Assignment 3

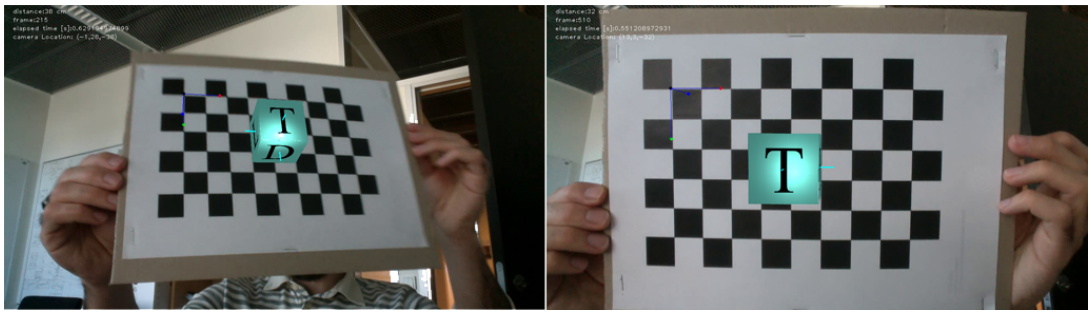

Figure 1: Augmented cube with shading. The light source is located on the camera center.

Purpose & Learning goals The purpose of this assignment is to be able to implement the illumination and shading that are the two basic concepts in computer graphics. By having a full description of the camera relative to the world coordinate system and also a virtual light source, you can do some augmentation in 3D, and map the points from one coordinate system to another in a hierarchy of the coordinates systems (world, camera, object, and light). In this part, you will calculate the three components of the Phong illumination model which are Ambient, Diffuse, and Specular shading and then you will use the flat/interpolated shading models for shading the augmented cube (Figure 1).
In this assignment you will continue working on the file Assignment_Cube.py of the Assignment2_part2, and doing some texture mapping and shading on the same augmented cube you had in the previous assignment. As you already know, inside the file Assignment_Cube.py there are several comment lines as below:
”’ <comment code> SOME TEXT...”’
The <comment code> is the address of the comment used in this document and indicates the order of the steps you need to follow for this assignment. Please follow this document step by step and write your piece of code for each step below the comment line in the Assignment_Cube.py. We suggest you to make a function for each step and just call the function below the comment line.

You should use the same groups as in assignment 1. Again for this assignment, you should possibly delegate the tasks among group members.

Submission The deadline for the report is **18th May**. The report should additionally be handed-in together with the other assignments by the end of the semester to the examination office. Make sure the report contains:

- Your written report with names and emails on the front page
- Result videos and possibly test data.
- A link to where the code can be downloaded
- The printed program.

Your report should NOT be a list of answers to the questions given in the exercise sheet. The questions should be considered as a guide that can help you to write a good report. The report should then either directly or indirectly answer the questions. Each report should be self contained and you have to describe what is the overall method (perhaps supported with figures showing individual steps) and what your assumptions are. Remember to show that your implementation works (testing). As with any other group work, it is important that you distribute the workload among the group members. So read through this document carefully and decide how you can work in the best possible way.
The **final report** (collection of all assignments) should be handed in to the examination office at the end of the semester. The final report should contain:

- Your written reports, perhaps arranged nicely :)

- A DVD with your report(s)
- Code
- Result videos and possibly test data.

Preparing the cube

So far you have successfully augmented a cube in the video sequence on top of the chessboard pattern. The cube is drawn as a wireframe. Here we want to add some textures to the sides of the cube. Inside the folder *assignment2_part2* there is a folder called "Images". You see 5 image files in this folder (Top.jpg, Down.jpg, ...). We want to load this images and map them to the corresponding faces of the cube.

1. The faces of the cube have been already defined in the assignment3.py (e.g., TopFace, and RightFace). Go to the comment line <010>. Call a function with the signature below:

   ```
   image=TextureFace(image,TopFace,currentCamera,"Images/Top.jpg")
   ```

   You need to make this function such that it takes the image of the current video frame and draw the texture on the corresponding face. You need to use homography and cv2.warpPerspective inside the function to do the texture mapping properly. You have done similar texture mapping in the assignment2_part1.

