Θέματα Όρασης Υπολογιστών

Ακαδημαϊκό Έτος 2024-2025

Άσκηση 2

ΒΑΣΙΚΟΙ ΓΕΩΜΕΤΡΙΚΟΙ ΜΕΤΑΣΧΗΜΑΤΙΣΜΟΙ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

Ονοματεπώνυμο: Φωτάκης Ανδρέας

ΑΜ: 1084674

Ερώτημα 1

1. imread

Η συνάρτηση imread διαβάζει μια εικόνα από ένα αρχείο και τη φορτώνει στον χώρο εργασίας του MATLAB ως πίνακα (array) δεδομένων. Υποστηρίζει διάφορες μορφές αρχείων εικόνας, όπως JPEG, PNG, BMP, TIFF, κ.ά. Η σύνταξή της είναι:

```
A = imread(filename);
```

2. imwarp

Η συνάρτηση imwarp εφαρμόζει γεωμετρικούς μετασχηματισμούς σε μια εικόνα, όπως μετατοπίσεις, περιστροφές, κλιμακώσεις και στρεβλώσεις. Η βασική σύνταξη είναι:

```
B = imwarp(A, tform);
```

Όπου A είναι η αρχική εικόνα και tform είναι ένας μετασχηματισμός, όπως affine2d ή projective2d. Η συνάρτηση επιστρέφει την μετασχηματισμένη εικόνα B.

mathworks.com

3. affine2d

Το affine2d δημιουργεί ένα αντικείμενο που αντιπροσωπεύει έναν 2D affine μετασχηματισμό. Οι affine μετασχηματισμοί περιλαμβάνουν μετατοπίσεις, περιστροφές, κλιμακώσεις και διατμήσεις. Η δημιουργία ενός τέτοιου αντικειμένου γίνεται ως εξής:

```
tform = affine2d(T);
```

Όπου T είναι ένα 3x3 μητρώο που καθορίζει τον affine μετασχηματισμό. Το αντικείμενο tform μπορεί να χρησιμοποιηθεί με συναρτήσεις όπως η imwarp για την εφαρμογή του μετασχηματισμού σε εικόνες.

4. projective2d

Το projective2d δημιουργεί ένα αντικείμενο που αντιπροσωπεύει έναν 2D προβολικό μετασχηματισμό. Οι προβολικοί μετασχηματισμοί μπορούν να αλλάξουν την προοπτική μιας εικόνας, επιτρέποντας μετασχηματισμούς που δεν είναι δυνατοί με τους affine μετασχηματισμούς. Η σύνταξη είναι:

```
tform = projective2d(T);
```

Όπου T είναι ένα 3x3 μητρώο που καθορίζει τον προβολικό μετασχηματισμό. Το αντικείμενο tform μπορεί να χρησιμοποιηθεί με τη συνάρτηση imwarp για την εφαρμογή του μετασχηματισμού σε εικόνες.

5. imref2d

Το imref2d δημιουργεί ένα αντικείμενο που αποθηκεύει τη σχέση μεταξύ των εσωτερικών συντεταγμένων μιας 2D εικόνας και των χωρικών συντεταγμένων σε ένα σύστημα παγκόσμιων συντεταγμένων. Αυτό είναι χρήσιμο για τη διατήρηση της γεωμετρίας της εικόνας κατά την εφαρμογή μετασχηματισμών. Η δημιουργία ενός τέτοιου αντικειμένου γίνεται ως εξής:

R = imref2d(imageSize, xWorldLimits, yWorldLimits);

Όπου imageSize είναι το μέγεθος της εικόνας και xWorldLimits, yWorldLimits είναι τα όρια των συντεταγμένων στον παγκόσμιο χώρο. Το αντικείμενο R μπορεί να χρησιμοποιηθεί με συναρτήσεις όπως η imwarp για τον καθορισμό της προβολής της εξόδου.

mathworks.com

6. implay

Η συνάρτηση implay ανοίγει ένα παράθυρο αναπαραγωγής πολυμέσων που μπορεί να χρησιμοποιηθεί για την προβολή ακολουθιών εικόνων ή βίντεο. Η βασική σύνταξη είναι:

implay(filename);

Όπου filename είναι το όνομα του αρχείου βίντεο ή μια ακολουθία εικόνων. Αυτό είναι χρήσιμο για την ανάλυση δυναμικών δεδομένων εικόνας.

Ερώτημα 2

Ο **Affine μετασχηματισμός** είναι μια γραμμική μετατροπή που διατηρεί τις ευθείες και τις παραλληλίες μεταξύ των σημείων. Ένας από τους βασικούς τύπους affine μετασχηματισμών είναι η **κλιμάκωση (scaling)**, η οποία αλλάζει το μέγεθος ενός αντικειμένου.

