On the structure of this practise, on how to submit the solutions and on how to report errors found in this practise

Structure

This practise consists on exercises identified by number. You need to upload in canvas **one single `.zip` file** with the filename:

{`your-full-name`}`-graded-practise-4.zip`.

If the following instructions are not properly followed, the automatic grader will give you 0 **points**

This `zip` file should contain **only** directories named `exercise01`, `exercise02`, ... **Notice that** if you compress *the directory containing these directories*, your `.zip` will not stick to the correct structure and will receive `0` points.

For example, imagine that you have your directories `exercise01`, `exercise02`, ... inside a directory named `pc4`. If you right-click on top of `pc4` and choose *compress*, **the zip file that you will generate will be incorrect and you will receive** `0` **points**. The reason of this is: your `zip` file will contain, inside it, the directory `pc4` *which will contain, inside it*, the directories `exercise01`, `exercise02`, ... **And this should not be in this way. The directories** `exercise01`, `exercise02`, ... **should be at the root level inside the** `zip` **file, without having the *container* directory named** `pc4` **(or any other name)**.

The way to achieve this effect is simple: just select with your mouse *all the directories* `exercise01`, `exercise02`, ... (NOT the directory containing them, but all the directories), press the right button of your mouse and select *compress*. This procedure will generate a `zip` file that will contain the exercises in the root level.

Note: not following the correct directory structure will result in a grade of `0` **points.**

Convention for the names of the files containing the solutions for the exercises.

Inside the directory `exercise01` must be a file named `solution.py` that must implement the function requested in the description of the exercise 1, inside the directory `exercise02` must be a file named `solution.py` that must implement the function requested in the description of the exercise 2, and so on. **The filename must follow this convention in a very rigorous manner. Solutions that does not follow this convention will**

receive 0 points in the exercise.

How will the evaluation work

Each problem will be automatically judged through the usage of automatic unit tests written in python.

How to report errors that are found in the present document

If you find any error (either small or big) in the present document, please report it via Slack private message to Eric Biagioli.

Problems (Total: 100 points)

Camera model.

1. (10 points) Project a cloud of points.

Description

Implement the function

```
project_points(
  full_path_input_mesh,
  optical_center_x,
  optical_center_y,
  optical_center_z,
  optical_axis_x,
  optical_axis_y,
  optical_axis_z,
  output_width_in_pixels,
  output_height_in_pixels,
  full_path_output
)
```

that *takes a photo* of the vertices present in the mesh described in `full_path_input_mesh`. Assumptions:

- the optical center (this is: the pinhole) of the camera is located at `optical_center`
- the distance between the optical center and the projection plane is `focal_distance`
- the *principal point* (this is: the intersection between the optical axis and the projection plane) is perfectly centered in the projection sensor
- similarly to what we did in the PC3, we will assume that the pinhole is *behind* the projection plane (this is: a positive focal distance means that the projection plane is *closer* to the point being projected than a negative focal distance).

Parameters

- `full_path_input_mesh`: full path containing the input mesh. Only the vertices (and their colors) will be used in this exercise.
- `optical_center_{x, y, z}`: parameters indicating (in world coordinates) the position of the optical center of the camera.

- `optical_axis_{x, y, z}`: parameters indicating (in world coordinates) the direction of the optical axis. Notice that this vector is normal to the projection plane.
- `focal_distance`: from the optical center, a translation of exactly *this* amount in the direction given by `optical_axis` arrives in a point in the projection plane (specifically: in the *principal point*.
- `output_width_in_pixels`: width of the output, specified in pixels.
- `output_height_in_pixels`: height of the output, specified in pixels.
- `full_path_output`: full path of the output.

Examples

```
project_points(
  full_path_input_mesh='/home/someone/sphere-rectangles.off',
  optical_center_x=5.0,
  optical_center_y=5.0,
  optical_center_z=5.0,
  optical_axis_x=0,
  optical_axis_y=0,
  optical_axis_z=-1.0,
  output_width_in_pixels=1920,
  output_height_in_pixels=1080,
  full_path_output='/home/someone/projection-1.png'
)
```

Constraints

- Time constraint: the proposed solution has to run in less than 1 second in all the test cases in which it will be tested.
- All the test cases that will be used will be `OFF` meshes with no more than 1 million vertices, potentially with colors.

