרתם כבר משולשים והם סומנו על התמונות), וגם לא על קודקודים של משולשים שכבר מופיעים.

שימו לב: לאחר שיצרתם משולשים תואמים בין התמונות, זוג הנקודות הבא שאתם מסמנים צריך להיות **חוקי** – כפי שמוסבר בהמשך בשלב השני.

מספיק לסמן **מספר קטן של זוגות נקודות מתאימות** על מנת לקבל תוצאות יפות. כאשר אתם מסמנים הרבה נקודות התאמה, התרגיל ירוץ לאט יותר, מכיוון שיקח יותר זמן לייצר כל תמונת ביניים ברצף התמונות. לכן כאשר אתם עובדים על התרגיל ומדבגים אותו, סמנו מספר קטן מאוד של זוגות נקודות מתאימות (זוג אחד או שניים), ובדקו שהתרגיל עובד היטב. כאשר הכל יעבוד טוב תוכלו לסמן יותר זוגות של נקודות, על מנת לקבל תוצאות יפות יותר של morphing. בכל מקרה, אין צורך לסמן יותר מ-7 זוגות של נקודות מתאימות.

<u>שלב שני – בנית משולשים מתאימים מתוך הנקודות</u>

כאמור, עלינו ליצור משולשים שקודקודיהם הם הנקודות התואמות שסומנו בשלב 1. המשולשים צריכים להיות מסודרים ברשימה שבה יש חשיבות לסדר – המשולש באינדקס i ברשימת המשולשים של תמונת המקור צריך להתאים למשולש באינדקס i ברשימת המשולשים של תמונת היעד. התאמה זו מתקבלת מכך שעוברים על הנקודות שסומנו בשלב 1 על פי הסדר שהן סומנו, מכיוון שנקודות אלה כבר יוצרות התאמות בין אוביקטים בתמונת המקור לאוביקטים דומים להם בתמונת היעד.

לצורך ביצוע שלב זה, עליכם לממש את הפונקציות הבאות, על פי ההגדרות המפורטות להלן.

ב-gui יש **כפתור שנקרא draw triangles**, שכאשר לוחצים עליו נקראת הפונקציה create_triangles, אותה אתם תממשו בקובץ ex7.py. פונציה זו מייצרת משולשים עבור כל אחת מהתמונות (המקור והיעד) **בנפרד**. כמו כן בעת לחיצה על כפתור זה נקראת גם הפונקציה do_triangle_lists_match שתממשו , הבודקת האם שתי רשימות המשולשים שנוצרו עבור תמונת המקור ותמונת היעד אכן יוצרות משולשים תואמים (כלומר ע"פ ההתאמות של הנקודות שסומנו), המסודרים בסדר הנכון ברשימות המשולשים.

שימו לב: כפי שנאמר קודם, לאחר כל זוג נקודות תואמות שאתם מסמנים על ה-gui יש ללחוץ על הכפתור **draw triangles** היוצר ומצייר משולשים. לאחר שסומנו משולשים על התמונות, זוג הנקודות התואמות הבא שמסומן צריך להיות **חוקי,** ביחס למשולשים שנוצרו עד עכשיו. הסבר: נסמן את זוג הנקודות החדש ב-point1, point2, כאשר

point1 היא נקודה בתמונת המקור ו-point2 היא נקודה מתאימה לה בתמונת היעד. צריך להתקיים שהמשולש אליו שייכת הנקודה point1 **יתאים** למשולש שבו נמצאת הנקודה point2 (כלומר המשולשים צריכים להיות **באותו אינדקס** ברשימות המשולשים של תמונת המקור ותמונת היעד, המתקבלות מהפונקציה create_triangles שאתם תממשו).
אם המשולשים לא מתאימים, זוג הנקודות החדש לא חוקי, ואז הפונקציה do_triangle_lists_match שלכם תחזיר False. במקרה כזה התוכנית תעצור ותצא עם ערך שגיאה (exit(1)), ותודפס הודעת השגיאה הבאה ל-shell:

"Marked points are not in matching triangles. Please mark the points again".

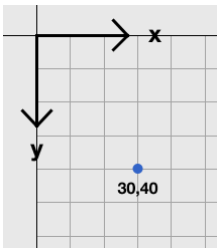
ואז יש להריץ מחדש את ה-gui ולסמן נקודות חדשות.

הפונקציה create_triangles שלכם תקבל מה-gui בתור פרמטר רשימה של נקודות שסומנו באחת התמונות (תמונת היעד או תמונת המקור), והיא צריכה ליצור מרשימת נקודות זו רשימה של משולשים מתאימים. פונקציה זו תיעזר בפונקציה נוספת שעליכם לממש שמתוארת בהמשך.

לאחר שתממשו את בניית המשולשים, לחיצה על הכפתור draw triangles ב-gui תקרא לפונקציה create_triangles שלכם, ותצייר על פני התמונות משולשים (כחולים) לפי רשימת המשולשים שאתם מייצרים.

שימו לב שלאורך כל התרגיל, מערכת הצירים שנעבוד איתה היא כזו:

(באיור מסומנת נקודה שהקורדינטות שלה הן x=30, y=40).



כלומר ראשית הצירים (0,0) נמצאת בפינה השמאלית העליונה של התמונות, כאשר ציר ה-x מתקדם ימינה, וציר ה-y מתקדם כלפי מטה.

הגודל של כל אחת מהתמונות הוא 500 x 350, כלומר הערך המקסימלי של x הוא 500, והערך המקסימלי של y הוא 350. במילים אחרות, בכל תמונה יש 350 שורות של פיקסלים ו-500 עמודות של פיקסלים.

לאורך כל התרגיל אתם יכולים להניח לגבי כל הנקודות (x,y) שערכי x הם בין 0 ל-500, וערכי y הם בין 0 ל-350.