Η κλιμάκωση σε δύο διαστάσεις εκφράζεται ως:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

όπου:

- s_x είναι ο συντελεστής κλιμάκωσης στον άξονα x
- s_y είναι ο συντελεστής κλιμάκωσης στον άξονα y
- (x, y) είναι οι αρχικές συντεταγμένες ενός σημείου
- (x', y') είναι οι μετασχηματισμένες συντεταγμένες

Εάν $s_x = s_y$, τότε έχουμε **ομοιόμορφη κλιμάκωση (uniform scaling)**. Αν είναι διαφορετικοί, τότε έχουμε **μη-ομοιόμορφη κλιμάκωση (non-uniform scaling)**.

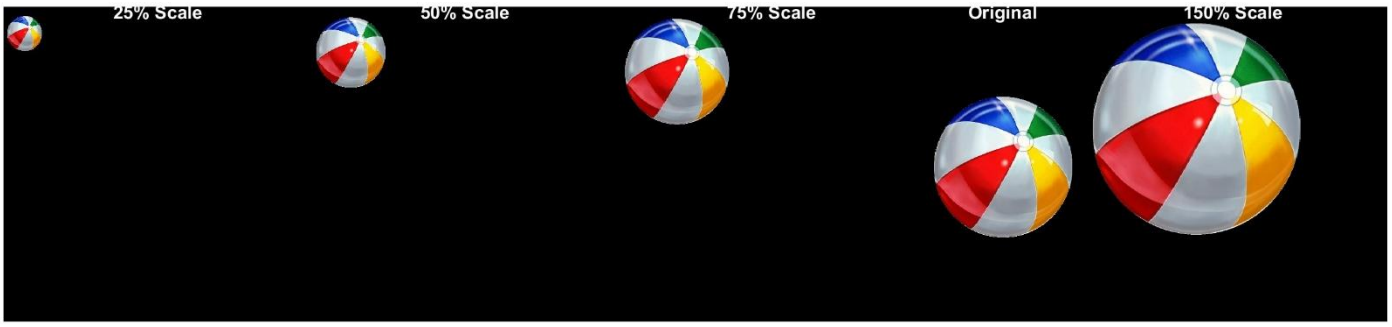
Για την εφαρμογή μετασχηματισμών κλιμάκωσης στην αρχική εικόνα, ορίζουμε ένα αντικείμενο μετασχηματισμού με την συνάρτηση affine2d() και μέσω της συνάρτησης imwrap() εφαρμόζεται πάνω στην εικόνα. Διαφορετικά μπορούμε να χρησιμοποιήσουμε την imresize.

```
sx = 1.5; % Κλιμάκωση κατά 1.5 φορές στον άξονα x
sy = 2.0; % Κλιμάκωση κατά 2 φορές στον άξονα y % Εφαρμογή της κλιμάκωσης scaled_img =
imresize(img, [round(size(img,1) * sy), round(size(img,2) * sx)]);
```

Εφαρμογή 4 παραμορφώσεις ομοιόμορφης κλιμάκωσης στην εικόνα της μπάλας.

```
scale1 = affine2d([0.5 0 0; 0 0.5 0; 0 0 1]);   % 50% Scale
scale2 = affine2d([1.5 0 0; 0 1.5 0; 0 0 1]);   % 150% Scale
scale3 = affine2d([0.75 0 0; 0 0.75 0; 0 0 1]);  % 75% Scale
scale4 = affine2d([0.25 0 0; 0 0.25 0; 0 0 1]);  % 25% Scale

I1 = imwarp(ball_removed_bg, scale1, 'OutputView', outputView);   % 50%
I2 = imwarp(ball_removed_bg, scale2, 'OutputView', outputView);   % 150%
I3 = imwarp(ball_removed_bg, scale3, 'OutputView', outputView);   % 75%
I4 = imwarp(ball_removed_bg, scale4, 'OutputView', outputView);   % 25%
```



Ερώτημα 3 & 4

Η διαδικασία του shearing εφαρμόζεται μέσω της χρήσης ενός μετασχηματισμού Affine. Αρχικά, ορίζονται οι συντελεστές διάτμησης (sh_x, sh_y), όπου sh_x καθορίζεται από το ημιτονοειδές σήμα $x\_n(i)$, επιτρέποντας τη σταδιακή μεταβολή της διάτμησης σε κάθε καρέ.

```
for i = 1:frames
    ...

    x_n = 0.2 * sin(2 * pi * f * t);

    sh_y = 0;
    sh_x = x_n(i);

    A = [1 sh_x 0;
         sh_y 1 0;
         0    0 1];

    tform = affine2d(A');

    %% Apply scaling to image with imwarp
    im_temp = imwarp(im, tform, 'FillValues', 1.0);

    ...
end
```

