CS 763/CS 764: Lab Six

Is this for Real?

- Announced: 3/16, Due: March 3/23, at 17:00

1 Introducing AR

Augmented Reality is the overlaying of digital (synthetic) data onto the real world. In a typical first person view, we will pretend that a robot is looking at a scene, and an object with specified dimensions in the real world will be placed synthetically on the view seen by the robot. If, instead of a robot, you are wearing a head mounted display then you will see the object overlaid on the scene as if the object existed at the location specified by the movie director.

The key to making the object look realistic is to paint the object with the exact proportions it would occupy were the object really in the scene.

Specifically in this problem a book will be overlaid on a plane in an image, such that it is perceived as if the book actually lies in the scene. This requires the knowledge of the camera for projecting the book from a particular viewpoint.

See Figure 1 for examples.

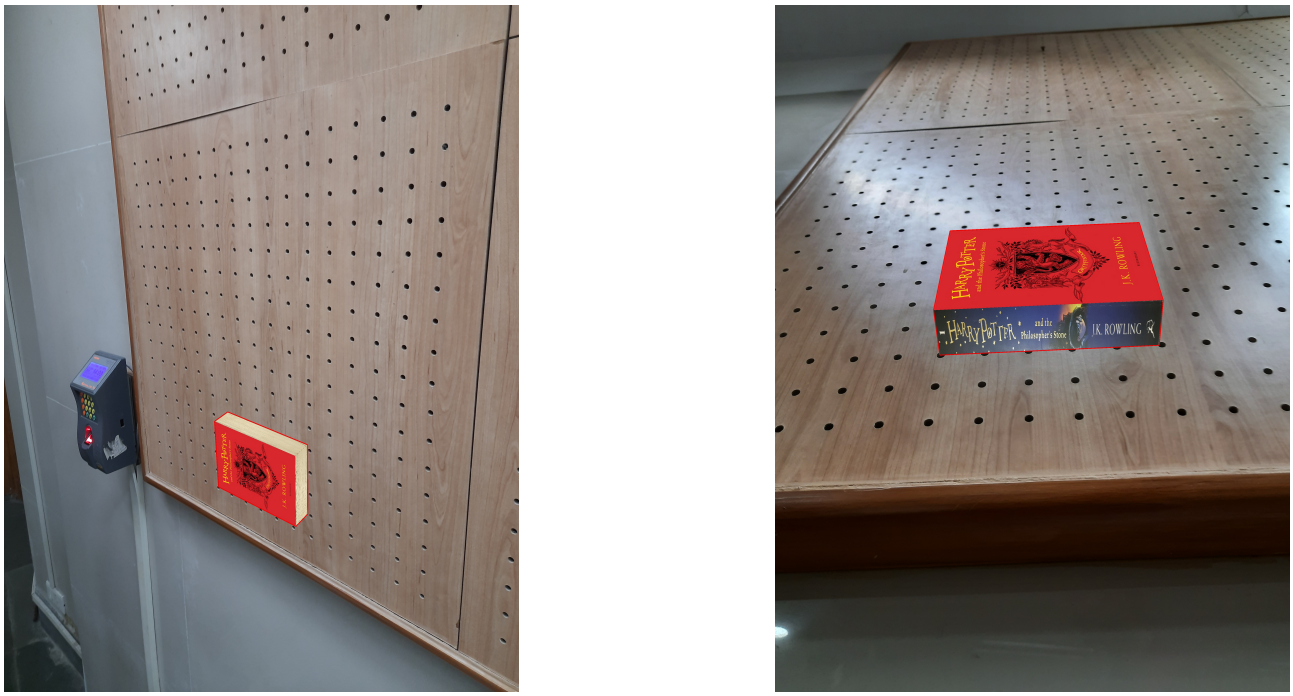

Figure 1: In the real world, it is impossible for a book to lie suspended in space like a cantilever. Left: We are able to see only the front cover of the book. Right: An ant on the floor is able to see the side cover.

1.1 Tasks

In this lab, a "book" is specified by three dimensions. We refer to the longest dimension as "length", the shortest as "width", and the remaining as "breadth". This information is to be specified in the command line argument.

1. All necessary data (input and output) for this problem is to be stored in `data/augment_book/`.

2. Take at least four different views of a typical "wall" (i.e., vertical structure) in your immediate environment.

   We will judge your book by the cover, so we provide two textures of a book in `front.png` and `side.png`.

   Read the data in your program. We would like to see a `python` function `get_data` for this.

3. One specific image. We want to "place" the virtual book sticking (and floating) on the wall for an upcoming Harry Potter movie. A picture Fig. 2 is provided for reference. Notice that the approximate layout of how the book is sticking to the wall. The "back cover" (last page) is stuck to the "wall". The front cover is hanging like a cantilever. The book is aligned parallel to the holes of the wall, so that one can read the text if one were to fly and rotate ourselves.

   Draw the boundary edges of the implicit cuboid corresponding to the book on each of the provided images. The boundary must be in red, and make sure to use a thick pencil.

   We would like to see a `python` function `draw_book_on_img` for this.