חלק ראשון – להגשה עד ה-18.12.2014:

1. הפונקציה is_point_inside_triangle

עליכם לממש את הפונקציה is_point_inside_triangle, הבודקת האם נקודה נתונה נמצאת בתוך משולש שקודקודיו נתונים.

חתימת הפונקציה צריכה להיות:

```
def is_point_inside_triangle(point, v1,  v2, v3):
```

הפונקציה מקבלת ארבעה פרמטרים:

point – הנקודה שאנו רוצים לבדוק האם היא נמצאת בתוך המשולש. הנקודה מתקבלת כ-tuple של שני מספרים (floats), המייצגים את הקורדינטות (x,y) של הנקודה.

v1, v2, v3 – הקודקודים של המשולש. גם הקודקודים הם נקודות הנתונות כ-tuple של שני מספרים (floats), המייצגים את הקורדינטות (x,y) של כל קודקוד.

הפונקציה צריכה להחזיר tuple בגודל 2 המכיל את שני הדברים הבאים:

א. ערך מסוג bool שמשמעותו: True אם הנקודה point נמצאת בתוך המשולש שקודקודיו הם v1, v2, v3, ו-False אחרת. (אם הנקודה נמצאת על היקף המשולש, יש להחזיר True).
ב. tuple בגודל 3 שמכיל את שלושת המספרים a, b, c כפי שמוסבר בהדרכה הבאה: (גם a,b,c הם floats)

הדרכה:

נסמן את הנקודה הנתונה point ב- $p = (p_x, p_y)$ ואת שלושת הקודקודים של המשולש ב- $v1 = (v1_x, v1_y)$, $v2 = (v2_x, v2_y)$, $v3 = (v3_x, v3_y)$.

אנו רוצים למצוא פתרון עבור המשתנים a,b,c במערכת המשוואות הבאה:

$$v1_x \cdot a + v2_x \cdot b + v3_x \cdot c = p_x$$

$$v1_y \cdot a + v2_y \cdot b + v3_y \cdot c = p_y$$

$$a + b + c = 1$$

יש לנו כאן מערכת של 3 משוואות עם 3 נעלמים: a,b,c. לאחר שנפתור אותה נקבל פתרון (כלומר השמה) למשתנים a,b,c.

מתקיים שעבור נקודה p **הנמצאת בתוך משולש** שקודקודיו הם v1, v2, v3, הפתרון של מערכת המשוואות הנ"ל נותן ערכי a,b,c **שכל אחד מהם בין 0 ל-1**, כלומר: $a, b, c \in [0,1]$.

(נקודה הנמצאת על היקף המשולש נחשבת כנמצאת בתוך המשולש).

ואילו עבור נקודה p **הנמצאת מחוץ משולש** שקודקודיו הם v1, v2, v3, הפתרון של מערכת המשוואות הנ"ל נותן ערכי a,b,c **שלפחות אחד מהם קטן מ-0.** כלומר: $a < 0$ או $b < 0$ או $c < 0$.

לצורך פתרון מערכת משוואות ב-3 נעלמים אתם יכולים לקרוא לפונקצית העזר solve_linear_3 שכבר מומשה עבורכם בקובץ SolveLinear3.py. פונקציה זו מקבלת 2 פרמטרים:

coefficients_list – רשימה של 3 רשימות של המקדמים של a,b,c.

הרשימה הראשונה צריכה להכיל את המקדמים של a,b,c במשוואה הראשונה במערכת המשוואות הנ"ל, כלומר $[v1_x,\ v2_x,\ v3_x]$. הרשימה השנייה צריכה להכיל את המקדמים של a,b,c במשוואה השנייה במערכת המשוואות הנ"ל, ובאותו אופן לגבי הרשימה השלישית.

right_hand_list – רשימה של כל הביטויים הכתובים באגף ימין במערכת המשוואות הנ"ל, כלומר: $[p_x, p_y, 1]$.

הפונקציה solve_linear_3 מחזירה tuple בגודל 3 של הפתרון עבור a,b,c למערכת המשוואות הנ"ל.

2. הפונקציה create_triangles:

עליכם לממש את הפונקציה create_triangles היוצרת רשימה של משולשים מתוך רשימה של נקודות.

חתימת הפונקציה צריכה להיות:

```
def create_triangles(list_of_points):
```

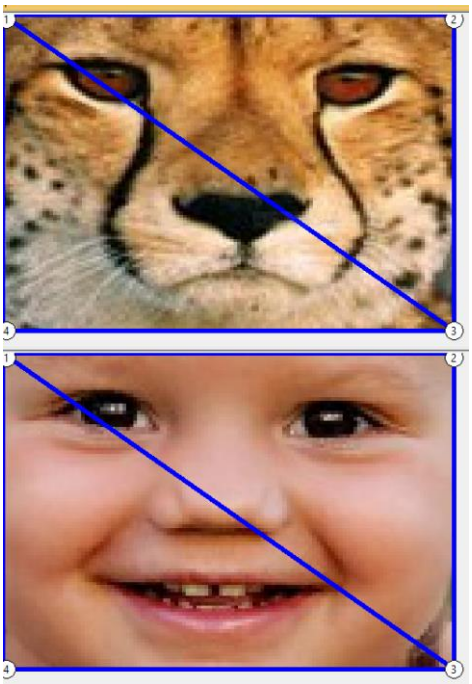
הפונקציה מקבלת פרמטר אחד: list_of_points, המייצג את רשימת הנקודות שסומנו על פני אחת מהתמונות (תמונת המקור או תמונת היעד). כלומר זה list של נקודות, כאשר כל נקודה היא tuple בגודל 2 של מספרים (floats), המהווים את הקורדינטות (x,y) של הנקודה.

