Create the following synthetic scene ...

Let the background plane be described as $Z = Z_0 + pX + qY$, and the foreground cube be of size $2 \times 2 \times 2$, and positioned by a translation $T_c = (U_c, V_c, W_c)$ and a rotation $R_c = (\alpha_c, \beta_c, \gamma_c)$. The constrain is that all of the cube's 8 vertices, $(X_i, Y_i, Z_i)$, $i = 1, 2, ..., 8$, must satisfy $Z_i \leq Z_0 + pX_i + qY_i$. Another restriction is that none of the 8 vertices can be behind the image plane, which is defined as $Z = 1$. The rigid motion of observer is given as a translation $t = (U, V, W)$ and a rotation $\omega = (\alpha, \beta, \gamma)$.

The difficulty of this problem, given an image point $(x, y, 1)$, is to determine which plane the object point is on. There are 7 planes in this problem, among which 1 is the background, and 6 are of the cube. Intuitively by intersecting the line of sight (from origin through the image point) with 7 planes and pick the joint point nearest to origin, I can find the corresponding object point, as well as the plane it is on.

But there is one subtle point here. The 6 facets of the cube are not infinite planes, thus the line of sight may have no intersection with some/any of them. Actually since the cube is a convex polyhedron, the line of sight must intersect with at least (furthermore we can show at most) two of its facets or none at all. I need to determine if there is an intersection, and if so, which one is visible. Next is my solution.

First I find all 6 intersection points of the line of sight with 6 facet planes, then apply the inverse translation $T_c^- = -T_c$ and inverse rotation $R_c^- = -R_c$ on them. Now they are as if in a coordinate system with origin at the center of the cube and three axis parallel to cube' s edges, and can be easily determined to be on the cube's facets or not. Due to rounded error the intersection point may not be exactly on one facet but within a small range.

Now the whole process of computing motion field (on a set of grid points only) can be descibed as the next algorithm.

(a) Position the cube's 8 vertices, $c_i(x, y, z)$, $i = 1, ..., 8$, using $T_0$ and $R_0$.

(b) Compute the cube's 6 facets as $a_k X + b_k Y + c_k Z = 1$, $k = 1, ..., 6$.

(c) For every grid point on image plane, $(x, y, 1)$:

   i. For each facet, numbered as $k = 1, ..., 6$:

      A. Find the intersection point $(X_k, Y_k, Z_k)$ of this facet with the line of sight, which passes the origin and the image point $(x, y, 1)$.

      B. In case this facet plane passes the origin, the intersection point is set to $(0, 0, 0)$.

      C. Apply inverse translation and rotation, $-T_c$ and $-R_c$, on $(X_k, Y_k, Z_k)$ to get $(X_k', Y_k', Z_k')$.

      D. Determine, in the cube's coordinate system, if $(X_k', Y_k', Z_k')$ is on (or very near) this facet.

      E. If so, keep it. Otherwise set $(X_k, Y_k, Z_k) = (0, 0, 0)$.

   ii. Compute the intersection point $(X_7, Y_7, Z_7)$ of the line of sight and the background plane.

   iii. Among all possible intersection points $(X_k, Y_k, Z_k)$, $k = 1, ..., 7$, find the one with minimal and nonzero $Z_k$. This is the object point. Let it be $(X, Y, Z)$.

iv. The object point velocity is

$$\begin{cases} X' &= -U - \beta Z + \gamma Y \\ Y' &= -V - \gamma X + \alpha Z \\ Z' &= -W - \alpha Y + \beta X \end{cases}$$

or

$$\vec{R}' = -\vec{t} - \vec{\omega} \times \vec{R}$$

v. The image motion flow is

$$\begin{cases} u &= (\frac{X}{Z})' = \frac{X'}{Z} - \frac{XZ'}{Z^2} \\ v &= (\frac{Y}{Z})' = \frac{Y'}{Z} - \frac{YZ'}{Z^2} \end{cases}$$

Once motion field is known, by projecting the motion flow onto the gradient direction it's easy to get normal motion field.