Στη συνέχεια, δημιουργείται το μητρώο μετασχηματισμού A, ο οποίος περιέχει τον όρο sh_x στο αντίστοιχο σημείο για να προκαλέσει οριζόντια στρέβλωση της εικόνας. Η συνάρτηση $affine2d(A')$ χρησιμοποιείται για να μετατρέψει αυτό το μητρώο σε ένα αντικείμενο μετασχηματισμού που εφαρμόζεται στην εικόνα μέσω της $imwarp()$, επιτρέποντας την οριζόντια στρέβλωση της γεωμετρίας της εικόνας μέσα στην ακολουθία των frame, δίνοντας ένα οπτικό εφέ κλίσης (shearing) που μοιάζει με κύμα. Το ['FillValues', 1.0] στην $imwarp$ εξασφαλίζει ότι οποιεσδήποτε κενές περιοχές που δημιουργούνται λόγω του μετασχηματισμού γεμίζονται με την τιμή 1 (λευκό).

```
    %% Place image to the right coordinates
    [k, l, ~] = size(im_temp);
    start_m = 1;
    end_m = start_m + k - 1;

    if sh_x > 0
        start_n = ceil(n / 2) - ceil((m * sh_x));
    else
        start_n = ceil(n / 2);
    end
    end_n = start_n + l - 1;

    image(start_m:end_m, start_n:end_n, :) = im_temp;
```

Αφού εφαρμοστεί ο μετασχηματισμός, ορίζονται οι διαστάσεις της νέας εικόνας im_temp και υπολογίζονται οι κατάλληλες συντεταγμένες για την τοποθέτησή της στον

συνολικό καμβά image. Ο συνολικός καμβάς έχει πλάτος 2*n, δηλαδή είναι πιο πλατύς από την αρχική εικόνα, για να υπάρχει χώρος για μετατοπίσεις.

Ορίζοντας το κέντρο του οριζόντιου άξονα ως ceil(n / 2), διασφαλίζεται ότι η εικόνα θα τοποθετηθεί γύρω από το κεντρικό σημείο του καμβά.

Όταν sh_x > 0:

Η εικόνα μετατοπίζεται προς τα δεξιά. Για να αντισταθμιστεί αυτή η μετατόπιση και να παραμείνει οπτικά κεντραρισμένη, υπολογίζεται ένα επιπλέον offset: ceil(m * sh_x).Έτσι, το start_n (η αρχική οριζόντια θέση στο canvas) μειώνεται κατά αυτό το offset, μετακινώντας την εικόνα προς τα αριστερά ώστε να αντισταθμίσει την δεξιά μετατόπιση.

Όταν sh_x ≤ 0:

Δεν απαιτείται αντιστάθμιση για θετική μετατόπιση, οπότε η τοποθέτηση ξεκινά απλά από το κέντρο (ceil(n / 2)).

Με το να ορίζουμε το start_m στο 1 και το end_m ως start_m + k - 1, διασφαλίζουμε ότι το ύψος της μετασχηματισμένης εικόνας παραμένει αμετάβλητο και τοποθετείται σωστά στην κάθετη διεύθυνση.

Ερώτημα 5

Αρχικά αφού διαβαστούν οι εικόνες `windmill.png, windmill_mask.png, windmill_back.jpeg` η μάσκα (`windmill_mask`) ελέγχεται αν είναι έγχρωμη και μετατρέπεται σε ασπρόμαυρη (`rgb2gray`). Στη συνέχεια, δυαδικοποιείται (`imbinarize`) και δημιουργείται η αντίστροφη μάσκα (`inverseMask`), όπου η δυαδικότητα αντιστρέφεται για σωστή ενσωμάτωση στην τελική εικόνα.

```
% Get the dimensions of the images
[windmillRows, windmillCols, ~] = size(windmill);
[bgRows, bgCols, ~] = size(background);

% Calculate the center of the windmill and the background
centerWindmill = [windmillCols / 2, windmillRows / 2];
centerBackground = [bgCols / 2, bgRows / 2];
```

Μέσω των διαστάσεων των εικόνων υπολογίζεται το κέντρο του ανεμόμυλου και του background.

