CS 153 Assignment 3

Due: 17 February, 2025

Advanced Image Composition

In the third assignment for CS153 in Spring 2025 we are going to extend the image composition techniques from Assignment 2 to incorporate additional skills and techniques we have learned. In this assignment you will be asked to build a number of additional helper functions that can be used in conjunction with a modified version of your `place_object` function from Assignment 2. The goal is to gain more control over the objects we place, and build in the functionality to add more objects taking into account depth plane information. Note that you may re-use your own code from Assignment 2 and/or from instructor solutions, and as a reminder you are free to use functions from libraries such as OpenCV and to define your own helper functions. All of your code should be written in `A3.py`, and submitted on Gradescope.

As with Assignment 2, this assignment contains a mixture of written questions (30 points) and coding questions (30 points). The coding questions are described here, and the written questions should be submitted directly on Gradescope. One written question depends on the output of your code.

Question 1: Green Screen Extraction

[14 points]

In Assignment 2 you were provided objects that already had a transparent background. For this question, you will need to write a function that can produce element and mask pairs from a green screen image. Please note that images are specified by filename, and so you are free to design specific extraction rules for individual images that you are provided, but your function should also have a general rule that it applies when it does not recognize the name of the input image (if you want to write your function with only a general rule, that is also fine!).

To complete this question, you will write `green_extract` according to the description given in the function declaration in `A3.py`. The function will take in a path for an image and return paired lists of content and transparency channel images, each representing a minimal bounding box over an objects you are able to find in the image.

You are provided with a number of green screen images. The seven that you will be evaluated on are in the `test_imgs` folder, and the others are included in the `custom_imgs` folder. You will be evaluated on your ability to extract high quality masks of the following:

- ball3.png
- ball_toss.png (two objects to find!)
- calden_falling.png
- calden_umbrella.png
- francine_poppins.png
- head.png
- head_smiles.png (five objects to find!)

To get full marks for this question, you should be able to generate good quality masks for at least 6 of the 7 test images provided (remember, you can construct custom rules!).

Question 2: Advanced Insert

[16 points]

To complete this question, you will create a new function for inserting objects into a scene that increases the available options for controlling object appearance: `affine_insert`. Here we will describe the conceptual elements of this function, but you should refer to the function descriptions in `A3.py` for detailed input and output descriptions.

One of the primary extensions is the introduction of `scene_depth` and `eldepth`. `scene_depth` is a map of the same spatial dimensions as the input scene that records a depth argument for every location in the scene, while `eldepth` specifies at what depth the inserted element should be placed. The inserted object should appear in a given pixel of the resultant composite scene *only* if `eldepth` is less than the current `scene_depth` for that pixel. For any pixel that has the object placed in its location, the returned `scene_depth` should be updated to be equal to `eldepth`. This should be updated even when the transparency value is set to something other than 1 (so long as it is greater than 0). It should also be possible to specify a `scene_depth` of `None`, in which case all depth values should default to `float('inf')`.

In addition to depth, affine insert adds the option of specifying an angle for the inserted object (in degrees), specifying the clockwise rotational position of the inserted object (note that we ignore shear and non-uniform scaling options, despite these also being forms of affine transformations).

Note 1: The `imutils` library has some convenient functions for doing simple transformations, including rotations. I find it much easier to use than OpenCV, and recommend it. Specifically, I recommend you use `imutils.rotate_bounded` rather than `imutils.rotate`, as the latter will potentially cut off image content due to the rotation.

Note 2: Many of the calculations used in this question work best over float values, but your functions expect to take in and return integer images. A standard practice in image processing that I recommend getting used to is converting to float, doing all your calculations, and then converting back to integer for the output.

Note 3: You can develop this question without having completed Question 1 by simply using the images provided in Assignment 2 as your elements and masks (just remember to compute the minimal bounding box, since we expect `green_extract` to do that)!

Written Question: Putting it all Together

[5 points]

Now it's time to use the functions you've built to make something cool! Write a function `custom_compose` that will provide a recipe to output an interesting composition using the tools we've built. You are free to use any of the images we have provided, or to provide your own. You will submit your custom composition on Gradescope along with the other written questions.