1 前提提示

1.1 def get_face_color(normal, point_light_direction=(0, 0, 1)):(0,0,1)

表示函数如果没有参数**传进**来的默**认**值

1.2 取 mean 等，如果 axis=x 就是去除那一个**维**数

将包含元**组**的数**组变为**数组

```
corners = np.array([(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)])
print(corners)
```

```
[[0 0 0]
 [0 0 1]
 [0 1 0]
 [0 1 1]
 [1 0 0]
 [1 0 1]
 [1 1 0]
 [1 1 1]]
```

促成正方体形成

corner 是形如(6,4,3) normal 是各个面的中**间**点

1.3 get_cub 函数

matmul 是矩**阵**乘法

```
def get_cube(center=(0, 0, 2), rotation_angles=[0., 0., 0.],
with_normals=False, scale=1.):
     corners = np.array([(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0),
(1, 0, 1), (1, 1, 0), (1, 1, 1)])
    corners = corners - np.array([0.5, 0.5, 0.5], dtype=np.float32).reshape(1,
3)
    # Let's scale the cube
    corners = corners * scale
    # And we rotate the cube wrt. the input rotation angles
    rot_mat = R.from_euler('xyz', rotation_angles, degrees=True).as_matrix()
    corners = np.matmul(corners, rot_mat.T)
    # Finally, we shift the cube according to the input center tuple
    corners = corners + np.array(center, dtype=np.float32).reshape(1, 3)

    # The 6 faces of the cube are then given as:
    faces = np.array([
    # all faces containing (0, 0, 0)
    [corners[0], corners[1], corners[3], corners[2]],
    [corners[0], corners[1], corners[5], corners[4]],
    [corners[0], corners[2], corners[6], corners[4]],
    # all faces containing (1, 1, 1)
    [corners[-1], corners[-2], corners[-4], corners[-3]],
    [corners[-1], corners[-2], corners[-6], corners[-5]],
    [corners[-1], corners[-3], corners[-7], corners[-5]],
    ])
```

```
    if with_normals:
        normals = np.array([(-1, 0, 0), (0, -1, 0), (0, 0, -1), (1, 0, 0), (0,
1, 0), (0, 0, 1)])
        normals = np.matmul(normals, rot_mat.T)
        return faces, normals
    else:
        return faces
```

2 简单立体图形旋转表示

2.1 get_camera_intrinsics

def get_camera_intrinsics(fx=70, fy=70, cx=W/2., cy=H/2.)

return K = np.array([[fx, 0, cx], [0, fy, cy], [0, 0, 1],], dtype=np.float32)

```
def get_camera_intrinsics(fx=70, fy=70, cx=W/2., cy=H/2.):
  K = np.array([
              [fx, 0, cx],
              [0, fy, cy],
              [0, 0,  1],
  ], dtype=np.float32)

  assert(K.shape == (3, 3) and K.dtype == np.float32)
  return K
```

2.2 get_perspective_projection

input x_c, K 输入 x_c 为[x,y,z]即三维点

输出 return a 2D vector for x_s [x,y]

经过了三维根据 K 到二维的视角转换

It takes in a 3D point in camera space x_c and the camera matrix K

```
  def get_perspective_projection(x_c, K):

  assert(x_c.shape == (3,) and K.shape == (3, 3))

  xc_projected = np.matmul(K, x_c)

  x_s = xc_projected[:2] / xc_projected[-1]
  assert(x_s.shape == (2,))
  return x_s
```

2.3 project_cube

x_s 先变成[[[x,y,z],[x,y,z]][,,]] 经过改造变成[[x,y,z],[x,y,z],[x,y,z],,,]统一计算透视投影，在变为[[x,y],[x,y],[x,y],,]

请注意！这里的输出与输入的 cube 的前两维的长度相等，因为这里使用了 s 来储存了其原本的形状。这一点使得后面的调用的参数输入匹配正确

```
def project_cube(cube, K):
    s = cube.shape
    assert(s[-1] == 3)
    cube = cube.reshape(-1, 3)
    projected_cube = np.stack([get_perspective_projection(p, K) for p in cube])
    projected_cube = projected_cube.reshape(*s[:-1], 2)
    return projected_cube
```

如下，reshape 第一个参数可以用-1 替代，自动补齐

```
import numpy as np
pro=np.array([[2,3],[1,4],[5,4]])
pro=pro[:-1]
pro=pro.reshape(-1,2)
print(pro)
```

```
[[2 3]
 [1 4]]
```