Σε κάθε frame:

```
numFrames = 150; % Total frames for rotation sequence
rotationAngles = linspace(0, 360, numFrames);

for i = 1:numFrames
   ...

    theta = rotationAngles(i); % Rotation angle in degrees
    A = [ cosd(theta) sind(theta) 0;
         -sind(theta) cosd(theta) 0;
          0           0           1];

    tform_rt = affine2d(A'); % Create affine transformation

    % Define reference object for the transformation
    imref_temp = imref2d(size(windmill)); % Reference for transformation

    % Apply transformation to the windmill image & mask
    rotatedWindmill = imwarp(windmill, imref_temp, tform_rt, 'Interp', 'cubic',
'FillValues', 1);
    rotatedMask = imwarp(windmill_mask, imref_temp, tform_rt, 'Interp',
'cubic', 'FillValues', 0);
```

Υπολογίζουμε για κάθε frame έναν Affine μετασχηματισμό περιστροφής με βάση την αντίστοιχη γωνία από το διάνυσμα rotationAngles. Η συνάρτηση affine2d μετατρέπει το μητρώο σε ένα αντικείμενο μετασχηματισμού για την εφαρμογή αυτού του μετασχηματισμού τόσο στην εικόνα του ανεμόμυλου όσο και στη μάσκα της μέσω της συνάρτησης imwarp.

```
    % Center the rotated windmill on the background
    [rotRows, rotCols, ~] = size(rotatedWindmill);
    offsetX = round(centerBackground(1) - rotCols / 2);
    offsetY = round(centerBackground(2) - rotRows / 2);
```

Στη συνέχεια, υπολογίζουμε τις μετατοπίσεις που απαιτούνται για να τοποθετήσουμε την περιστρεφόμενη εικόνα στο κέντρο της εικόνας του φόντου μέσω των υπολογισμών του κέντρου της εικόνας που περιγράφηκε προηγουμένως.

```
% Create a canvas the size of the background
    canvasWindmill = zeros(bgRows, bgCols, 3, 'uint8');
    canvasMask = false(bgRows, bgCols);

    % Place the rotated windmill and mask on the canvas
    yRange = max(1, offsetY):min(bgRows, offsetY + rotRows - 1);
    xRange = max(1, offsetX):min(bgCols, offsetX + rotCols - 1);

    % Ensure the ranges are valid
    windmillYRange = max(1, 1 - offsetY):min(rotRows, bgRows - offsetY);
    windmillXRange = max(1, 1 - offsetX):min(rotCols, bgCols - offsetX);

    % Add the rotated windmill to the canvas, respecting transparency
    canvasWindmill(yRange, xRange, :) = rotatedWindmill(windmillYRange,
windmillXRange, :);
```

```
    canvasMask(yRange, xRange) = rotatedMask(windmillYRange, windmillXRange);
```

Αρχικά, δημιουργούνται δύο καμβάδες με τις ίδιες διαστάσεις όπως το φόντο: ένας για την εικόνα του ανεμόμυλου (canvasWindmill) και ένας binary για τη μάσκα (canvasMask). Υπολογίζονται οι έγκυρες περιοχές (ranges) στον καμβά όπου θα τοποθετηθεί η περιστρεφόμενη εικόνα και η αντίστοιχη μάσκα, λαμβάνοντας υπόψη τις μετατοπίσεις offsetX και offsetY, δηλαδή στο κέντρο της εικόνας του φόντου. Οι μεταβλητές windmillYRange και windmillXRange καθορίζουν τα αντίστοιχα τμήματα της περιστρεφόμενης εικόνας που ταιριάζουν με τον καμβά. Η χρήση των max και min εξασφαλίζει ότι οι τιμές παραμένουν εντός ορίων, αποτρέποντας προβλήματα με υπερβάσεις των ορίων της εικόνας.

```
% Combine the canvas with the background
    combinedImage = background;
    for c = 1:size(background, 3)
        combinedImage(:, :, c) = combinedImage(:, :, c) .* uint8(~canvasMask) +
...
                                 canvasWindmill(:, :, c) .* uint8(canvasMask);
    end
```

Τέλος, η εικόνα και η μάσκα ενσωματώνονται στον καμβά και, με ένα βρόχο για κάθε κανάλι χρώματος, η εικόνα συνδυάζεται με το φόντο με βάση την διαφάνεια που ορίζει η μάσκα. Για κάθε κανάλι, τα pixel του φόντου διατηρούνται εκεί που η μάσκα είναι μη ενεργή, ενώ αντικαθίστανται από την εικόνα του ανεμόμυλου στις περιοχές που είναι ενεργή η μάσκα.

Αυτή η εικόνα που δημιουργείται σε κάθε frame γράφεται στο αρχείο του βίντεο.

