CS 498: Assignment 1: Perspective Projection

Due by 5:00pm Friday, February 11 2022

January 28, 2022

Submission

In this assignment you will be modifying the files homography.py perspective.py. Please put together a single PDF with your answers and figures for each problem, and submit to Gradescope (Course Code: JBXJVZ). We recommend you to add your answers to the latex template files we provided. For code submission, make sure you use the provided ".py" files with your modification and the dumped ".npz" file. The graders will check both your PDF submission and your code submission if needed.

Homography [8pts]

In this question, we will examine properties of homography transformations and see how to estimate a planar homography from a set of correspondences.

Question 1 [1pt]: You are given a photo of the State Farm Center (`uiuc.png`), UIUC's indoor arena that hosts our basketball teams. We marked and plotted the four corners of the court, as shown in Fig. 1 (left). A standard basketball court is 28.65 meters long and 15.24 meters wide, as shown in Fig. 1 (right). Now let's consider a 2D coordinate system defined on the planar surface of the basketball court. The origin of this coordinate system is point #3; the long edge is x axis and short edge is y axis. Please write down the four corner's coordinate in this 2D coordinate frame (in meters) and fill in your answer in the numpy array "corners_court".

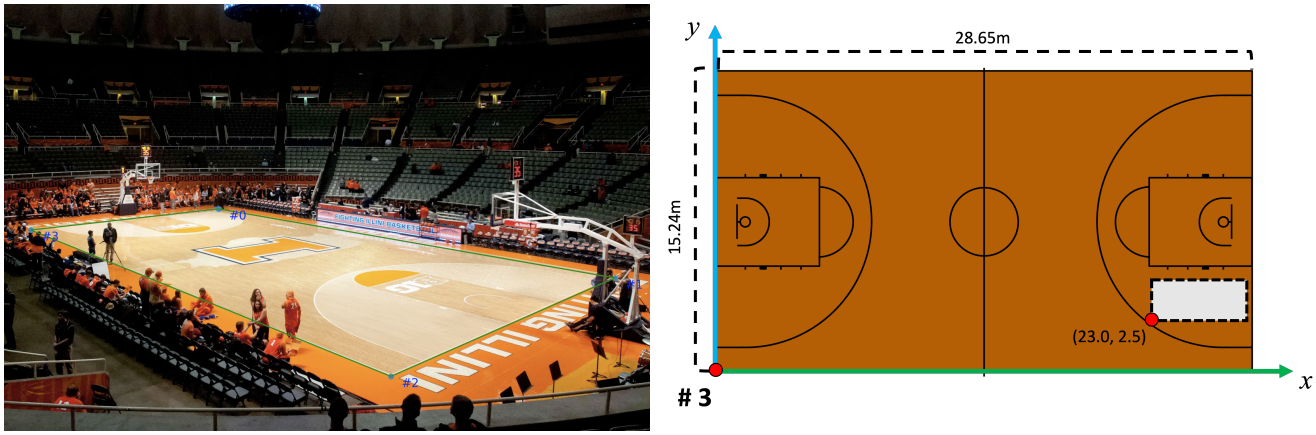

Figure 1: Left: target image with keypoints; Right: court dimensions and coordinate convention.

Answer:

The four corners' coordinates in the given 2D frame are represented as below:

#0: (0, 15.24), #1: (28.65, 15.24), #2: (28.65, 0), #3: (0, 0)

These coordinates are filled in a numpy array `corners_court` as below:

```
corners_court = np.array([(0, 15.24),
                          (28.65, 15.24),
                          (28.65, 0),
                          (0, 0)])
```

Question 2 [3pts]: Given your Q1's answer, now we could establish four pairs of point correspondence between two planes: the 2D basketball court plane and the 2D image plane. Using what we have learned in Lecture 3, complete the function `findHomography` and use it to estimate the homography matrix from the court to the image. Please briefly explain what you do for homography estimation and report the resulting matrix in this document. Hints: 1) you might find the function `vstack, hstack` to be handy for getting homogenous coordinate; 2) you might find `numpy.linalg.svd` to be useful; 3) lecture 3 described the homography estimate process.

