**CSE 559A/Fall 2020. Problem Set 4 Solution Key.**

1 (a) Let the world co-ordinates be $(X, Y, Z)$ in the chosen co-ordinate system (aligned with the left camera). Then, $x = fX/Z, y = fY/Z, x' = f(X - t_x)/Z, y' = fY/Z$ from just applying the projections to the homogeneous 3-D co-ordinates $(X, Y, Z, 1)$, and converting the homogeneous 2-D back to Cartesian 2-D. Therefore, $y = y'$ and $x - x' = ft_x/Z$, which is $\geq 0$ since $t_x$ and $Z$ are both $\geq 0$.

(b) $\alpha X + \beta Y + Z\gamma = k \rightarrow \frac{1}{Z} = \frac{\alpha}{k}\frac{X}{Z} + \frac{\beta}{k}\frac{Y}{Z} + \frac{\gamma}{k}$. Replacing $x = fX/Z, y = fY/Z$ and using $d = ft_x/Z$: $d = ft_x\frac{1}{Z} = \frac{\alpha t_x}{k}x + \frac{\beta t_x}{k}y + \frac{ft_x\gamma}{k}$. So $a = t_x\alpha/k, b = t_x\beta/k, c = t_x f\gamma/k$.

(c) $x = fX/Z, y = fY/Z$ and $x' = fX/(Z - t_z), y' = fY/(Z - t_z)$. So, $x' = Zx/(Z - t_z) = x/(1 - t_z/Z)$ and $y' = y/(1 - t_z/Z)$. This is for known $Z$. For all possible $Z \geq t_z$, we see that the locus of possible values of $(x', y')$ is $x' = \alpha\ x, y' = \alpha\ y$, with $\alpha \in (1, \infty)$: $\alpha = 1$ for $Z = \infty$, and $\alpha = \infty$ for $Z = t_z$. This is basically a ray in the direction joining the center of the image to the original co-ordinates $(x, y)$, starting $(x, y)$ and radiating outwards. So, as we move forward, points will appear to move concentrically outwards from the image center, with points that are really far away appearing to stay still.

2 (a)

```
def buildcv(left,right,dmax):
    cv = 24 * np.ones([left.shape[0],left.shape[1],dmax+1], dtype=np.float32)
    lc = census(left); rc = census(right)
    for i in range(1,dmax+1):
        cv[:,i:,i] = hamdist(lc[:,i:],rc[:,0:(left.shape[1]-i)])
    return cv
```

(b)

```
def bfilt(cv,X,K,sgm_s,sgm_i):
    H = X.shape[0]; W = X.shape[1]; D = cv.shape[2]
    cv1 = np.zeros(cv.shape); B = np.zeros([H,W,1])
    for y in range(-K,K+1):
        for x in range(-K,K+1):
            if y < 0:
                y1a = 0; y1b = -y; y2a = H+y; y2b = H
            else:
                y1a = y; y1b = 0; y2a = H; y2b = H-y
            if x < 0:
                x1a = 0; x1b = -x; x2a = W+x; x2b = W
            else:
                x1a = x; x1b = 0; x2a = W; x2b = W-x

            bxy = X[y1a:y2a,x1a:x2a,:] - X[y1b:y2b,x1b:x2b,:]
            bxy = np.sum(bxy*bxy,axis=2,keepdims=True)
            bxy = bxy/(sgm_i**2) + np.float32( y**2 + x**2)/(sgm_s**2)
            bxy = np.exp(-bxy/2.0)

            B[y1b:y2b,x1b:x2b,:] = B[y1b:y2b,x1b:x2b,:]+bxy
            cv1[y1b:y2b,x1b:x2b,:] = cv1[y1b:y2b,x1b:x2b,:]+bxy*cv[y1a:y2a,x1a:x2a,:]
    return cv1/B
```