Το αποτέλεσμα φαίνεται στο αρχείο βίντεο **transf_windmill.mp4**

Ερώτημα 6

```
rotatedWindmill = imwarp(windmill, imref_temp, tform_rt, 'Interp', 'method', 'FillValues', 1);
rotatedMask = imwarp(windmill_mask, imref_temp, tform_rt, 'Interp','method', 'FillValues', 0);
```

Στην περίπτωση της μεθόδου **nearest neighbor (**`'method'= 'nearest'`**)**, η παρεμβολή γίνεται με βάση το πλησιέστερο διαθέσιμο pixel, χωρίς να λαμβάνονται υπόψη τα γειτονικά. Αυτό έχει ως αποτέλεσμα να εμφανίζονται έντονα σκαλοπάτια στις άκρες του ανεμόμυλου και μια πιο "pixelated" εικόνα. Παρόλο που είναι η πιο γρήγορη μέθοδος, η ποιότητα του αποτελέσματος αναμένεται να είναι η πιο χαμηλή.

Με τη μέθοδο **linear (**`'method'= 'linear'`**)** , η παρεμβολή γίνεται με τη χρήση των τεσσάρων πλησιέστερων pixel, παίρνοντας τη μέση τιμή των τιμών τους. Το αποτέλεσμα είναι πιο ομαλές άκρες και λιγότερο εμφανή σκαλοπάτια σε σχέση με τη μέθοδο nearest neighbor. Ωστόσο, η ποιότητα της εικόνας ενδέχεται να επηρεαστεί από ελαφριά θολούρα, καθώς η ανάμειξη των pixel μπορεί να μειώσει την ευκρίνεια των λεπτομερειών.

Η μέθοδος **cubic (**`'method'= 'cubic'`**)** χρησιμοποιεί πληροφορία από 16 γειτονικά pixel για να υπολογίσει την τιμή του νέου pixel. Αυτό έχει ως αποτέλεσμα ένα ιδιαίτερα πιο ομαλό αποτέλεσμα, χωρίς απότομες μεταβολές και με καλύτερη διατήρηση των λεπτομερειών. Παρόλα αυτά απαιτεί περισσότερους υπολογισμούς και είναι πιο αργή από τις άλλες δύο μεθόδους.
Ο τελικός οπτικός αντίκτυπος των διαφορετικών μεθόδων παρεμβολής που παρατηρήθηκε δεν είναι αρκετός, αλλά παρουσιάζει μικρές διαφορές, στην ομαλότητα περιστροφής του ανεμόμυλου.

Ερώτημα 7

Φορτώνουμε τρεις εικόνες: τη «beach.jpg» ως φόντο, τη «ball.jpg» ως εικόνα της μπάλας και τη «ball_mask.jpg» ως μάσκα της μπάλας. Έπειτα, μετατρέπουμε τη μάσκα σε δυαδική (0 ή 1) μέσω της imbinarize, ώστε να μπορούμε να τη χρησιμοποιήσουμε για «κόψιμο» (masking) πάνω στην εικόνα της μπάλας. Τέλος, ορίζουμε τη μεταβλητή inverseMask ως το συμπλήρωμα της μάσκας, δηλαδή 1 - ball_mask, που είναι χρήσιμη όταν θέλουμε να κρατάμε το υπόβαθρο σε συνδυασμό με τη μπάλα.

Καθορίζουμε τις φυσικές παραμέτρους της κίνησης της μπάλας σε «μονάδες εικονοστοιχείων ανά καρέ». Η μεταβλητή pixel_gravity ορίζει την επιτάχυνση της βαρύτητας σε εικονοστοιχεία ανά καρέ τετράγωνο. Όταν η μπάλα αναπηδά, χάνει ένα ποσοστό ενέργειας καθορισμένο από τη μεταβλητή bounciness. Επιπλέον, μειώνουμε το μέγεθος της μπάλας κατά scale_factor = 0.2. Ορίζουμε γωνία εκτόξευσης θ = 70° και αρχική ταχύτητα v0_pixels = 35. Έτσι, οι συνιστώσες της αρχικής ταχύτητας είναι: $v0x = v_{0x} = v_0\, cos\theta$ $και$ $v_{0y} = v_0 \sin\theta$, όπως υλοποιείται με τις εντολές `cosd` και `sind` (σε μοίρες).

```
pixel_gravity = 0.6;       % downward acceleration per frame^2
bounciness = 0.6;          % energy loss factor on bounce
scale_factor = 0.2;        % ball scale

% Launch angle and velocity (in "pixels/frame")
theta = 70;                % Launch angle in degrees (increased from 50)
v0_pixels = 25;            % Initial speed in pixels/frame (increased from 25)

% Horizontal and vertical velocity components
v0x = v0_pixels * cosd(theta);
v0y = v0_pixels * sind(theta);
```