Answer:

Below is the implemented `cv2.findHomography` function:

```
def findHomography(pts_src, pts_dst):
    N, _ = pts_src.shape

    A = np.zeros((2*N, 9))
    for i in range(N):
        Xi = np.append(pts_src[i], 1)
        A[2*i, 3:6] = Xi
        A[2*i+1, 0:3] = Xi

        A[2*i, 6:9] = - pts_dst[i][1] * Xi
        A[2*i+1, 6:9] = - pts_dst[i][0] * Xi

    U, S, Vh = np.linalg.svd(A)
    h = Vh[-1]
    H = h.reshape(3, 3) / h[-1]

    return H
```

The algorithm follows the formulation in Lecture 3. Assuming that N point pairs are given, a $2N \times 9$ matrix `A` is constructed, where every odd and even rows are filled with the appropriate values for each point pair. The flattened homography matrix `h` is then obtained by solving a linear equation `Ah = 0`, where `h` satisfies the column vector corresponding to the minimal singular value in the right unitary matrix `V` of `A`'s singular value decomposition `USV'`. Then a vector `h` is reshaped into a 3×3 matrix `H` normalized by `H[2,2]` to comply with the result of the off-the-shelf function `cv2.findHomography`.

The resulting homography matrix compared with that of `cv2.findHomography` is as follows:

```
>>> H = findHomography(corners_court, keypoints_im)
>>> print("H_ours:\n", H)
H_ours:
    [[ 1.20809640e+01  4.44696370e+01  7.92073117e+01]
    [-8.05770293e+00  5.76174339e+00  6.42252451e+02]
    [-2.12530603e-02  1.65326163e-02  1.00000000e+00]]

>>> H_cv2, _ = cv2.findHomography(corners_court, keypoints_im)
>>> print("H_cv2:\n", H_cv2)
H_cv2:
    [[ 1.20809613e+01  4.44696378e+01  7.92073135e+01]
    [-8.05770579e+00  5.76174634e+00  6.42252441e+02]
    [-2.12530631e-02  1.65326169e-02  1.00000000e+00]]
```

It is observed that the implemented `findHomography` function returns the homography matrix same as that of `cv2.findHomography`.

Question 3 [4pts]: We want to promote our CS498 class across the university. One of the marketing idea we came up is to create an on-court ad in Illinois's State Farm Center. Now you are taking in charge of the virtual signage task – inserting the logo of our class (`logo.png`) electronically onto the basketball court (`court.png`). Specifically, the size of the logo needs to be 3x6 meters; we want to place the bottom left logo corner at $(23, 2)$ on the basketball court. In order to do so, we need two steps:

- **3.a [1pt]:** calculate the homography transform between the image coordinate of the two images `logo.png` and `court.png`. Hints: 1) could you compute the transform from `logo.png` to the basketball court 2) could you leverage the homography matrix you computed in Question 2?

Answer:

Below is the code establishing the homography transformation from a 2D image `logo.png` to the 2D plane `court.png`:

```
h, w, _ = logo.shape
# be care of axis convention in image coordinate
corners_logo_img = np.array([(0, 0), (w, 0), (w, h), (0, h)])
corners_logo_court = np.array([(23, 5), (29, 5), (29, 2), (23, 2)])

H_logo_court = findHomography(corners_logo_img, corners_logo_court)
H_court_target = findHomography(corners_court, keypoints_im)

target_transform = H_court_target @ H_logo_court
```

`corners_logo_img` contains the four corners of `logo.png`—starting from top-left and clockwise as we did from #0 to #3—in pixel coornidates and `corners_logo_court` expresses their corresponding locations on the court (where the banner will be appeared) in metric coordinates. Behold that the axis convention of `corners_logo_img` is different from that of `corners_logo_court`, as pixel coordinates start from the top of an image. Then, the homography from the orignal image (`logo.png`) to the 2D-court is established by applying `H_logo_court = findHomography(corners_logo_img, corners_logo_court)`.

`H_court_target = findHomography(corners_court, keypoints_im)` governs the homography transformation from the 2D-court to the image (`uiuc.png`) on which the banner will be projected. The resulting transform from `logo.png` to `uiuc.png` is then obtained by multiplying these homography matrices sequentially:

```
target_transform = H_court_target @ H_logo_court
```

- **3.b [2pt]:** complete the `warpImage` function and use the homography you computed in 3.a to warp the image `logo.png` from its image coordinate to the `court.png`'s image coordinate. Hints: 1) suppose $(x', y', 1)^T = \mathbf{H}(x, y, 1)^T$, we have $I_{\text{target}}(x', y') = I_{\text{source}}(x, y)$; 2) you might find `numpy.meshgrid` and `numpy.ravel_multi_index` to be useful.