הפונקציה מחזירה list של משולשים המיוצגים כ- tuples בגודל 3. כלומר כל tuple כזה הוא שלשה של נקודות שמהוות קודקודים של משולש, כאשר גם נקודות אלה הן tuple של שני מספרים (floats), המייצגים את הקורדינטות (x,y) של הנקודה. זאת אומרת, בסה"כ מחזירים list של tuples בגודל 3, שכל איבר ב-tuple כזה הוא בעצמו tuple בגודל 2.

הפונקציה צריכה להשתמש בפונקציה is_point_inside_triangle שמימשתם קודם (כעת נשתמש רק בערך הבוליאני של True או False שפונקציה זו מחזירה, ולא במקדמים a,b,c. בהם נשתמש בפונקציות אחרות בהמשך).

הפונקציה צריכה לעבור על הנקודות שברשימה על פי הסדר שלהן, וליצור מהן משולשים, על פי שני השלבים הבאים:

שלב א': לעבור רק על 4 הנקודות הראשונות ברשימה (כלומר על הנקודות שבאינדקסים 0 עד 3 ברשימה). נקודות אלה הן תמיד הפינות של התמונה, הנכנסות ראשונות לרשימת הנקודות באופן דיפולטיבי על ידי ה-gui. מתוך 4 הנקודות האלה עליכם ליצור שני משולשים גדולים, הנראים כך:

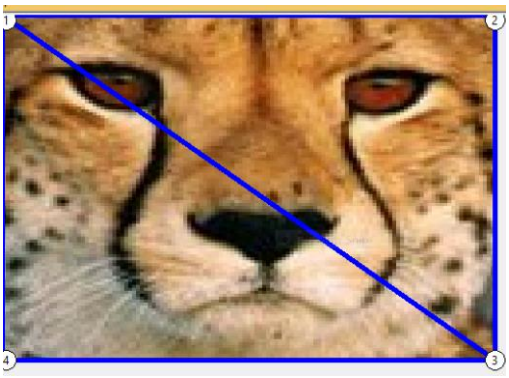


כלומר אם לא מסמנים שום נקודות נוספות על התמונות ולוחצים על הכפתור draw triangles ב-gui כאשר מופיעות רק הנקודות של הפינות על התמונות, אז המשולשים שצריכים להיות מצויירים על התמונה הם אלה שמופיעים בתמונה הנ"ל.

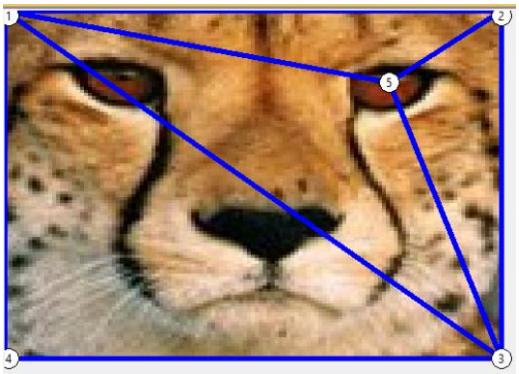
שלב ב': לעבור על הנקודות הבאות, שנמצאות ברשימה אחרי 4 הנקודות הראשונות (כלומר החל מאינדקס 4). יש לעבור על הנקודות האלה לפי הסדר שלהן ברשימה, ועבור כל אחת מהן לבדוק בתוך איזה מבין המשולשים שקיימים כבר ברשימת המשלושים היא נמצאת. (בהתחלה כאמור יש שני משולשים גדולים כמו בתמונה הנ"ל).

עבור כל נקודה A כזו ברשימה (החל מאינדקס 4): לאחר שמוצאים בתוך איזה משולש נמצאת הנקודה A (נקרא למשולש זה T), יש **למחוק** מרשימת המשולשים את המשולש T, ובמקומו להכניס לרשימת המשולשים **3 משולשים חדשים** הנוצרים ע"י חיבור של 3 הקודקודים של המשולש T עם הנקודה A.

לדוגמא: אם סומנה נקודה בתוך המשולש הגדול העליון (נקודה מספר 5), הוא צריך להפוך לשלושת המשולשים הבאים:

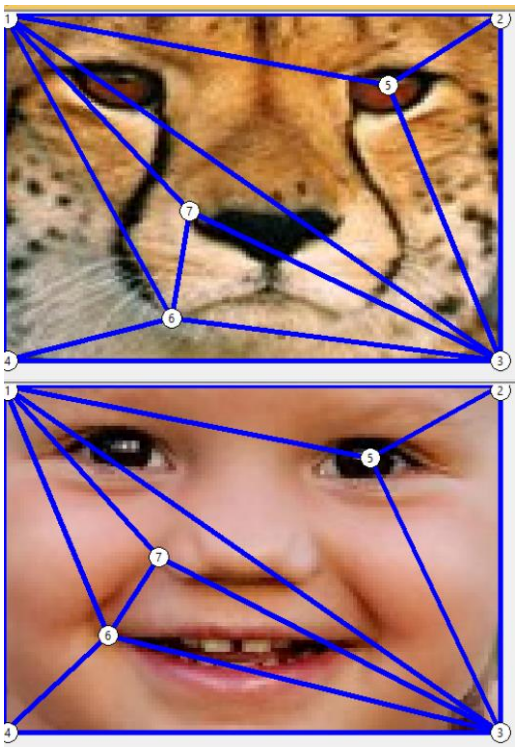


המשולשים **לפני** סימון נקודה מספר 5



המשולשים **אחרי** סימון נקודה מספר 5

דוגמא ליצירת משולשים מתאימים בין התמונות:



3. הפונקציה do_triangle_lists_match:

חתימת הפונקציה צריכה להיות:

def do_triangle_lists_match(list_of_points1, list_of_points2):

הפונקציה מקבלת שני פרמטרים: list_of_points1, list_of_points2 , שהם שתי רשימות של נקודות. הפונקציה צריכה לקרוא לפונקציה create_triangles שמימשתם קודם עבור כל אחת משתי רשימות הנקודות הללו, ולקבל שתי רשימות של משולשים – נסמן אותן ב- triangles_list1, triangles_list2. לאחר מכן הפונקציה צריכה לבדוק שעבור כל נקודה באינדקס i ברשימה list_of_points1 – נסמן נקודה זו ב-point_i_1 , ועבור כל נקודה **באותו אינדקס i** ברשימה list_of_points2 – נסמן נקודה זו ב-point_i_2, שמתקיים:

שהאינדקס (ברשימת המשולשים triangles_list1) של המשולש ש-point_i_1 נמצאת בו שווה לאינדקס (ברשימת המשולשים triangles_list2) של המשולש שהנקודה point_i_2 נמצאת בו.

אם מוצאים אינדקס i אחד כזה שעבורו זה לא מתקיים, הפונקציה צריכה להחזיר False. ואם זה מתקיים לכל אינדקס i, הפונקציה צריכה להחזיר True.

על מנת לקבל את האינדקס של משולש שנקודה מסוימת נמצאת בו, אתם יכולים להשתמש בפונקציה is_point_inside_triangle שמימשתם קודם (או שפונקצית עזר שלכם המשתמשת בה).

<u>שלב שלישי – יצירת רצף של תמונות המהווה מעבר הדרגתי מתמונת המקור לתמונת היעד</u>

בשלב זה אתם צריכים לממש את הפונקציות הבאות:

4. הפונקציה get_point_in_segment :

חתימת הפונקציה צריכה להיות:

def get_point_in_segment(p1, p2, alpha):

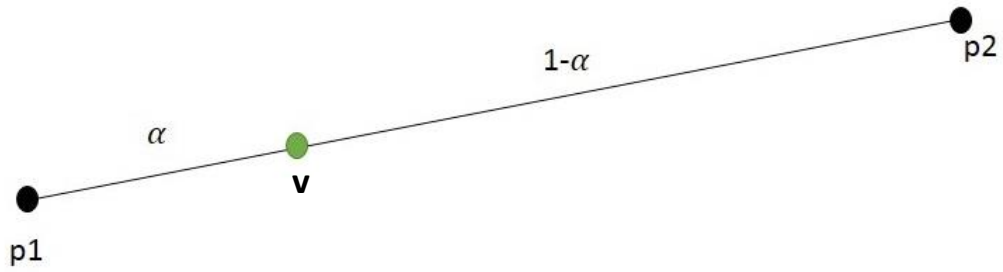
הפונקציה מקבלת שלושה פרמטרים:

p1, p2 – שתי נקודות. כל אחת מהן נתונה כ-tuple של שני מספרים (floats) שהם הקורדינטות של הנקודות.

alpha – מספר (float) בין 0 ל-1. (יכול להיות גם 0 או 1)

לצורך ההסבר נסמן את הקטע המחבר בין p1 ל-p2 ב-I. ונסמן ב- $|I|$ את אורך הקטע הזה.

הפונקציה צריכה להחזיר נקודה v (גם כן כ-tuple של שני מספרים (floats) שהם הקורדינטות של הנקודה), כך ש:
הנקודה v נמצאת על הקטע המחבר בין p1 ל-p2, במרחק $\alpha|I|$ מ-p1, ובמרחק $(1-\alpha)|I|$ מ-p2.



הדרכה: אם נסמן את הנקודה שאנו צריכים להחזיר ב-$v=(v_x,v_y)$, ואת הנקודות הנתונות ב- $p1=(p1_x,p1_y)$, $p2=(p2_x,p2_y)$, אז צריך להתקיים:

$$v_x=(1-\alpha)\cdot p1_x+\alpha\cdot p2_x$$
$$v_y=(1-\alpha)\cdot p1_y+\alpha\cdot p2_y$$

5. **הפונקציה get_intermediate_triangles:**

חתימת הפונקציה צריכה להיות:

def get_intermediate_triangles(source_triangles_list, target_triangles_list, alpha):

פונקציה זו מקבלת שני פרמטרים:

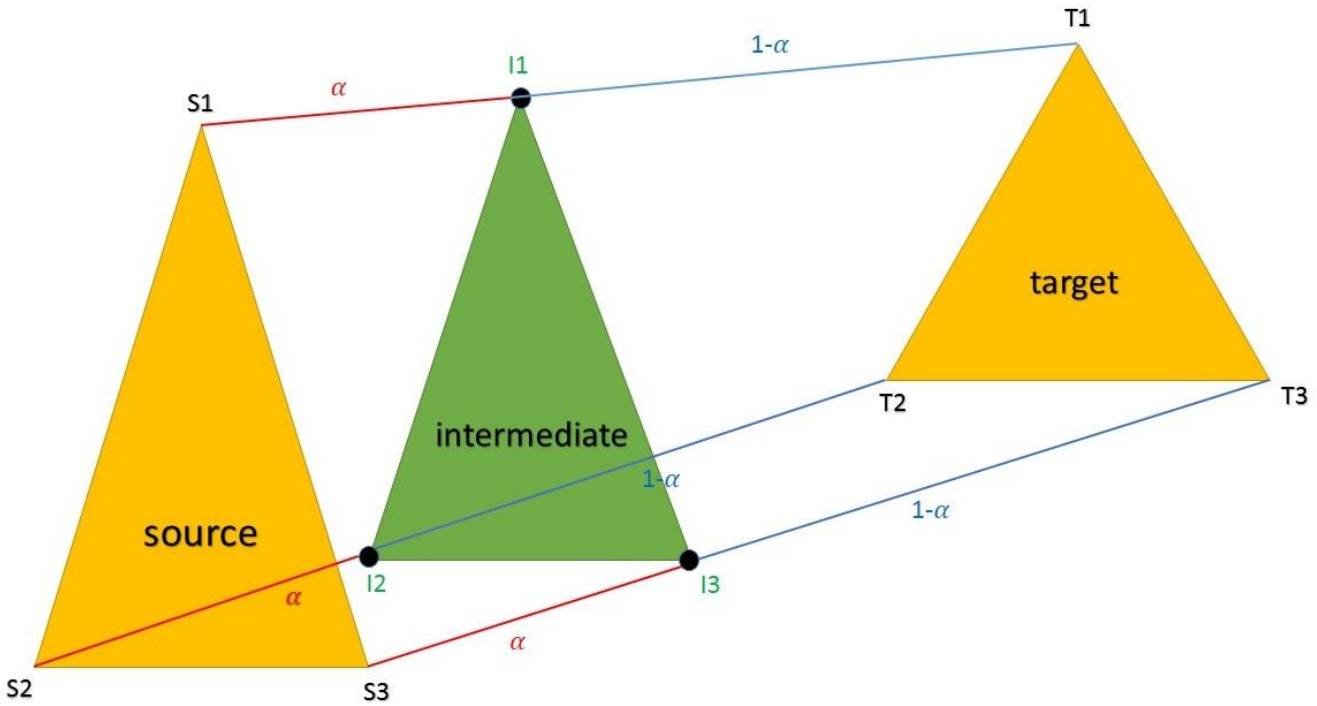
source_triangles_list - רשימת המשולשים של תמונת המקור. רשימה זו היא list של tuples בגודל 3, כאשר כל tuple כזה מייצג 3 קודקודים של משולש. כל קודקוד במשולש הוא נקודה המיוצגת ע"י tuple בגודל 2 של קורדינטות x,y.

target_triangles_list - רשימת המשולשים של תמונת היעד. נתונה באותה צורה כמו המשולשים של תמונת המקור.

alpha - מספר (float) בין 0 ל-1. (יכול להיות גם 0 או 1)

פונקציה זו צריכה להשתמש בפונקציה get_point_in_segment שמימשתם קודם.

עבור כל זוג משולשים מתאימים (כלומר משולש אחד הנמצא באינדקס i ברשימת המשולשים של תמונת המקור, ומשולש שני שנמצא באותו אינדקס i ברשימת המשולשים של תמונת היעד), הפונקציה צריכה למצוא משולש ביניים מתאים – שקוקודיו נמצאים במרחק α ממשולש המקור, ובמרחק $1-\alpha$ ממשולש היעד.



כלומר צריך למצוא את קודקודי משולש הביניים : $I1, I2, I3$ כך שמתקיימים התנאים הבאים :

לצורך ההסבר נסמן את הקטע המחבר בין S1 ל-T1 ב-$(S1, T1)$. ונסמן ב- $|(S1, T1)|$ את אורך הקטע הזה.

הפונקציה צריכה להחזיר את הנקודה $I1 = (I_x, I_y)$ כך שהנקודה נמצאת על הקטע המחבר בין S1 ל-T1, במרחק $\alpha|(S1, T1)|$ מ-S1, ובמרחק $(1-\alpha)|(S1, T1)|$ מ-T1.

ובאותו אופן בדיוק עבור שני הקודקודים האחרים של משולש הביניים : $I2, I3$.

בסופו של דבר הפונקציה צריכה להחזיר רשימה של משולשים כאלה (שכל משולש הוא tuple בגודל 3 של נקודות, וכל נקודה היא tuple בגודל 2 של מספרים (floats)), כאשר ברשימה זו יש באינדקס i את משולש הביניים הנוצר בדרך הנ"ל מזוג המשולשים המתאימים הבא : המשולש שנמצא באינדקס i ברשימת המשולשים של תמונת המקור, והמשולש שנמצא באינדקס i ברשימת המשולשים של תמונת היעד.

חלק שני – להגשה עד ה-25.12.2014:

6. הפונקציה get_array_of_matching_points:

חתימת הפונקציה צריכה להיות :

```
def get_array_of_matching_points(size, triangles_list, intermediate_triangles_list):
```

הפונקציה מקבלת 3 פרמטרים :

size – tuple בגודל 2 המכיל את (max_x, max_y) , כלומר את רוחב התמונה - max_x ואורך התמונה - max_y. (ערכים אלה מתקבלים בשורת ההרצה של התוכנית, כאשר הערכים הדיפולטיביים שלהם הם max_x=500, max_y=350).

triangles_list – רשימה של משולשים, או של תמונת המקור או של תמונת היעד (זה לא משנה לצורך הפונקציה הזו אם זה של תמונת המקור או היעד)

intermediate_triangles_list – רשימת משולשי ביניים שמחושבים בהתאמה ע"י הפונקציה get_intermediate_triangles שמימשתם קודם. (רשימות המשולשים ניתנות בדיוק כמו בפונקציה הקודמת, כ-list של tuples בגודל 3 של קודקודי משולשים).

הפונקציה צריכה להחזיר רשימה דו-מימדית (list של list) בגודל 350 x 500 (כלומר בגודל התמונה), כך שלכל תא (x,y) ברשימה הדו מימדית, הערך שמוחזק בו שווה לנקודה המתאימה לו (x',y') במשולש המקור/היעד (לא משנה מי מהם).