2.4 plot_projected_cube

matplotlib 简明教程

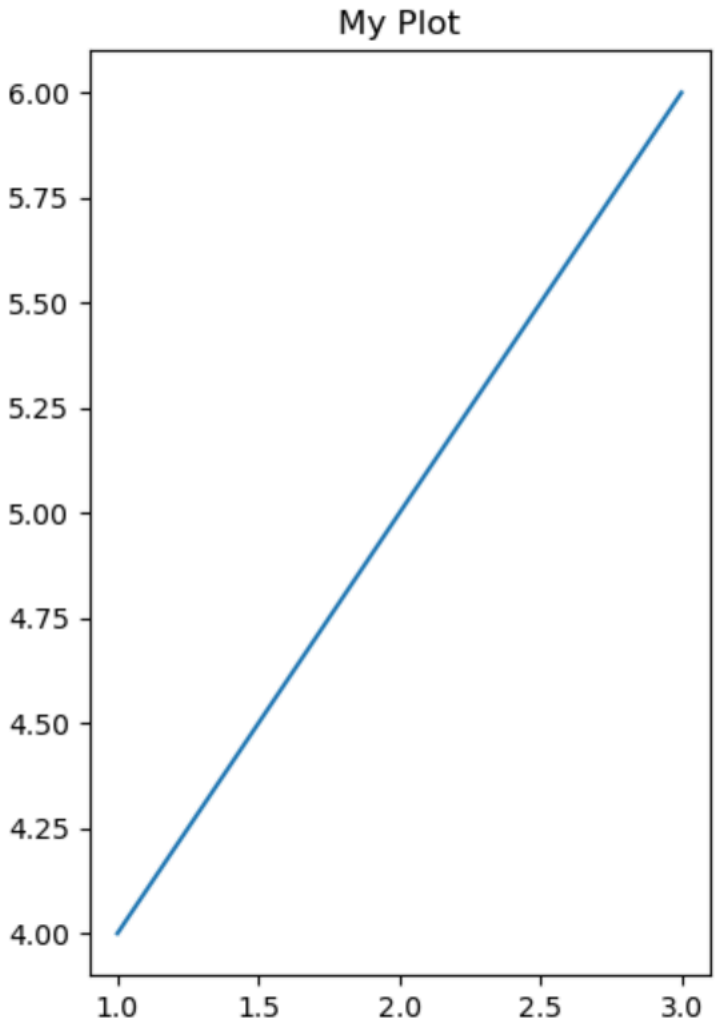
[137]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(4, 6))

ax.plot([1, 2, 3], [4, 5, 6])

ax.set_title('My Plot')

plt.show()
```



assert(projected_cube.shape == (6, 4, 2)) 很明显的六个面每个面四个点对应的 x，y

```
def plot_projected_cube(projected_cube, figsize=(5, 5), figtitle=None,
colors=None, face_mask=None):
    assert(projected_cube.shape == (6, 4, 2))
    fig, ax = plt.subplots(figsize=figsize)
    if figtitle is not None:
        fig.suptitle(figtitle)
    if colors is None:
        colors = ['C0' for i in range(len(projected_cube))]
    if face_mask is None:
        face_mask = [True for i in range(len(projected_cube))]
    ax.set_xlim(0, W), ax.set_ylim(0, H)
    ax.set_xlabel('Width'), ax.set_ylabel("Height")
    for (cube_face, c, mask) in zip(projected_cube, colors, face_mask):
        if mask:
            ax.add_patch(Polygon(cube_face, color=c))
    plt.show()
```

None 表示“未指定”,默认情况下为 True，使得 mask 在此处都要执行

suptitle 就是个加在最上面的标题的设置

zip 很明显是将一群元组包裹的，一般用在 range 后面的参数中，由于需要打包给元组赋值

polygon 是可视化的一个函数

2.5 主函数

```
K = get_camera_intrinsics()
cube = get_cube(rotation_angles=[110, 70, 30])
projected_cube = project_cube(cube, K)
plot_projected_cube(projected_cube, figtitle='Projected Cube')
```

3 增加光线照射效果的立体图形显示

3.1 get_face_color

前提提示，元组的元素变为向量的元素

```
    a=(2,2,3)
    b=np.array(a)
    print(b)
    #[2 2 3]
```

3.2 遮挡判定

光线照射方向与面朝向的点乘小于零时才能显现

