Augmented Reality
Assignment 2

Gareth Young, Binh-Son Hua
Trinity College Dublin

In this assignment, you will build an augmented reality system with hand tracking. The AR system will display live content from a video camera (e.g., the built-in or USB camera on your computer). The system will track the hands that appeared in the video in real-time and output the 3D coordinates of the hands based on a template of 21 landmarks. The system will use the predicted 3D coordinates of the hands to recognize hand gestures and enable interactions with a virtual 3D object, such as grabbing a box and moving it in the 3D space.

You will implement this system in Python with OpenCV and MediaPipe, as in Lab 6. Additionally, to facilitate the rendering of 3D objects, we will further employ moderngl, a modern Python wrapper over the OpenGL core. The basic idea behind implementing this system is to detect the hand landmarks in 3D and ensure that these landmarks match your hand movement in the video. This will allow us to detect when the hands touch the virtual object and, therefore, allow us to implement what real hands and virtual object interactions.

To help you implement this idea, this assignment is organized into five tasks:

Task 1. Implement hand tracking using OpenCV and MediaPipe.
The task is similar to lab 6, but instead of using a face landmark detector, we switch to a hand landmark detector. (2%)

Task 2. Display MediaPipe tracked hand landmarks in OpenCV. Make sure the projection of the 3D landmarks is aligned with the hands appearing in the OpenCV video. (4%)

This task is a bit involved due to the 3D hand landmarks output convention from MediaPipe. Particularly, there are two types of hand landmarks output from MediaPipe, stored in an instance of HandLandmarkerResult returned by the detector:

- hand_landmarks: 2D landmarks in image coordinates, normalized to [0, 1]. You will need to multiply the x and y values of the landmarks with the width and height of the video frame to obtain pixel coordinates of the landmarks. We refer to this result as the **image landmarks**.
- hand_world_landmarks: 3D landmarks in world coordinates, in meters. However, note that they are not absolute 3D landmarks in the world space. In fact, they are 3D landmarks defined in the model space relative to the origin at the geometric center of the hand. See https://developers.google.com/mediapipe/solutions/vision/hand_landmarker/python
  We refer to this result as the **model landmarks**.

The fact that MediaPipe does not return the absolute world coordinates of the 3D hand landmarks prevents us from directly using this output for implementing real-hand and virtual object interaction. The issue is that when your hands touch the virtual object in a rendered image, the 3D landmarks do not actually intersect with the virtual object, or there exists a mismatch between the video appearance and the 3D landmarks. This can be checked by projecting the 3D landmarks from MediaPipe back to the video frame using perspective projection, and it can be seen that the 3D landmarks do not match the hand.

So our goal in this task is to realign the 3D landmarks with the hands in the video. This realignment can be done by estimating a transformation matrix (representing 3D rotation and 3D translation) that transforms the **model landmarks** such that their projection matches the **image landmarks**. This can be done by an algorithm known as the perspective-n-points (PnP) problem in computer vision. OpenCV supports an implementation of this algorithm via cv2.solvePnP.

After calling solvePnP, we can construct a transformation matrix to transform the detected landmarks from the model space to the world space. We can perform a sanity check to verify if the projected landmarks match the image landmarks with reprojection error within a few pixels (~3-5 pixels). Note that we do not expect a perfect match because the transformation is only an approximation and we do not have exactly the camera parameters (e.g., focal length) to relate the 3D landmarks and the image landmarks.

Task 3. Explore moderngl examples, and then render the live video camera from OpenCV capture to OpenGL in Python. (1%)

We will use OpenGL to overlay virtual objects to the video captured from OpenCV. As an example, we will interact with a virtual cube at the center of the video with a distance of 30 units from the camera.

This task requires you to recall your OpenGL background in lab 1, lab 2 and assignment 1. The moderngl package provides an easy way to use modern OpenGL (from version 3.3) in Python. The source code of moderngl includes several examples that you can run easily to learn how moderngl works. See https://github.com/moderngl/moderngl/blob/main/examples/basic_colors_and_texture.py for an example on how to render 3D objects with texture mapping.

To display an OpenCV image (e.g., a video frame from the live capture in OpenCV) using moderngl, we will draw a rectangle that has a similar size to the window and then attach the OpenCV image as a texture to this rectangle. See this example: https://github.com/moderngl/moderngl/blob/main/examples/image_shader_example.py
For simplicity, you can set the OpenGL window size to the video frame size.