Υπολογίζουμε τις «κλιμακωμένες» διαστάσεις της μπάλας (ball_height_scaled, ball_width_scaled) πολλαπλασιάζοντας το αρχικό ύψος και πλάτος με τον παράγοντα κλίμακας scale_factor. Ορίζουμε την αρχική θέση της μπάλας (position_x, position_y) κοντά στην αριστερή πλευρά και αρκετά ψηλά (το beach_height - 300 την τοποθετεί 300 εικονοστοιχεία πάνω από το κάτω άκρο). Επίσης αντιστοιχούμε τις αρχικές ταχύτητες velocity_x και velocity_y στις τιμές που υπολογίσαμε νωρίτερα. Το ground_level ορίζεται ως το ύψος στο οποίο θα θεωρήσουμε ότι βρίσκεται το έδαφος, λίγο πάνω από τον πραγματικό πάτο της εικόνας.

```
ball_height_scaled = ball_height * scale_factor;
ball_width_scaled  = ball_width  * scale_factor;

% Start near the left side, but higher up:
position_x = ball_width_scaled / 2;
position_y = beach_height - 300;  % Higher than -200 to make the ball start
higher

velocity_x = v0x;
velocity_y = v0y;

% "Ground" is a bit above the bottom of the beach
ground_level = beach_height - 100;
```

Δημιουργούμε μία μεταβλητή rotation_angle για τη γωνία περιστροφής της μπάλας και μια σταθερή rotation_step που καθορίζει πόσο θα μεταβάλλεται η γωνία ανά καρέ (αρνητική τιμή σημαίνει περιστροφή δεξιόστροφα). Στη συνέχεια ξεκινάμε έναν βρόχο for από το 1 έως num_frames, ώστε να δημιουργήσουμε κάθε καρέ του βίντεο.

```
rotation_angle = 0;
rotation_step = -2;

for frame = 1:num_frames
    ...
end
```

Μέσα στον βρόχο, πρώτα ενημερώνουμε την ταχύτητα κατά τον κατακόρυφο άξονα με την επίδραση της βαρύτητας. Αν θεωρήσουμε ότι κάθε καρέ αντιστοιχεί σε ένα βήμα χρόνου $\Delta t = 1/fps$, μπορούμε να πούμε ότι $v_y(t + \Delta t) = v_y(t) - g \cdot \Delta t$.

Επειδή όμως εργαζόμαστε σε εικονοστοιχεία ανά καρέ, γράφουμε απλούστερα velocity_y = velocity_y - pixel_gravity.

Στη συνέχεια υπολογίζουμε τη νέα θέση της μπάλας: $x \leftarrow x + v_x$ $\kappa\alpha\iota$ $y \leftarrow y + v_y$. Το αρνητικό πρόσημο στη συνάρτηση του y συμβαδίζει με το ότι στον χώρο της εικόνας αυξάνεται προς τα κάτω.

```
velocity_y = velocity_y - pixel_gravity;
position_x = position_x + velocity_x;
position_y = position_y - velocity_y;
```

Ελέγχουμε αν η μπάλα «διαπέρασε» το έδαφος υπολογίζοντας το κάτω άκρο της, δηλαδή `bottom_of_ball = position_y + μισό_ύψος`. Αν αυτό το σημείο βρεθεί κάτω από το `ground_level`, θεωρούμε ότι έγινε σύγκρουση με το έδαφος. Επαναφέρουμε τη μπάλα ακριβώς στο έδαφος και αντιστρέφουμε την κατακόρυφη ταχύτητα πολλαπλασιάζοντάς την με τον παράγοντα `bounciness`, ώστε $v_y \leftarrow -v_y \times bounciness$. Με αυτόν τον τρόπο μοντελοποιούμε την απώλεια ενέργειας κατά την αναπήδηση.

```
bottom_of_ball = position_y + ball_height_scaled / 2;
    if bottom_of_ball > ground_level
        position_y = ground_level - ball_height_scaled / 2;
        velocity_y = -velocity_y * bounciness;
    end
```

Με παρόμοιο τρόπο, ελέγχουμε εάν η μπάλα βγαίνει εκτός των αριστερών ή δεξιών ορίων της εικόνας (το `1` και το `beach_width` αντιστοιχούν στα pixel όρια). Αν η μπάλα βγει εκτός, επανατοποθετούμε τη θέση της ακριβώς εντός ορίων και αντιστρέφουμε την οριζόντια ταχύτητα `velocity_x` (επίσης πολλαπλασιάζοντάς την με `bounciness` για πιθανή απώλεια ενέργειας).

Αφού ενημερώσουμε τη rotation_angle, κατασκευάζουμε τις επιμέρους Affine μετασχηματισμούς. Το Tcenter μετακινεί το κέντρο της εικόνας της μπάλας στη αρχή των συντεταγμένων (0,0). Το Tscale εφαρμόζει ομοιόμορφο scaling στην εικόνα. Το μητρώο R υλοποιεί την περιστροφή κατά rotation_angle μοίρες. Τέλος, το Tback επαναφέρει το κέντρο στην αρχική του θέση.