Answer:

Below is the function warping an input image `image` into the shape of `shape` upon the homography `H`:

```
def warpImage(image, H, shape):
    Hs, Ws, _ = image.shape
    Wt, Ht = shape
    image_warped = np.zeros((Ht, Wt, 4))

    Xs, Ys = np.meshgrid(np.arange(Ws), np.arange(Hs))
    ps = np.array([Xs.flatten(), Ys.flatten(), np.ones(Xs.size)]).T
    pt = (H @ ps.T).T
    pt /= np.expand_dims(pt[:,-1], axis = 1)
    pt = pt.astype(np.int32)
    Xt = pt[:, 0]; Yt = pt[:, 1]
    image_warped[Yt, Xt] = image.reshape(-1, 4)

    return image_warped
```

`ps` is the array of pixel coordinates (in homogeneous form) in the source image (`image`). `pt`, the array of corresponding pixel locations on the target image (`image_warped`), is then obtained by applying the homography matrix `H` to `ps`, with rounding to the closest integers. The target pixels are then colored with the source values and the output `image_warped` is returned.

The warped `logo.png` to the target location on the court of `uiuc.png` is in Fig. 2.

- **3.c [1pt]:** alpha-blend the warped logo onto the court: $I = \alpha F + (1 - \alpha)B$

Answer:

Below is the code blending the warped `logo.png` and the target `uiuc.png`:

```
alpha_logo = logo_warp[:,:,-1]
alpha_logo = np.expand_dims(alpha_logo, axis=-1)
im[:,:] = logo_warp[:,:] * alpha_logo + im[:,:] * (1 - alpha_logo)
```

The alpha values—the extent to which each pixel is weighed when blending—are in the last channel of each pixel location. `alpha_logo` reads those of the warped logo `logo_warp`.

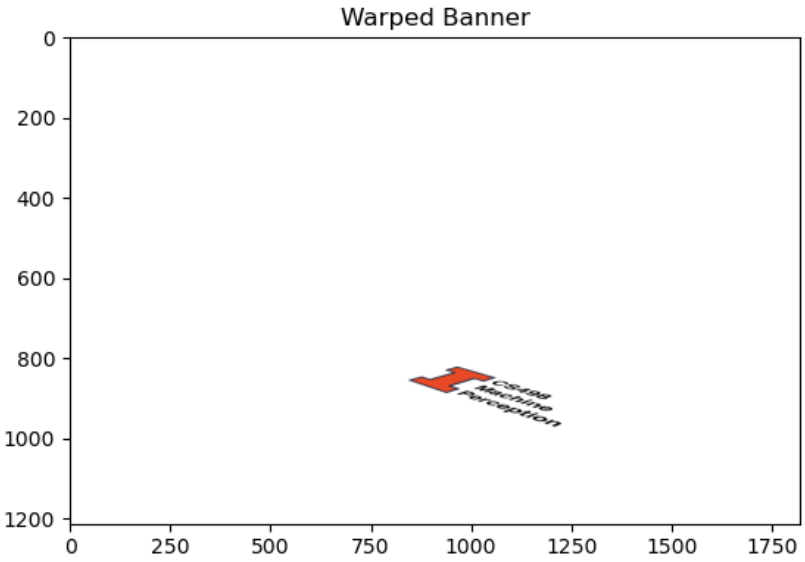

Figure 2: The warped `logo.png`.

Then, the blending of the background (`im`) and the logo (`logo_warp`) is done for every pixel.

The blended outcome is in Fig. 3.

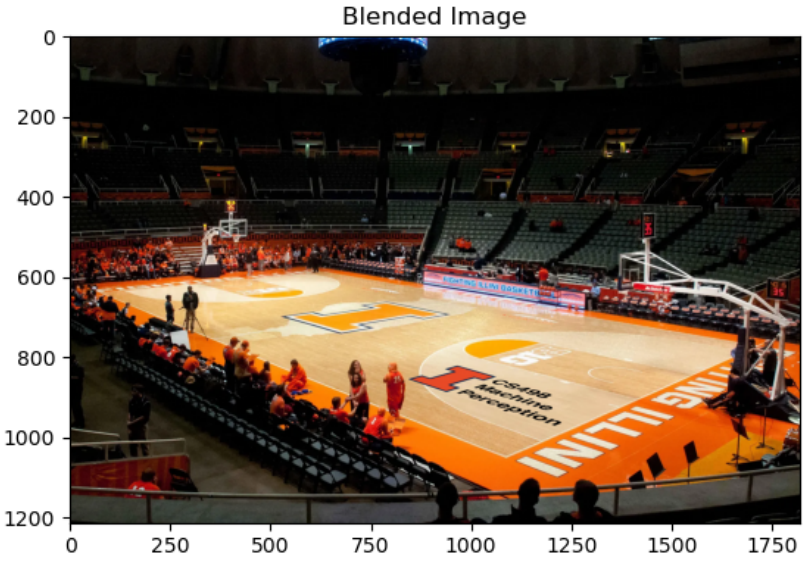

Figure 3: The blended image.