Other remarks

- In the Canvas platform there are examples of inputs that will be used during the automatic grading of this problem. It is adviced to use them during the implementation of the problem.

2. (10 points) Generate an animation

Description

Implement the function

```
sequence_of_projections(
  full_path_input_mesh,
  [optical_center_x],
  [optical_center_y],
  [optical_center_z],
  [optical_axis_x],
  [optical_axis_y],
  [optical_axis_z],
  output_width_in_pixels,
  output_height_in_pixels,
  prefix_output_files
)
```

that uses the previous exercise to take *a sequence of photos* of the vertices present in the mesh described in `full_path_input_mesh`. Everything is exactly the same as in the previous exercise, but the parameters enclosed by square brackets are *lists*. The assumptions made in this exercise are the same as the ones made in the previous exercise. The files produced as output must be `{prefix_output_files}-1.png`, `{prefix_output_files}-2.png`, ...

Parameters

The parameters are the same as in the previous exercise, with the difference that the parameters that are surrounded by square brackets are lists.

Examples

```
project_points(
  full_path_input_mesh='/home/someone/sphere-rectangles.off',
  optical_center_x=[4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5.0],
  optical_center_y=[5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0],
  optical_center_z=[5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0],
  optical_axis_x=[0,0,0,0,0,0,0,0,0,0],
  optical_axis_y=[0,0,0,0,0,0,0,0,0,0],
  optical_axis_z=[-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0],
  output_width_in_pixels=1920,
  output_height_in_pixels=1080,
  prefix_output_files='/home/someone/cloud_of_points_moving_camera'
)
```

```
-> Should produce the files:
/home/someone/cloud_of_points_moving_camera-1.png
/home/someone/cloud_of_points_moving_camera-2.png
/home/someone/cloud_of_points_moving_camera-3.png
/home/someone/cloud_of_points_moving_camera-4.png
/home/someone/cloud_of_points_moving_camera-5.png
/home/someone/cloud_of_points_moving_camera-6.png
/home/someone/cloud_of_points_moving_camera-7.png
/home/someone/cloud_of_points_moving_camera-8.png
/home/someone/cloud_of_points_moving_camera-9.png
/home/someone/cloud_of_points_moving_camera-10.png
```

Constraints

- Time constraint: the proposed solution has to run in less than N seconds in all the test cases in which it will be tested, where N is the number of frames generated.
- All the test cases that will be used will be `OFF` meshes with no more than 1 million vertices, potentially with colors.

Other remarks

- In the Canvas platform there are examples of inputs that will be used during the automatic grading of this problem. It is adviced to use them during the implementation of the problem.

Augmented reality.

3. (20 points) Simple example of augmented reality

Description

The idea of this exercise is to implement a simple marker-based augmented reality program. Use as a marker the simple QR code that was used in class. Draw a mesh with the correct pose and size in the place where you detected the marker.

You are expected to implement the function

```
draw_mesh_on_top_of_marker(
  full_path_input_image,
  full_path_mesh,
  full_path_output_image
)
```

that draws the input mesh (with simple cosine illumination, assuming that the faces are white) on top of the detected marker, **such that the base of the bounding box of the mesh matches exactly the position of the detected marker**.

Examples

```
project_points(
  full_path_input_image='/home/someone/input-1-augmented-reality.png',
  full_path_mesh='/home/someone/gargoyle-10k-faces.off',
  full_path_output_image='/home/someone/output-1-augmented-reality.png'
)
```

Constraints

- Time constraint: the proposed solution has to run in less than 15 seconds.
- All the test cases that will be used will be `OFF` meshes with no more than 10k faces.

Other remarks

- In the Canvas platform there are examples of inputs that will be used during the automatic grading of this problem. It is adviced to use them during the implementation of the problem.

Image Stitching.

4. (20 points) Simple example of image stitching

Description

Use OpenCV's image stitcher, and create an image stitcher that is able to create a panorama from a set of input images.