```
def get_face_mask(cube, normals, camera_location=(0, 0, 0)):
    assert(cube.shape == (6, 4, 3) and normals.shape[-1] == 3)
    camera_location = np.array(camera_location).reshape(1, 3)

    face_center = np.mean(cube, axis=1)

    viewing_direction = camera_location - face_center
    dot_product = np.sum(normals * viewing_direction, axis=-1)
    mask = dot_product > 0.0
    return mask
```

其他函数请看文档，值得一提的是 sum 的写法,按最后一行合并，少一个维数

```
dot_product = np.sum(normals * viewing_direction, axis=-1)
```

以及最终呈现的函数的形式

```
plot_projected_cube(projected_cube, figtitle="Projected Cuboid with Shading",
colors=colors, face_mask=mask)
```

4 动态实现

4.1 get animation 函数

```
def get_animation(K_list, cube_list, figsize=(5, 5), title=None):
    assert(len(K_list) == len(cube_list))
    cubes = [i[0] for i in cube_list]
    normals = [i[1] for i in cube_list]

    colors = [get_face_colors(normals_i) for normals_i in normals]
    masks = [get_face_mask(cube_i, normals_i) for (cube_i, normals_i) in
zip(cubes, normals)]

    projected_cubes = [project_cube(cube, Ki) for (cube, Ki) in zip(cubes,
```

```
K_list)]

    uv = projected_cubes[0]
    patches = [Polygon(uv_i, closed=True, color='white') for uv_i in uv]

    def animate(n):
        uv = projected_cubes[n]
        color = colors[n]
        mask = masks[n]
        for patch, uv_i, color_i, mask_i in zip(patches, uv, color, mask):
            if mask_i:
                patch.set_xy(uv_i)
                patch.set_color(color_i)
            else:
                uv_i[:] = -80
                patch.set_color(color_i)
                patch.set_xy(uv_i)
        return patches

    fig, ax = plt.subplots(figsize=figsize)
    if title is not None:
        fig.suptitle(title)
    plt.close()
    ax.set_xlim(0, W)
    ax.set_ylim(0, H)
    for patch in patches:
        ax.add_patch(patch)
    anim = animation.FuncAnimation(fig, animate, frames=len(K_list),
interval=100, blit=True)
    return anim
```

值得注意的是，对于看不到的面，我们将它的 x，y 坐标设成-80，这样，面上的所有点将会聚集在一个点上，不显示该面。

4.2 main(rotate)

```
K_list = [get_camera_intrinsics() for i in range(30)]
cube_list = [get_cube(rotation_angles=[2*angle, angle, angle],
with_normals=True) for angle in np.linspace(0, 360, 30)]
anim = get_animation(K_list, cube_list, title="Rotation of Cube")
HTML(anim.to_html5_video())
```

4.3 main2(stretch)

```
K_list = [get_camera_intrinsics(fx=f) for f in np.linspace(10, 150, 30)]
cube_list = [get_cube(rotation_angles=(0, 30, 50), with_normals=True) for i in
range(30)]
anim = get_animation(K_list, cube_list, title="Change of focal length along the
x-axis.")
HTML(anim.to_html5_video())
```

4.4 main3(further)

```
K_list = [get_camera_intrinsics(fx=f, fy=f) for f in np.linspace(10, 150, 30)]
cube_list = [get_cube(rotation_angles=(0, 30, 50), with_normals=True) for i in
range(30)]
anim = get_animation(K_list, cube_list, title="Change of focal length along
both axes.")
HTML(anim.to_html5_video())
```

4.5 main(moving)

```
K_list = [get_camera_intrinsics() for i in range(30)]
cube_list = [get_cube(center=(i, 0, 2), rotation_angles=(0, 30, 50),
with_normals=True) for i in np.linspace(-2, 2, 30)]
anim = get_animation(K_list, cube_list, title="Change of cube translation along
y-axis.")
HTML(anim.to_html5_video())
```

4.6 同时改**变**焦距和物体坐**标**

```
K_list = [get_camera_intrinsics(fx=f, fy=f) for f in np.linspace(30, 500, 30)]
cube_list = [get_cube(center=(0, 0, i), rotation_angles=(30, 50, 0),
with_normals=True) for i in np.linspace(1., 10, 30)]
anim = get_animation(K_list, cube_list, title="Dolly Zoom Effect.")
HTML(anim.to_html5_video())
```