Perspective Projection [7pts]

Till now we have been working on transformations between 2D coordinate systems. Now let us to lift up to the 3D coordinate. Consider the same image of the state farm center, but this time, we annotate four additional points, corresponding to the four corners of the basketball backboard.

Question 4 [1pt]: The lower rim of the backboard is 2.745 meters above the ground; the backboard width is 1.83 meters and backboard height is 1.22 meters. The backboard protrudes 1.22 meters out from the baseline. Now let us consider the world frame. The world frame coordinate origin is at the point #3, where x-axis is along the long edge (sideline) and y-axis is along the short edge (baseline), and z-axis is upwards, perpendicular to the basketball court. Could you compute the 3D coordinate of all the eight keypoints?

Answer:

The eight corners' coordinates—four on the court and the others on the rim, starting from top-left and rotates clockwise—in the given 3D coordinate convention are represented as below:

#0: (0, 15.24, 0), #1: (28.65, 15.24, 0), #2: (28.65, 0, 0), #3: (0, 0, 0),
#4: (1.22, 6.705, 3.965), #5: (1.22, 8.535, 3.965), #6: (1.22, 8.535, 2.745), #7: (1.22, 6.705, 2.745)

These coordinates are filled in a numpy array `corners_3d` based on the dimensions of court and rim pre-defined as variables. (For details, see the code.)

Question 5 [4pt]: Now we have established eight pairs of 2D and 3D keypoint correspondences. Consider this set of correspondence pairs in homogeneous coordinate $(\mathbf{x}_i, \mathbf{X}_i)$. Our goal is to compute a perspective projection matrix such that:

$$\mathbf{x}_i = \alpha \mathbf{P} \mathbf{X}_i = \alpha \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \mathbf{p}_3^T \end{bmatrix} \mathbf{X}_i = \alpha \begin{bmatrix} \mathbf{p}_1^T \mathbf{X}_i \\ \mathbf{p}_2^T \mathbf{X}_i \\ \mathbf{p}_3^T \mathbf{X}_i \end{bmatrix}$$

where $\mathbf{p}_i^T$ is ith row of $\mathbf{P}$; $\mathbf{x}_i = (x_i, y_i, 1)^T$; α is an arbitrary scalar.

- **5.a [1pt]:** Please prove that:

$$\mathbf{A}_i \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \mathbf{p}_3^T \end{bmatrix} = \mathbf{0} \text{ where } \mathbf{A}_i = \begin{bmatrix} \mathbf{0} & \mathbf{X}_i & -y_i \mathbf{X}_i \\ \mathbf{X}_i & \mathbf{0} & -x_i \mathbf{X}_i \end{bmatrix}$$

Hints: consider using cross product between two similar vectors.

Answer:

Proof:
($*$ Premise: $\mathbf{p}_i^T$ is a row vector. $\mathbf{x}_i$ and $\mathbf{X}_i$ are column vectors.)

$$\mathbf{x}_i \times \mathbf{P}\mathbf{X}_i = \mathbf{0} \quad (\because \mathbf{x}_i = \alpha \mathbf{P}\mathbf{X}_i)$$
$$\rightarrow \begin{bmatrix} y_i \mathbf{p}_3^T \mathbf{X}_i - \mathbf{p}_2^T \mathbf{X}_i \\ -x_i \mathbf{p}_3^T \mathbf{X}_i + \mathbf{p}_1^T \mathbf{X}_i \\ x_i \mathbf{p}_2^T \mathbf{X}_i - y_i \mathbf{p}_1^T \mathbf{X}_i \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

From the first and the second rows,

$$\mathbf{X}_i^T(\mathbf{p}_2^T)^T - y_i\mathbf{X}_i^T(\mathbf{p}_3^T)^T = 0,$$
$$\mathbf{X}_i^T(\mathbf{p}_1^T)^T - x_i\mathbf{X}_i^T(\mathbf{p}_3^T)^T = 0.$$