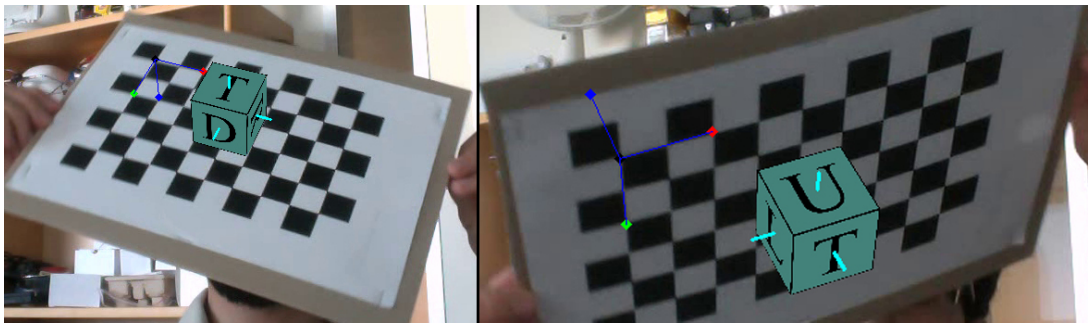

Figure 2: Texture mapping on the augmented cube

2. Repeat the texture mapping for all sides of the cube.
3. The result of your texture mapping would be transparent when you use cv2.addWeighted(). Use some other methods to make the textures not transparent. (Hint: create a mask from the face and use cv2.bitwise_and and cv2.bitwise_or to combine the texture and the image. You have used the same technique before for the exercise image arithmetic of the Exercises3)

   ```
   I1=cv2.bitwise_and( Mask,image)
   image=cv2.bitwise_or(I1, texture)
   ```

The result of the texture mapping will be something similar to Figure 2 except that you see the faces that should be hidden from the camera perspective.

Back-Face Culling

As you have noticed your augmented cube does not look very nice after the texture mapping. All you need here is to remove the faces that are hidden in the current perspective.

1. Go to the comment line<012>, calculate the normal vectors of the cube faces and draw these normal vectors on the center of each face in the image.
2. Go to the comment line <013>, calculate the angle between the normal vector and the line passing through the camera origin and the face center, and use that for hidden face removal.

Shading on the textured cube

The algorithm that you are going to follow for the shading can be divided into three steps:

step1: First, calculating the shading values (R,G, and B channel) for each point of the real cube in the world coordinate system by having the information about the camera, light source, and the cube position.

step2: Then, finding the corresponding pixel in the image for each point.

step3: And finally, Multiplying the shading value to the pixel intensity (for each channel).

$$I = T(s,t)[I_a k_a(x) + I_l(x) k_d(x) \max(n(x)\cdot l(x),0) + I_s k_s (r\cdot v)^{\alpha}]$$

There are some functions in the *SIGBTools.py* that help you in implementing the shading part (e.g., Bilinear-Interpo(), GetFaceNormal(), and CalculateFaceCornerNormals). In the following, we describe how to use these functions, and give you some more code to begin with and then we will leave you alone to finish the shading by yourself :)

1. In the main body of the program before the comment line <000> add the line bellow:

```
TopFaceCornerNormals,RightFaceCornerNormals,LeftFaceCornerNormals,UpFaceCornerNormals,
    DownFaceCornerNormals=CalculateFaceCornerNormals(TopFace,RightFace,LeftFace,UpFace,
    DownFace)
```

   The function CalculateFaceCornerNormals() that exists in the SIGBTools.py calculates the normal vectors at the corners of the cube and return four vectors for each face. You will use that for the interpolation shading (Phong intepolation shading) **later**.

2. Go to the comment line <011>, show the distance between the camera origin and the world origin in the image. Run the program and see whether the distance measured makes sense. What is the unit of the measurement here and how to adjust that? (hint: camera translation can be obtained form the camera matrix!)

3. Go to the comment line <010>, and right after texture mapping add the line below:

```
image=ShadeFace(image,TopFace,TopFaceCornerNormals,currentCamera)
```

   This line calls the function *ShadeFace()* and shades the faces of the cube in the image. The main structure of this function is presented below.