Ο συνολικός μετασχηματισμός προκύπτει από το γινόμενο

$$\text{transformMatrix} = T_{back} \cdot R \cdot T_{scale} \cdot T_{center}$$

Έπειτα, ορίζουμε τον μετασχηματισμό tform ως affine2d(transformMatrix) για να τον εφαρμόσουμε στην εικόνα και την μάσκα με imwarp.

```
rotation_angle = rotation_angle + rotation_step;

    cx = ball_width / 2;
    cy = ball_height / 2;

    Tcenter = [1 0 0; 0 1 0; -cx -cy 1];
    Tscale = [ scale_factor 0           0
               0           scale_factor 0
               0           0            1 ];
    R = [ cosd(rotation_angle)  sind(rotation_angle)  0
         -sind(rotation_angle)  cosd(rotation_angle)  0
          0                     0                     1 ];
    Tback = [1 0 0; 0 1 0; cx cy 1];

    transformMatrix = Tback * R * Tscale * Tcenter;
    tform = affine2d(transformMatrix);

    rotated_ball = imwarp(ball, ball_ref, tform, 'cubic', 'FillValues', 0);
    rotated_mask = imwarp(inverseMask, mask_ref, tform, 'cubic', 'FillValues', 0);
```

Υπολογίζουμε τις συντεταγμένες (σε pixels) όπου θα «κολλήσουμε» την περιστρεφόμενη μπάλα πάνω στην εικόνα του φόντου. Ορίζουμε το πάνω-αριστερό σημείο της μπάλας ως $(position\_x - \pi\lambda\acute{\alpha}\tau o\varsigma/2,\ position\_y - \acute{\upsilon}\psi o\varsigma/2)$ και το στρογγυλοποιούμε για να ταιριάζει σε ακέραιες τιμές pixel. Μετά, περιορίζουμε τις τιμές να μην ξεφεύγουν από τα όρια της εικόνας του φόντου. Αν ξεφεύγουν τότε μόνο αυτές που περιέχονται στις διαστάσεις του καμβά θα εμφανίζονται.

```
    top_left_x = round(position_x - size(rotated_ball,2)/2);
    top_left_y = round(position_y - size(rotated_ball,1)/2);

    x_start = max(1, top_left_x);
    y_start = max(1, top_left_y);
    x_end   = min(beach_width,  top_left_x + size(rotated_ball,2) - 1);
    y_end   = min(beach_height, top_left_y + size(rotated_ball,1) - 1);
```

Κόβουμε την περιστρεφόμενη μπάλα και τη μάσκα της στις αντίστοιχες διαστάσεις που θα επικαλύψουν την περιοχή της εικόνας φόντου.

Έπειτα, για κάθε χρωματικό κανάλι (R, G, B), συνδυάζουμε την αρχική περιοχή του φόντου με την μπάλα. Πιο συγκεκριμένα, region είναι η περιοχή φόντου, cropped_ball είναι η εικόνα της μπάλας και cropped_mask είναι η δυαδική μάσκα. Όταν η μάσκα έχει τιμή 1, προβάλλεται η μπάλα, ενώ όταν έχει τιμή 0, παραμένει το φόντο. Το αποτέλεσμα εγγράφεται πίσω στην εικόνα frame_image.

```
cropped_ball = rotated_ball(ball_y_start:ball_y_end, ball_x_start:ball_x_end, :);
 cropped_mask = rotated_mask(ball_y_start:ball_y_end, ball_x_start:ball_x_end);

 frame_image = beach;  % Start with the beach
 for c = 1:3
     region = frame_image(y_start:y_end, x_start:x_end, c);
     ball_overlay = double(cropped_ball(:, :, c)) .* cropped_mask;
     frame_image(y_start:y_end, x_start:x_end, c) = ...
         uint8(double(region) .* (1 - cropped_mask) + ball_overlay);
 end
```

Τέλος, γράφουμε κάθε ενημερωμένο καρέ στο αρχείο βίντεο μέσω της writeVideo. Το αποτέλεσμα που προκύπτει φαίνεται στο βίντεο transf_beach.