Hence,

$$\begin{bmatrix} \mathbf{0}_{1\times3} & \mathbf{X}_i^T & -y_i\mathbf{X}_i^T \\ \mathbf{X}_i^T & \mathbf{0}_{1\times3} & -x_i\mathbf{X}_i^T \end{bmatrix} \begin{bmatrix} (\mathbf{p}_1^T)^T \\ (\mathbf{p}_2^T)^T \\ (\mathbf{p}_3^T)^T \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \quad \blacksquare$$

- **5.b [3pt]:** complete the `findProjection` function and use this function to recover the perspective camera projection matrix, from the 3D world coordinate to the 2D image coordinate. Hints: 1) establish a linear system based on your derivation 2) given the eight keypoint correspondences, how many equations and how many unknown variables? 3) consider using `numpy.linalg.svd` to solve the linear system. 4) you might find `numpy.concatenate` and `numpy.reshape` to be handy.

Answer:

Below is the function `findProjection`, returning the 3×4 matrix `P` projecting from the 3D world coordinate `xyz` to the 2D image coordinate `uv`.

```
def findProjection(xyz, uv):
    N, _ = xyz.shape

    A = np.zeros((2*N, 12))
    for i in range(N):
        A[2*i, 4:8] = xyz[i]
        A[2*i+1, 0:4] = xyz[i]

        A[2*i, 8:12] = - uv[i][1] * xyz[i]
        A[2*i+1, 8:12] = - uv[i][0] * xyz[i]

    U, S, Vh = np.linalg.svd(A)
    p = Vh[-1]
    P = p.reshape(3, 4) / p[-1]

    return P
```

The algorithm follows the formulation in 5.a. Assuming that N point pairs are given, a $2N \times 12$ matrix `A` is constructed, where every odd and even rows are filled with the appropriate values for each point pair. The flattened projection matrix `p` is then obtained by solving a linear equation `Ap = 0`, where `p` satisfies the column vector corresponding to the minimal singular value in the right unitary matrix `V` of `A`'s singular value decomposition `USV'`. Then a vector `p` is reshaped into a 3×4 matrix `P` normalized by `P[2,3]`.

The estimated projection matrix is as follows:

```
P:
[[ 1.18682798e+01  5.03681893e+01  1.45686779e+01  7.34119776e+01]
 [-8.26855706e+00  8.76915144e+00 -2.39149474e+01  6.35921983e+02]
 [-2.16285953e-02  2.02166814e-02  4.33368607e-02  1.00000000e+00]]
```

The keypoints projected upon the estimated projection matrix `P` compared to their truth are in Fig. 4.

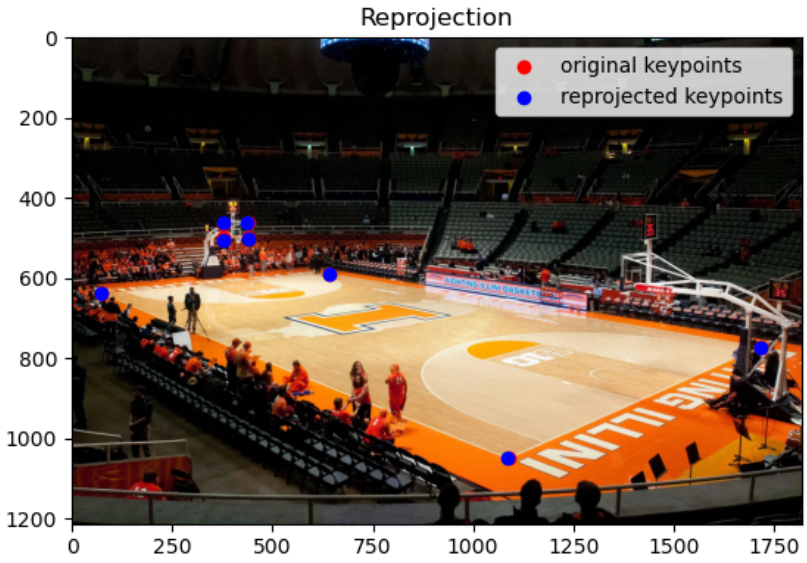

Figure 4: The comparison of reprojected and original keypoints.