Task 4. Transfer MediaPipe tracked hand landmarks to OpenGL. Make sure the 3D landmarks are aligned to the hands appeared in the video in OpenGL. (4%)

In this task, to support gesture recognition afterwards, we have to make sure that the 3D landmarks from hand tracking matches the hands in the video. At this stage, recall that we have done this sanity check in OpenCV in task 2, and here we repeat this check for OpenGL by displaying the 3D markers at the locations of the world 3D landmarks. If the 3D markers match the 2D landmarks in the rendering then the transfer from OpenCV/MediaPipe to OpenGL is correct.

Task 5. Recognize gestures based on the 3D landmarks and implement virtual 3D object interaction. (4%)

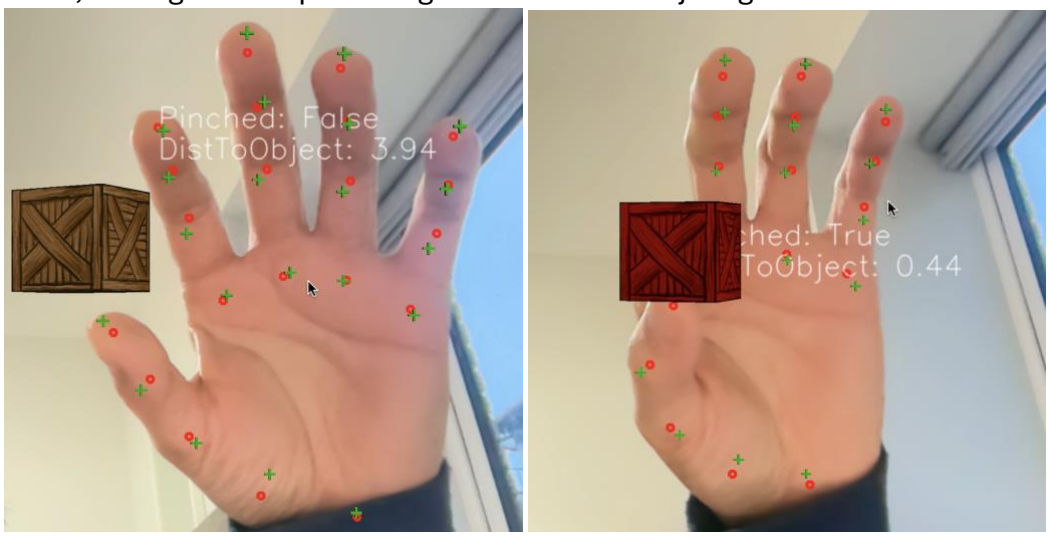
Once the landmarks are in place, it is now a good time to implement our AR application. Here is a suggestion of a real hand – virtual object interaction example.

- Implement a simple way to recognize pinching (when the index and thumb finger tip meets).
- Implement a simple way to check if object hit exists, i.e., the index finger touches the virtual cube.
- Implement a simple manipulation of the cube, e.g., move the cube when pinching and object hit occurs.

To walk you through the subtleties of this assignment, an example template code is provided. Fill in prediction.py for Task 1, 2, and gl.py for Task 3, 4, 5.

To run the example code, prepare a conda environment with the following packages:
$ pip install mediapipe
$ pip install moderngl moderngl-window pywavefront

Below are two example screenshots of the completed application. Left is the open hand, and right is the pinched gesture with the object grabbed.



The red circles are 2D landmarks, and the green plus signs are the 3D landmarks. The rendering is OpenGL.

Submission:
Package the following into assignment2_results.zip: (1) your Python files and any dependencies if needed. Do not package your conda environment or any pip packages, (2) an mp4 video that shows an example of the interaction of a real hand and the virtual cube. Crop the video to remove or blur any faces in the video.
On Blackboard, go to Submissions -> Assignment 2 and upload your zip file.

Deadline: Friday, Mar 21, 2025, at 11:59am (noon).

Marking:
Your final source code and mp4 video results will be assessed.

In addition to correctness, the following requirements for robustness need to be met:

- Both hands (left and right) are detected. Partial marks will be given if only one hand is supported.
- Real hand and virtual cube interaction must follow depth cues. Partial marks will be given if only 2D landmarks or inaccurate 3D landmarks (e.g., incorrect depth, severe mismatched reprojections) are used to control the object.
- The performance of the system is real time (> 10 fps). Partial marks will be given if your system is too slow to be usable as an interactive application.
- The code is well formatted, clean, and easy to read.