3 (a)

```
def viterbilr(cv,P1,P2):
    H = cv.shape[0]; W = cv.shape[1]; D = cv.shape[2]

    dout = np.zeros((H,W),np.int32)
    back = np.zeros([H,W-1,D])
    prev = cv[:,0:1,:]
```

```
    Dvals = np.tile(np.arange(D).reshape([1,1,D]),[H,1,1])
    for i in range(W-1):
        prev = prev - np.min(prev,axis=2,keepdims=True)

        back[:,i:i+1,:] = np.argmin(prev,axis=2)[:,:,np.newaxis]
        nxt = P2*np.ones((H,1,D))

        cond = prev[:,:,:-1]+P1  < nxt[:,:,1:]
        back[:,i:i+1,1:][cond] = Dvals[:,:,:-1][cond]
        nxt[:,:,1:] = np.minimum(nxt[:,:,1:],prev[:,:,:-1]+P1)

        cond = prev[:,:,1:]+P1  < nxt[:,:,:-1]
        back[:,i:i+1,:-1][cond] = Dvals[:,:,1:][cond]
        nxt[:,:,:-1] = np.minimum(nxt[:,:,:-1],prev[:,:,1:]+P1)

        cond = prev < nxt
        back[:,i:i+1,:][cond] = Dvals[cond]
        nxt = np.minimum(nxt,prev)

        prev = nxt + cv[:,i+1:i+2,:]

    dout[:,W-1:W] = np.argmin(prev,axis=2)
    for i in range(W-1,0,-1):
        dout[:,i-1:i] = back[np.arange(H),i-1:i,dout[:,i]]

    return dout
```

(b)

```
def SGMstep(cv,P1,P2,ax,n0,dn,nsteps):
    cv1 = np.zeros(cv.shape); n = n0
    if ax == 0:
        prev = cv[n,:,:]
    else:
        prev = cv[:,n,:]
    prev = prev - np.min(prev,axis=1,keepdims=True)
    if ax==0:
        cv1[n,:,:] = prev
    else:
        cv1[:,n,:] = prev
    n = n + dn
    for i in range(nsteps-1):
        nxt = np.minimum(P2,prev)
        nxt[:,1:] = np.minimum(nxt[:,1:],prev[:,:-1]+P1)
        nxt[:,:-1] = np.minimum(nxt[:,:-1],prev[:,1:]+P1)
        if ax == 0:
            prev = nxt + cv[n,:,:]
        else:
            prev = nxt + cv[:,n,:]
        prev = prev - np.min(prev,axis=1,keepdims=True)
        if ax==0:
            cv1[n,:,:] = prev
        else:
            cv1[:,n,:] = prev
        n = n + dn
    return cv1

def SGM(cv,P1,P2):
```

```
    H = cv.shape[0]; W = cv.shape[1]; D = cv.shape[2]
    cv1 = SGMstep(cv,P1,P2,1,0,1,W)
    cv1 = cv1+SGMstep(cv,P1,P2,0,0,1,H)
    cv1 = cv1+SGMstep(cv,P1,P2,1,W-1,-1,W)
    cv1 = cv1+SGMstep(cv,P1,P2,0,H-1,-1,H)
    return np.argmin(cv1,axis=2)
```

4.

```
def lucaskanade(f1,f2,W):
    iavg = (f1+f2)/2
    ix = conv2(iavg,fx,'same'); iy = conv2(iavg,fy,'same')
    it = f2-f1

    ixx = conv2(conv2(ix*ix,np.ones((W,1)),'same'),np.ones((1,W)),'same') + 1e-6
    iyy = conv2(conv2(iy*iy,np.ones((W,1)),'same'),np.ones((1,W)),'same') + 1e-6
    ixy = conv2(conv2(ix*iy,np.ones((W,1)),'same'),np.ones((1,W)),'same')
    ixt = conv2(conv2(ix*it,np.ones((W,1)),'same'),np.ones((1,W)),'same')
    iyt = conv2(conv2(iy*it,np.ones((W,1)),'same'),np.ones((1,W)),'same')
    det = (ixx*iyy - ixy**2)
    u = -(iyy*ixt - ixy*iyt)/det; v = -(-ixy*ixt + ixx*iyt)/det

    return u,v
```