עבור כל נקודה (x, y) כזו , הדרך למצוא את הנקודה המתאימה לה (x', y') היא:

א. למצוא בתוך איזה משולש מבין משולשי הביניים (כלומר מתוך intermediate_triangles_list) נמצאת הנקודה (x, y). נסמן ב- i את האינדקס של משולש זה ברשימת משולשי הביניים.

ב. עבור המשולש שבו נמצאת הנקודה (x, y), לשמור את המקדמים a, b, c של קודקודי המשולש, המתקבלים מהפעלת הפונקציה is_point_inside_triangle שמימשתם קודם.

ג. למצוא את המשולש שנמצא באותו אינדקס i ברשימה triangles_list (זה יהיה משולש בתמונת המקור או היעד). נסמן את קודקודי משולש זה ב- $v1 = (v1_x, v1_y)$, $v2 = (v2_x, v2_y)$, $v3 = (v3_x, v3_y)$.

ד. לקבל נקודה מתאימה (x', y') בתמונת המקור / היעד על ידי:

$$x' = a \cdot v1_x + b \cdot v2_x + c \cdot v3_x$$

$$y' = a \cdot v1_y + b \cdot v2_y + c \cdot v3_y$$

כלומר בסופו של דבר הפונקציה צריכה להחזיר רשימה דו מימדית, שבכל מיקום (x, y) בה יש את הנקודה (x', y') המתאימה לו, כלומר יש בו tuple בגודל 2 של מספרים (floats) שהם הקורדינטות (x', y').

<u>הצעה (לא חובה)</u>: על מנת לייעל את הריצה של הפונקציה וכדי שהתוכנית תרוץ מהר יותר, אתם יכולים בכל איטרציה לשמור את המשולש שבו נמצא פיקסל מסויים x,y, ועבור הפיקסל הבא לנסות קודם כל לבדוק האם הוא נמצא באותו משולש ששמרתם קודם (יש לכך סיכוי גבוה, שכן פיקסלים אלה הם שכנים). זה יכול לחסוך לכם מעבר על כל המשולשים עד שמוצאים לאיזה משולש הנקודה שייכת.

7. הפונקציה create_intermediate_image:

עליכם לממש את הפונקציה create_intermediate_image המייצרת תמונה אחת (Frame) מתוך רצף התמונות הסופי של התרגיל, המהווה מעבר הדרגתי בין תמונת המקור לתמונת היעד.

חתימת הפונקציה צריכה להיות:

```
def create_intermediate_image(alpha, size, source_image, target_image, source_triangles_list,
target_triangles_list):
```

כפי שהוסבר קודם, מידת הדמיון של כל אחת מהתמונות ברצף התמונות מיוצגת על ידי המספר אלפא α, שמקבל ערכים בין 0 ל-1. כאשר $\alpha = 0$, התמונה זהה לתמונת המקור, וכש-$\alpha = 1$ התמונה זהה לתמונת היעד. כל שאר התמונות ברצף מקבלות ערכים שונים של α בין 0 ל-1, בהתאם למידת הדמיון שלהן לתמונת המקור ולתמונת היעד. למשל תמונה שעבורה $\alpha = \frac{1}{2}$ היא בדיוק באמצע בין תמונת המקור לתמונת היעד.

הפונקציה מקבלת 6 פרמטרים:

alpha – מספר בין 0 ל-1 המייצג את מידת הדמיון לתמונת היעד ולתמונת המקור, כפי שהוסבר למעלה.

size – tuple בגודל 2 המכיל את (max_x, max_y) , בדיוק כמו שקיבלתם בפונקציה הקודמת.

source_image – תמונת המקור. התמונה הזו נתונה כרשימה דו מימדית של פיקסלים (בגודל 350 x 500), כאשר **הגישה לפיקסל x,y מסויים במערך זה מתבצעת ע"י הסינטקס הבא:**

source_image[x, y]

גישה כזו לפיקסל x,y מחזירה tuple בגודל 3 של ערכי ה-RGB (Red, Green, Blue) של הפיקסל x,y בתמונה. כלומר source_image[x, y] נותן tuple של 3 מספרים שהם int בין 0 ל-255, המייצגים את מידת האדום, הירוק והכחול שיש בכל פיקסל בתמונה, שביחד יוצרים את הצבע של כל פיקסל כפי שאנו רואים אותו בתמונה.

הסבר לגבי ייצוג צבע במחשב בעזרת שיטת RGB תוכלו למצוא בקישורים הבאים:

http://he.wikipedia.org/wiki/%D7%99%D7%99%D7%A6%D7%95%D7%92_%D7%A6%D7%91%D7%A2_%D7%91%D7%9E%D7%97%D7%A9%D7%91

http://en.wikipedia.org/wiki/RGB_color_model

target_image – תמונת היעד. גם התמונה הזו נתונה כמערך דו מימדי בדיוק כמו תמונת המקור.

source_triangles_list – רשימת המשולשים של תמונת המקור. רשימה זו היא list של tuples בגודל 3, כאשר כל tuple כזה מייצג 3 קודקודים של משולש. כל קודקוד במשולש הוא נקודה המיוצגת ע"י tuple בגודל 2 של קורדינטות x,y.

target_triangles_list - רשימת המשולשים של תמונת היעד. רשימה זו נתונה בדיוק כמו רשימת המשולשים של תמונת המקור.

הפונקציה create_intermediate_image צריכה להחזיר image – תמונת ביניים אחת מתוך רצף התמונות הסופי. תמונה זו צריכה להיות רשימה דו מימדית בגודל של 350 שורות על 500 עמודות, כאשר בכל תא ברשימה יש tuple בגודל 3 שהוא יהיה ערכי ה-RGB של תא זה בתמונת הביניים.

בתחילת הפונקציה שלכם שמרו את רשימת משולשי הביניים המתקבלים מקריאה לפונקציה get_intermediate_triangles שמימשתם קודם.