- **5.c [1pt bonus]:** Try to answer the following questions: 1) assuming perfect correspondences, could we recover the projection matrix using even fewer points? Please indicate how many correspondences at least we need. 2) could we just use correspondences on the ground plane to recover this projection matrix? Please explain why or why not.

  Answer:

  - There are twelve unknown elements in the projection matrix $\mathbf{P}$ to be recovered, which means that we need at least twelve equations to obtain the solution of given matrix equation $\mathbf{Ap} = \mathbf{0}$. As each correspondence pair yields two linear equations in the matrix equation, at least six correspondences are required.
  - No. If only correspondences on the ground plane are given, the matrix $\mathbf{A}$ in $\mathbf{Ap} = \mathbf{0}$ becomes rank-deficient and the nontrivial solution $\mathbf{p}$ is not unique. Hence, additional correspondences outside the ground plane is necessary.

Question 6 [2pt]: Using our estimated projection matrix in Q5, we could insert any 3D object in the world coordinate frame onto the 2D image. In this question, let's consider the a simple case where we want to project a 3D point cloud onto the image plane and plot them. Suppose we want to put a giant stanford bunny statue in the center of the basketball court, could you compute the 2D coordinate of every vertex of this bunny using the projection matrix? Hints: 1) you might need to transform the bunny to the court center first 2) make sure the bunny is above the ground.

Answer:

Below is the code transforming the stanford bunny to the 3D coordinate on the court, project and represent it to the 2D image `uiuc.png`.

```
T_bunny_xyz = np.eye(4)
T_bunny_xyz[:3, -1] =\
    np.array([court_length/2, court_width/2, -verts[:, -1].min()])
verts =\
    T_bunny_xyz.dot(np.concatenate(
        [verts, np.ones((len(verts), 1))], axis = 1).T).T
bunny_uv = P.dot(verts.T).T
bunny_uv = bunny_uv / np.expand_dims(bunny_uv[:, 2], axis = 1)
```

First, the $SE(3)$ transformation matrix from the origin of the stanford bunny to that of 3D-coordinate on the court `T_bunny_xyz` is defined—this is intended to locate the bunny at the center of court on the ground. Note that only translational components `T_bunny_xyz[:3, -1]` are applied. Next, the bunny's pointclouds `verts` are expreseed in the 3D coordinate of the court by applying `T_bunny_xyz`. This is then projected onto the 2D image by applying the camera projection matrix `P`. As a result, we obtain the pointclouds projected on the image plane `bunny_uv` that we see.

The pointclouds of stanford bunny projected onto `uiuc.png` are in Fig. 5.

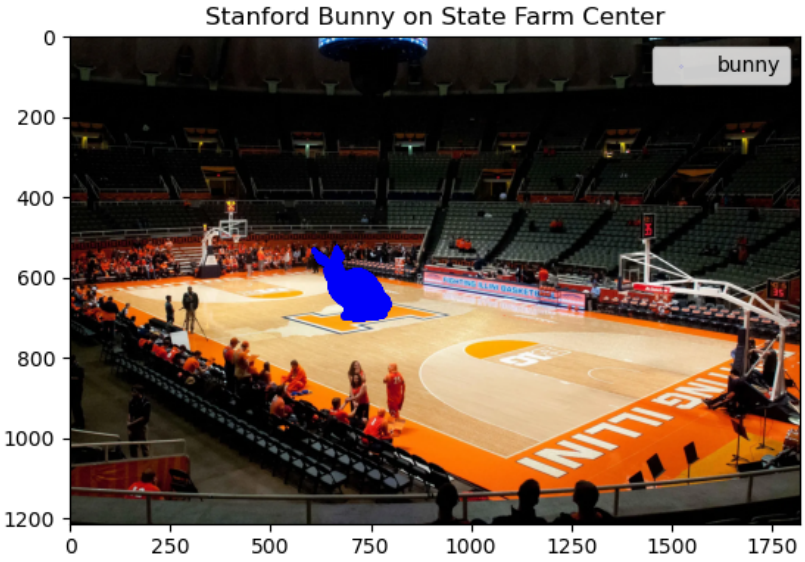

Figure 5: The projected stanford bunny in `uiuc.png`.

Question 7 [2pt bonus]: Try to use an off-the-shelf renderer to render the 3D mesh in a more realistic manner (e.g. pyrender, moderngl, blender, mitsuba2). You could consider the following perspectives that might influence the realism: 1) shading 2) shadows 3) occlusion reasoning 4) lighting environments.