Ερώτημα 8

Το ερώτημα αυτό ακολουθεί αντίστοιχη λογική με αυτή του προηγούμενου ερωτήματος. Οι διαφορές μεταξύ των δύο ερωτημάτων είναι:

- Εισάγεται και μια νέα λειτουργία: η δυναμική κλιμάκωση της μπάλας. Συγκεκριμένα, ορίζουμε ότι η μπάλα ξεκινά με κλίμακα initial_scale=0.5 (δηλαδή μεγαλύτερη, καθώς είναι κοντά στην κάμερα) και σταδιακά μειώνεται σε μέγεθος καθώς κινείται προς το βάθος της σκηνής, φτάνοντας την final_scale=0.1

- Η οριζόντια θέση της μπάλας ορίζεται ως το κέντρο της οθόνης ($position\_x = beach\_width/2$ }), πράγμα που σημαίνει ότι η μπάλα δεν μετακινείται προς τα αριστερά ή τα δεξιά. Η κατακόρυφη κίνηση καθορίζεται από τη θέση y που ξεκινά πολύ κοντά στο κάτω μέρος ($position\_y = beach\_height - 150$) και από την αρχική κατακόρυφη ταχύτητα. Έτσι, η οριζόντια συνιστώσα της κίνησης έχει αφαιρεθεί, επιτρέποντας στην κίνηση να φαίνεται ότι πηγαίνει προς το βάθος της σκηνής μέσω της μεταβολής του μεγέθους.

```
position_x = beach_width / 2;     % Fixed horizontal position (centered)
position_y = beach_height - 150;  % Starting near the bottom
velocity_y = initial_velocity;    % Start upward
```

- Υπολογίζεται ο τρέχων παράγοντας κλίμακας που μεταβάλλεται γραμμικά από την αρχική τιμή προς την τελική καθ' όλη τη διάρκεια του animation. Ορίζοντας το $\alpha = \frac{frame}{num\_frames}$, το οποίο παίρνει τιμές από 0 έως 1, η τρέχουσα κλίμακα υπολογίζεται ως:

$$current\_scal = (1 - \alpha) \times initial\_scal + \alpha \times final\_scale$$

Έτσι, στην αρχή ($\alpha = 0$), η κλίμακα είναι 0.5 (η μπάλα εμφανίζεται μεγάλη), ενώ προς το τέλος ($\alpha \to 1$) η κλίμακα μειώνεται σε 0.1, κάνοντας τη μπάλα να φαίνεται μικρότερη και, κατά συνέπεια, πιο απομακρυσμένη.

```
alpha = frame / num_frames;  % progress from 0 to 1 over the animation
current_scale = (1 - alpha) * initial_scale + alpha * final_scale;
```

- Δημιουργείται affine μετασχηματισμός για να εφαρμόσει τόσο την αλλαγή κλίμακας όσο και, αν υπάρχει, την κατακόρυφη αναστροφή. Αρχικά, μετατοπίζουμε το κέντρο της εικόνας της μπάλας στην αρχή των συντεταγμένων μέσω του $Tcenter$. Έπειτα, το μητρώο $Tscale_$flip εφαρμόζει την τρέχουσα κλίμακα $current\_scale$ τόσο στον άξονα x όσο και στον άξονα y, όμως, στον άξονα y πολλαπλασιάζεται επιπλέον με το $flip\_factor$, το οποίο προσομοιώνει την περιστροφή της μπάλας καθώς κατευθύνεται προς το βάθος. Η εφαρμογή αυτού του μητρώου, σε συνδυασμό με το ότι η οριζόντια θέση παραμένει σταθερή, επιτρέπει στη μπάλα να αλλάζει μέγεθος (και άρα να φαίνεται πως κινείται στο βάθος) χωρίς να μετακινείται προς τα πλάγια.

```
% Translate the center to the origin
    Tcenter = [1 0 0; 0 1 0; -cx -cy 1];

    % Create a scaling matrix that also flips vertically when flip_factor
= -1
    Tscale_flip = [ current_scale, 0, 0;
                    0, current_scale * flip_factor, 0;
                    0, 0, 1 ];

    % Translate the center back to its original location
    Tback = [1 0 0; 0 1 0; cx cy 1];

    % Combined affine transform
    transformMatrix = Tback * Tscale_flip * Tcenter;
    tform = affine2d(transformMatrix);
```

Το flip_factor ορίζεται παρακάτω και εφαρμόζεται κάθε δύο frame για την κατακόρυφη αναστροφή της εικόνας.

```
flip_interval = 2;  % Number of frames before the ball flips
    flip_factor = (-1)^(floor((frame-1)/flip_interval) + 1);  % Change
exponent if you want to start unflipped
```

- Σε αυτό το τελικό τμήμα, η μετασχηματισμένη μπάλα, η οποία έχει πλέον υποστεί δυναμική αλλαγή κλίμακας και (πιθανή) κατακόρυφη αναστροφή, τοποθετείται πάνω στο φόντο της παραλίας με το κέντρο της να ευθυγραμμίζεται με τις τιμές $position\_x$ και $position\_y$. Δεδομένου ότι το $position\_x$ είναι σταθερό (στο κέντρο της οθόνης), η μπάλα δεν μετακινείται οριζοντίως.

Το αποτέλεσμα κάθε frame αποθηκεύεται στο βίντεο `transf_beach_depth.mp4`.