לאחר מכן שמרו שני מערכים של נקודות התאמה: אחד עבור התאמה בין משולשי הביניים למשולשי המקור, ואחד עבור התאמה בין משולשי הביניים למשולשי היעד, בעזרת קריאה לפונקציה get_arrays_of_matching_points שמימשתם קודם (כלומר שלחו לפונקציה פעם אחת את source_triangles_list ופעם אחת את target_triangles_list בתור הפרמטר triangles_list).

כעת עליכם לרוץ על כל פיקסל בתמונת הביניים image, **ועבור כל פיקסל (x,y) כזה:**

א. למצוא נקודה (x',y') המתאימה לו בתמונת המקור. נסמן נקודה זו ב- source_match_point.
ב. למצוא נקודה (x',y') (נקודה אחרת) המתאימה לו בתמונת היעד. נסמן נקודה זו ב- target_match_point.

ג. קבלו את ערכי ה-RGB בתמונה source_image , במיקום של הנקודה source_match_point. ערכי ה-RGB צריכים להתקבל בתור tuple בגודל 3. נסמן את ה-tuple הזה בשם source_RGB.

ד. קבלו את ערכי ה-RGB בתמונה target_image , במיקום של הנקודה target_match_point. גם כאן ערכי ה-RGB צריכים להתקבל בתור tuple בגודל 3. שמרו את ה-tuple הזה בשם target_RGB.

(זכרו שיש לגשת לפיקסל במערך של source_image ו-target_image באופן המיוחד שתואר קודם).

ה. בצעו השמה של ערכי הצבע RGB לפיקסל (x,y) באופן הבא:
כל ערך R, G או B של הפיקסל (x,y) צריך להיקבע כך שהוא שווה ל- $(1-\alpha)$ כפול ערך ה-R, G או B בהתאמה שב-source_RGB (כלומר בנקודה המתאימה בתמונת המקור), ועוד α כפול ערך ה-R, G או B בהתאמה שב-target_RGB (כלומר בנקודה המתאימה בתמונת היעד).

כלומר באופן כללי הנוסחה לחישוב הצבע היא כזו:

$$RGB(x,y) = \alpha \cdot source\_RGB + (1-\alpha) \cdot target\_RGB$$

כאשר יש לעשות את החישוב הזה עבור כל אחד מערכי R, G, B של כל פיקסל בתמונת הביניים image.

לבסוף החזירו את תמונת הביניים image.

8. **הפונקציה create_sequence_of_images:**

עליכם לממש את הפונקציה create_sequence_of_images המייצרת רצף של תמונות (Frames), המהווה מעבר הדרגתי בין תמונת המקור לתמונת היעד.

חתימת הפונקציה צריכה להיות:

```
def create_sequence_of_images(size, source_image, target_image, source_triangles_list,
target_triangles_list, num_frames):
```

הפונקציה מקבלת 6 פרמטרים, כאשר החמישה הראשונים הם בדיוק חמשת הפרמטרים שתוארו קודם, שמקבלת הפונקציה create_intermediate_image (בלי אלפא).

הפרמטר השישי הוא num_frames, שהוא מספר התמונות שיהיו ברצף התמונות הסופי.

הפונקציה create_sequence_of_images צריכה לקרוא לפונקציה create_intermediate_image שלכם, ולשלוח לה את ארבעת הפרמטרים הראשונים בדיוק כפי שהיא מקבלת אותם. בנוסף יש לשלוח ל-create_intermediate_image גם פרמטר alpha המקבל ערכים בין 0 ל-1, שהוא שונה על פי האינדקס של התמונה (האינדקסים של התמונות ברצף הם בין 0 ל- num_frames -1). למשל עבור התמונה שהאינדקס שלה הוא i ברצף, הערך של alpha צריך להיות: alpha = i / (num_frames -1).

(בסופו של דבר num_frames צריך להיות 40. ראו הערה בטיפים בהמשך).

הפונקציה create_sequence_of_images צריכה להחזיר list באורך num_frames של תמונות, כאשר כל תמונה מיוצרת ע"י הפונקציה create_intermediate_image.

הכפתור morphing!:

כפי שראיתם, ב-gui יש כפתור של morphing!, ולחיצה עליו קוראת לפונקציה create_sequence_of_images שלכם. התמונות שאתם מייצרים ברצף התמונות ישמרו אוטומטית ע"י ה-gui בתיקייה חדשה שנקראת images (שה-gui מייצר באותו מיקום שבו הקובץ gui.py שמור). בכל לחיצה על הכפתור morphing! תימחק התיקייה images שיוצרה קודם, ובמקומה תיווצר תיקייה חדשה עם התמונות החדשות שיוצרו כעת.

טיפים והנחיות

- כאשר אתם עובדים על התרגיל, בהתחלה צרו רצפים קטנים יותר של תמונות (Frames), למשל של 2-4 תמונות במקום 40, ורק אחרי שתראו שהתוכנית שלכם עובדת טוב עבור רצף קצר של תמונות תעברו ליצור רצף ארוך יותר. עבור 40 פריימים של תמונות זמן הריצה צפוי להיות ארוך, וזה בסדר.
- כמו כן, מספר זוגות הנקודות התואמות המסומנות על התמונות בשלב הראשון משפיע מאוד על זמן הריצה של התוכנית. לכן כאשר אתם עובדים על התרגיל, סמנו ב-gui רק זוג אחד או שניים של נקודות מתאימות על תמונת המקור ותמונת היעד. לאחר שהתוכנית שלכם תעבוד טוב, סמנו יותר נקודות תואמות (אך אין צורך ביותר מ-7 זוגות של נקודות תואמות).
- דבר נוסף המשפיע על זמן הריצה של התוכנית הוא גודל התמונות – כלומר ערכי max_x ו-max_y. אתם יכולים להריץ את התרגיל עם ערכים קטנים יותר שלהם, אך שימו לב שזה יגרום לתצוגה של התמונות להיראות הרבה פחות טוב. בסופו של דבר ה-morphing של התמונות צריך להיראות טוב כאשר max_x=500, max_y=350, שהם הערכים הדיפולטיביים שלהם.