4. Ok, so far, so good! Here we will give you the code and algorithm needed for the steps 2 and 3 that are mentioned above, and you only need to calculate the shading values for each face. This means that you need to implement the function *CalculateShadeMatrix()* that you see inside the *ShadeFace()* below. **However, in your report, we expect you to describe exactly what is going on inside the *ShadeFace()* .**

```
def ShadeFace(image,points,faceCorner_Normals, camera):
    global shadeRes
    shadeRes=10

    videoHeight, videoWidth, vd = array(image).shape

    #.............................
    points_Proj=camera.project(toHomogenious(points))
    points_Proj1 = np.array([[int(points_Proj[0,0]),int(points_Proj[1,0])],[int(
        points_Proj[0,1]),int(points_Proj[1,1])],[int(points_Proj[0,2]),int(points_Proj
        [1,2])],[int(points_Proj[0,3]),int(points_Proj[1,3])]])

    #.............................
```

```
square = np.array([[0, 0], [shadeRes-1, 0], [shadeRes-1, shadeRes-1], [0, shadeRes-
    1]])

#.............................
H = estimateHomography(square, points_Proj1)


#.............................
Mr0,Mg0,Mb0=CalculateShadeMatrix(image,shadeRes,points,faceCorner_Normals, camera)
#  HINT
# type(Mr0): <type 'numpy.ndarray'>
# Mr0.shape: (shadeRes, shadeRes)



#.............................
Mr = cv2.warpPerspective(Mr0, H, (videoWidth, videoHeight),flags=cv2.INTER_LINEAR)
Mg = cv2.warpPerspective(Mg0, H, (videoWidth, videoHeight),flags=cv2.INTER_LINEAR)
Mb = cv2.warpPerspective(Mb0, H, (videoWidth, videoHeight),flags=cv2.INTER_LINEAR)


#.............................
image=cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
[r,g,b]=cv2.split(image)

#.............................
whiteMask = np.copy(r)
whiteMask[:,:]=[0]
points_Proj2=[]
points_Proj2.append([int(points_Proj[0,0]),int(points_Proj[1,0])])
points_Proj2.append([int(points_Proj[0,1]),int(points_Proj[1,1])])
points_Proj2.append([int(points_Proj[0,2]),int(points_Proj[1,2])])
points_Proj2.append([int(points_Proj[0,3]),int(points_Proj[1,3])])


cv2.fillConvexPoly(whiteMask,array(points_Proj2),(255,255,255))


#.............................
r[nonzero(whiteMask>0)]=map(lambda x: max(min(x,255),0),r[nonzero(whiteMask>0)]*Mr[
    nonzero(whiteMask>0)])
g[nonzero(whiteMask>0)]=map(lambda x: max(min(x,255),0),g[nonzero(whiteMask>0)]*Mg[
    nonzero(whiteMask>0)])
b[nonzero(whiteMask>0)]=map(lambda x: max(min(x,255),0),b[nonzero(whiteMask>0)]*Mb[
    nonzero(whiteMask>0)])


#.............................
image=cv2.merge((r,g,b))
image=cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

return image
```

In your report describe the algorithm used above, and describe the alternative methods that can be used instead, and their differences.

5. implement the *CalculateShadeMatrix()* function used in the ShadeFace() function that calculates the shading values of the faces. **First start with the flat shading and then make another version that can do the phong interpolation shading.** Use the values below for the phong illumination model. Define the point light source at the camera center (assume that the light is located at the center of the camera). The functions *BilinearInterpo()* and *GetFaceNormal()* in the SIGBTools can be used respectively for Bilinear interpolation of a rectangular faces and calculating a face normal vector.

```
#Ambient light IA=[IaR,IaG,IaB]
IA = np.matrix([5.0, 5.0, 5.0]).T
#Point light IA=[IpR,IpG,IpB]
IP = np.matrix([5.0, 5.0, 5.0]).T
#Light Source Attenuation
```

```
fatt = 1
#Material properties: e.g., Ka=[kaR; kaG; kaB]
ka=np.matrix([0.2, 0.2, 0.2]).T
kd= np.matrix([0.3, 0.3, 0.3]).T
ks=np.matrix([0.7, 0.7, 0.7]).T

alpha = 100
```

1 * Extra Credit

1. Modify the code such that the location of the light source can be moved by the mouse coordinates in the image (Light=[mouseX,mouseY,Z]).