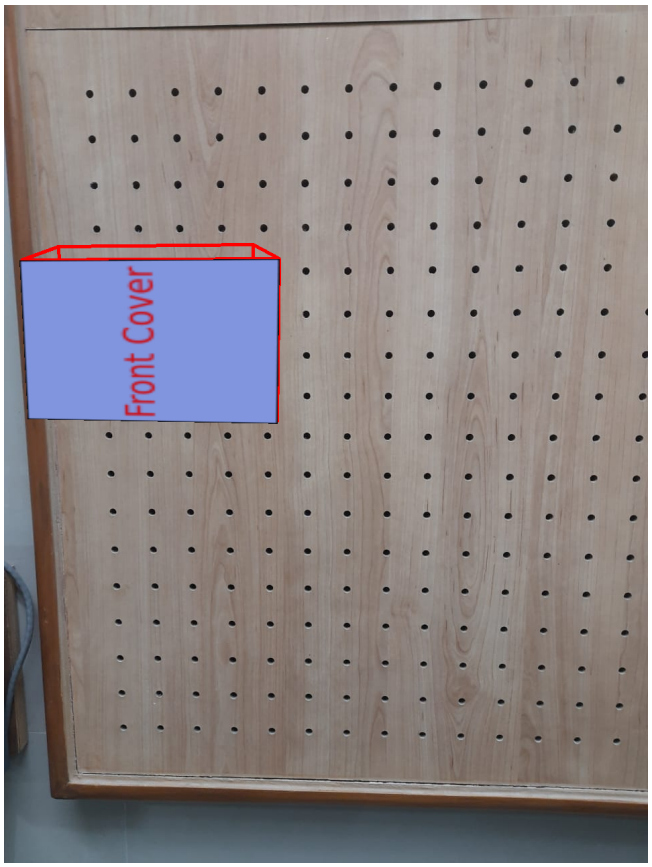

Figure 2: A book anchored to the wall like a cantilever.

   In this part, you are free to assume (and hard code) the location where the director has decided to put the box, although we have fixed the orientation of the book. Your location is likely to be different from that of another group submission. However, once fixed in one view, it must be fixed in another view at the same location, i.e., the book is anchored in the real world as shown.

   Hint: One way of doing this is to use `cv.solvePnP`.

4. Multiple Specified Images. Currently there is a single view of the book. In fact, the side plane of the book is occluded, and we can't see it! Fixing the book at exactly the same location, provide 3 other views of the book: the constraint here is that the camera is placed at sufficiently different

locations. (If, in the example provided, the camera was at a different location (for instance if the camera man was squatting and looking up) the side would have been visible.)

Q1.1 Have the choices of views been made so that all the surfaces (except the back-cover) visible in at least one view?

5. Your images. Next, we want you to place the book in your world. For this part, the TA is going to be the ultimate director with a random mood, so enable the TA to place the book by asking for the position of the book and orientation. Allow the TA (i.e., the user of your program) to specify the position of the center of the base of the book and the angle made by the book's spine (binding) with the real-world (not the camera) horizontal as command line parameters.

   The position of the center is specified as a pair of relative coordinates between 0% to 100% (w.r.t. both axes), and rounded to the nearest pixel positions. The angle from the horizontal is specified in degrees.

   To be clear, an angle of 0° means the spine (or binding) face is the bottom most plane, only visible by crouching, similar to Figure 1 (right).

6. Adding Realism. Use the provided texture images to "stick" the texture of the front and side to make the scene more realistic. Save the images on disk. This should work regardless of how the book is specified. However, if you can't get this to work, then at least show this for the "hard-coded" position in Step 3.

   Feel free to exercise your creativity by inventing your own textures or editing pictures from the internet for the different planes of the book.

   We would like to see a `python` function `texture_book_on_img` for this.

   Q1.2 How did you texture the book in the various views? If we were to provide you with a new view, or rotate the book by 90 degrees will your code be able to render the final result correctly? We are interested in making sure that the surfaces that would be visible for a real book are consistent with the ones visible in AR.

1.2 Run

Implement both parts (i.e., 'hardcoded' and TA-directed) with a single code file that has optional command line parameters – the latter enables the user to pick the book's position and orientation. The code for both the parts should display all `textured` images (with a suitable boundary visible).

The program for the first "self-directed" part is executed similar to

```
python3 code/augment_book.py data/augment_book -w 1.5 -l 5 -b 3
```

The code for the latter "TA-directed" part is executed similar to

```
python3 code/augment_book.py data/augment_book -w 1.5 -l 5 -b 3 -x 60 -y 40 -theta 45
```

2 Mirror, mirror on the wall

Mirror, mirror on the wall, who's the smartest of 'em all? This section is extra credit. In other words, you will not lose any marks if you do not complete this part. On the other hand, if you do complete this part, then you may be able to trade off some other work (say, in the deep learning parts which are heavily experimental in nature) for this part.

2.1 Adding more realism

At this point, the book is really there on the wall. Sort of. That's because the geometry is right. But the book would normally cast a shadow, for example were it there. So in this part, we want to add to the realism.

1. Synthetic Mirror. Repeat the book overlay with a reflective mirror replacing the opaque wall surface. Work with a big (long or wide) mirror, such as one in front of a basin or one attached to a dressing table. This is not likely to be very real, but basically instead of shadows, we want to see the reflection of the book.

2. Real Mirror. Sigh. Mirrors cause all kinds of problems to a poor camera. In this part, click pictures of an actual mirror from any view point. Repeat the experiment.

   Store all necessary data (input and output) for this problem in `data/augment_book_mirror/`.

Q2.1 Explain how you obtained the reflection of the book in image-space in a manner that was consistent with the Physics of reflection?

Note: While virtually placing the book on the real mirror and while synthesizing its reflection, you might end up with odd looking scenes left over in the mirror (an occluded camera person and camera for instance). This is fine for the purpose of this exercise.

2.2 Run

On running the code, all the textured images with a mirror as a backdrop should be displayed. The program for pre-specified book position will be executed as,

```
python3 code/augment_book_mirror.py data/augment_book_mirror -w 1.5 -l 5 -b 3
```

and the user directed variant will have the option to pass `x, y`, and `theta` as in Section 1.

3 Submission Guidelines

We are not repeating this. You should know the drill by now. Be sure to follow the explicitly or implicitly recommended folder structure.