You are expected to implement the function

```
stitch_images(
  [full_path_input_image],
  blender={FeatherBlender | MultiBandBlender},
  features_finder={AKAZE | ORB | SIFT | SURF},
  features_matcher={AffineBestOf2Nearest | BestOf2NearestRange},
  warper={  Affine | CompressedRectilinearPortrait | CompressedRectilinear
```

```
          | Cylindrical | Fisheye | Mercator | PaniniPortrait | Panini
          | Plane | Spherical | Stereographic | TransverseMercator },
  full_path_output_image
)
```

that performs an stitching of the images specified in the list `[full_path_input_image]`, and generates the image `full_path_output_image` using the specified options.

Note: you are allowed to use the built-in image stitcher of OpenCV.

Examples

```
stitch_images(
  full_path_input_image=[
    '/home/someone/panorama1-input-1.jpg',
    '/home/someone/panorama1-input-2.jpg',
    '/home/someone/panorama1-input-3.jpg',
    '/home/someone/panorama1-input-4.jpg',
    '/home/someone/panorama1-input-5.jpg',
    '/home/someone/panorama1-input-6.jpg',
  ],
  blender=MultiBandBlender,
  features_finder=SIFT,
  features_matcher=BestOf2NearestRange,
  warper=Mercator,
  full_path_output_image='/home/someone/panorama1-mercator.jpg'
)
-> Produces an output with mercator projection
```

Constraints

- Time constraint: the proposed solution has to run in less than 30 seconds.
- All the test cases will consist of no more than 8 input images, each of them being not larger than 1920×1080px.

Other remarks

- In the Canvas platform there are examples of inputs that will be used during the automatic grading of this problem. It is adviced to use them during the implementation of the problem.

Image recognition.

5. (20 points) Using YOLO

Description

Use YOLO v7 to count how many persons, cars and bikes are present in an input video. The video will be approximately 25 frames per second, 5 minutes, 1920x1080px. The execution time should not be larger than 5 minutes in the computers of the laboratory L507.

You are expected to implement the function

```
count_people_cars_and_bikes( full_path_input_video )
```

that returns **a list of 3 elements, in which the first element is the number of people detected in the video, the second element is the number of bikes detected in the video and the third parameter is the number of cars detected in the video**

Examples

```
result = count_people_cars_and_bikes(
  full_path_input_video=['/home/someone/example-input-yolo-1.mpg',
)
-> The output result should be like this:
result[0] --> number of people detected in the video
result[1] --> number of bikes detected in the video
result[2] --> number of cars detected in the video
```

Constraints

- Time constraint: the proposed solution has to run in less than 5 minutes.
- All the test cases will be a video of no more than 5 minutes of duration, with a framerate not larger than 25 fps, with resolution not larger than 1920x1080 px.

Other remarks

- In the Canvas platform there are examples of inputs that will be used during the automatic grading of this problem. It is adviced to use them during the implementation of the problem.

6. (20 points) Using SAM

Description

Use YOLO and SAM (the META's model for segmentation, named *Segment Anything Model*, (SAM)) in order to highlight cars, people and bikes in an image.

You are expected to implement the function

```
highlight_people_cars_and_bikes(
  full_path_input_image,
  color_scale_image,
  color_scale_people,
  color_scale_cars,
  color_scale_bikes,
  full_path_output_image
)
```

- The image should be converted to scale of `color_scale_image`.
- All the people (and only the pixels corresponding to the people) in the image should be converted to `color_scale_people`.
- All the cars (and only the pixels corresponding to the cars) in the image should be converted to `color_scale_cars`.
- All the bikes (and only the pixels corresponding to the bikes) in the image should be converted to `color_scale_bikes`.

Examples

```
highlight_people_cars_and_bikes(
  full_path_input_image='/home/someone/example-1.jpg',
  color_scale_image=(255,255,255),
  color_scale_people=(255,0,0),
  color_scale_cars=(0,255,0),
  color_scale_bikes=(0,0,255),
  full_path_output_image='/home/someone/detections-example-1.jpg'
```

Constraints

- Time constraint: the proposed solution has to run in less than 5 seconds.
- All the test cases will consist of an image with no more than 50 objects in total that should be detected.

- The resolution of the input image will not be larger than `1920x1080` px.

Other remarks

- In the Canvas platform there are examples of inputs that will be used during the automatic grading of this problem. It is adviced to use them during the implementation of the problem.