- הקפידו לכתוב את הקוד שלכם בצורה מדויקת וברורה, ולשים לב לאינדקסים וגבולות של לולאות וקונטיינרים.
- על מנת לראות בצורה יפה את המעברים בין התמונות ברצף התמונות שאתם מייצרים, כדאי לכם להפוך את רצף התמונות לסרט ולראות זאת (אפשר להפוך אותן לקובץ בפורמט gif למשל, או בכל דרך אחרת שתרצו).

ב-Windows, תוכלו גם לעשות זאת ע״י התקנת התוכנה Windows Movie Maker :

http://windows.microsoft.com/en-us/windows-live/movie-maker

פשוט העלו אליה את התמונות שהתקבלו לכם, ותוכלו לראות ולשמור סרט (בפורמט mp4) שנוצר מתמונות אלה.

- אתם יכולים להריץ את התרגיל על כל זוג תמונות שתרצו, ולקבל morphing של התמונות שלכם. פשוט שלחו את שמות התמונות שאתם רוצים כפרמטרים לתוכנית בשורת ההרצה.

נהלי הגשה

הלינק להגשה של החלק הראשון של התריל הוא תחת השם: ex7a

(לגבי הגשת החלק השני של התרגיל, הוראות התרגיל יעודכנו בשבוע הבא).

בתרגיל זה עליכם להגיש את הקבצים הבאים :

1. ex7.py – עם המימושים שלכם לפונקציות.
2. README (על פי פורמט ה-README לדוגמא שיש באתר הקורס, ועל פי ההנחיות לכתיבת README המפורטות בקובץ נהלי הקורס).

יש ליצור קובץ tar הנקרא ex7.tar המכיל בדיוק את שני הקבצים הנ״ל, בעזרת פקודת ה-shell הבאה :

tar cvf ex7.tar ex7.py README

- מומלץ לבדוק את קובץ ה-tar שיצרתם על ידי העתקת התוכן שלו לתיקייה נפרדת ופתיחתו (extract) בעזרת ביצוע הפקודה : **tar xvf ex7.tar**, ולאחר מכן יש לבדוק באמצעות הפקודה **ls -h** שכל הקבצים הדרושים קיימים שם ולא ריקים.

סקריפט קדם-הגשה (Pre submit script) : זהו סקריפט לבדיקה בסיסית של קבצי ההגשה של התרגיל.

על מנת להריץ את הסקריפט לתרגיל 7 (לחלק הראשון של התרגיל) הריצו את השורה הבאה ב-shell :

```
~intro2cs/bin/presubmit/ex7 ex7.tar
```

הסקריפט מייצר הודעת הצלחה במקרה של מעבר כל הבדיקות הבסיסיות והודעות שגיאה רלוונטיות במקרה של כישלון בחלק מהבדיקות.

שימו לב, סקריפט קדם ההגשה נועד לוודא רק תקינות בסיסית, ומעבר של בדיקות הסקריפט לא מבטיח את תקינותה של התוכנית! עליכם לוודא בעצמכם שהתוכנית שלכם פועלת כפי שדרוש.

הגשת קובץ tar

עליכם להגיש את הקובץ ex7.tar בקישור ההגשה של תרגיל 7 – **שימו לב להגיש את החלק הראשון של התרגיל במקום המיועד לו (ex7a)** ! לאחר הגשת התרגיל, ניתן ומומלץ להוריד אותו ולוודא כי הקבצים המוגשים הם אלו שהתכוונתם להגיש וכי הקוד עובד על פי ציפיותיכם.

לאחר שאתם מגישים את התרגיל באתר הקורס, תוך מספר שניות יווצר הקובץ submission.pdf . עליכם לבדוק שהכל תקין בקובץ submission.pdf! ואם יש בעיה כלשהי בקבצים שלכם שבאה לידי ביטוי בקובץ ה-pdf עליכם לתקן אותה, גם אם לא נאמר שום דבר בפירוש בתרגיל לגבי זה, כך שקובץ ה-pdf שמיוצר מהתרגיל שלכם יהיה תקין לגמרי. **וודאו שאין שורות ארוכות מדי** בקוד שלכם שנחתכות בקובץ ה-pdf, ושהקובץ מסודר ורואים בו את הכל בצורה טובה וברורה. זכרו את ההגבלה של ***79 תווים לכל היותר בשורה*** (כולל הערות).

הוראות להתקנת הספריה PIL על <u>Windows</u> (במחשבים של האקווריום זה כבר מותקן):

פתחו את התיקייה שבה נמצא פייתון אצלכם על המחשב. למשל: C:\Python34.

1. פתחו command line ע"י כתיבת האותיות cmd בכתובת של החלון (כך תקבלו תקבלו command line שכבר נמצא בתיקייה של פייתון).
2. ב- command line הקלידו cd Skripts , על מנת להיכנס לתיקייה Skripts של פייתון.
3. הקלידו ב- command line: easy_install.exe Pillow>2.0.0

- אם זה לא עובד לכם, ניתן להתקין את PIL גם בעזרת installers אחרים שיש באנטרנט. שימו לב רק שאתם מתקינים גירסה של PIL שמתאימה **לפייתון 3**, ומתאימה למערכת ההפעלה שיש לכם.

בהצלחה! ☺