Figure1 and figure2 show the motion fields and corresponding normal motion fields (where the gradient direction is generated using rand function).

4. Compute the normal flow field for the Yosemite Sequence ...

The part about Lucas and Kanade's method in the paper by Barron et al. is attached at the end of this report. Please refer there for the details of their algorithm.

I closely followed the requirement in problem statement. The only freedom allowed us is the selection of threshold in determining if the spatial derivatives are large enough. I used 1. If the spatial gradient magnitute is large than 1, then I go on to compute the motion flow at that point. Otherwise I simply set the motion flow to be zero there.

Figure shows the averaged image (frame #8 out of 15), the spatial and temporal derivatives, and the motion field and normal field.

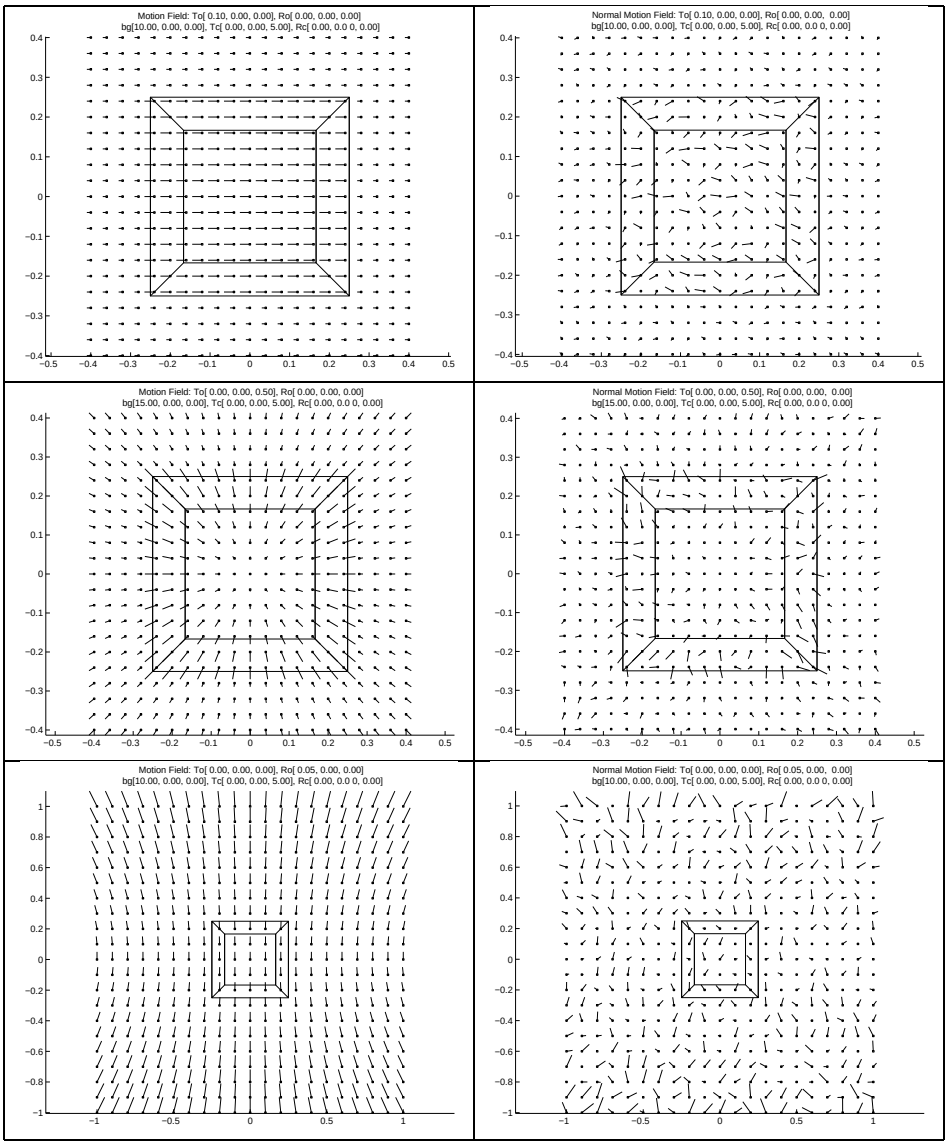


Figure 1: Computed motion fields and normal motion fields

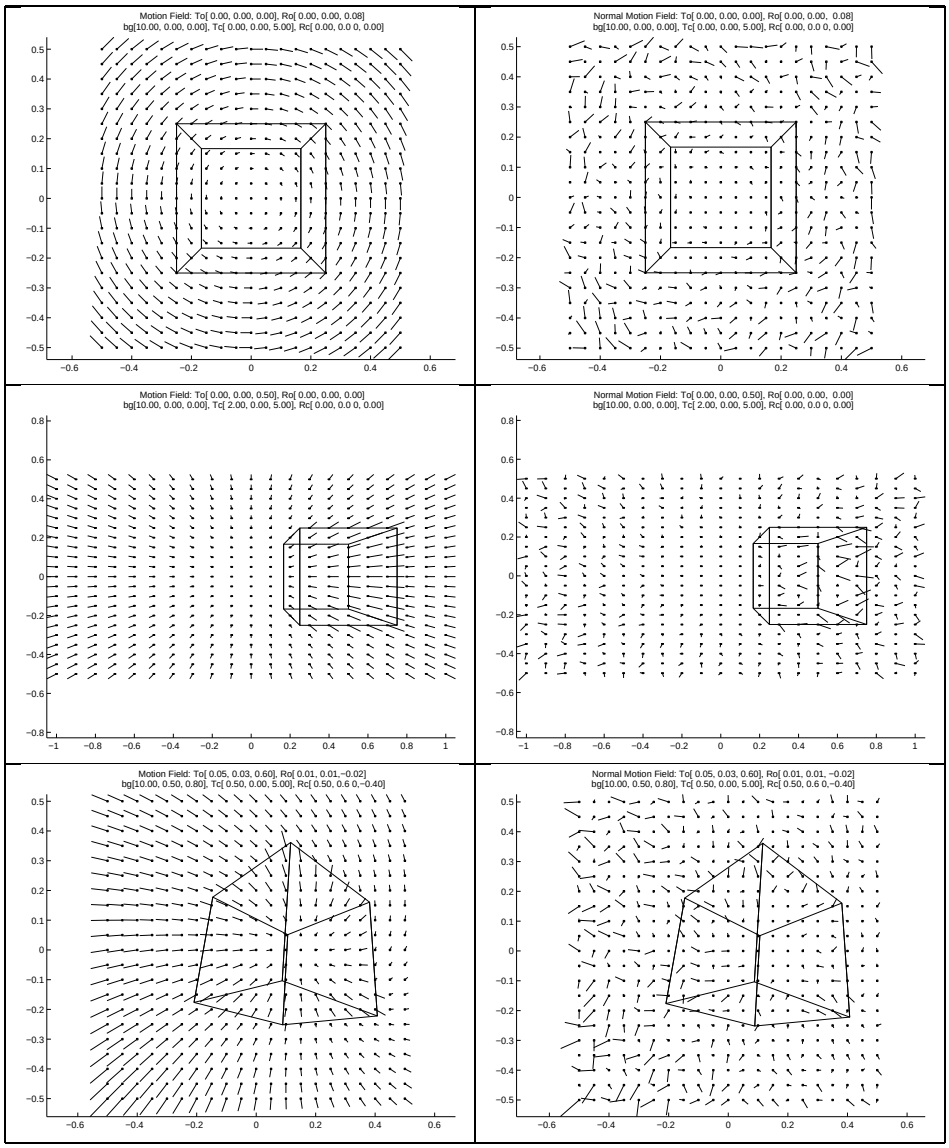


Figure 2: Computed motion fields and normal motion fields

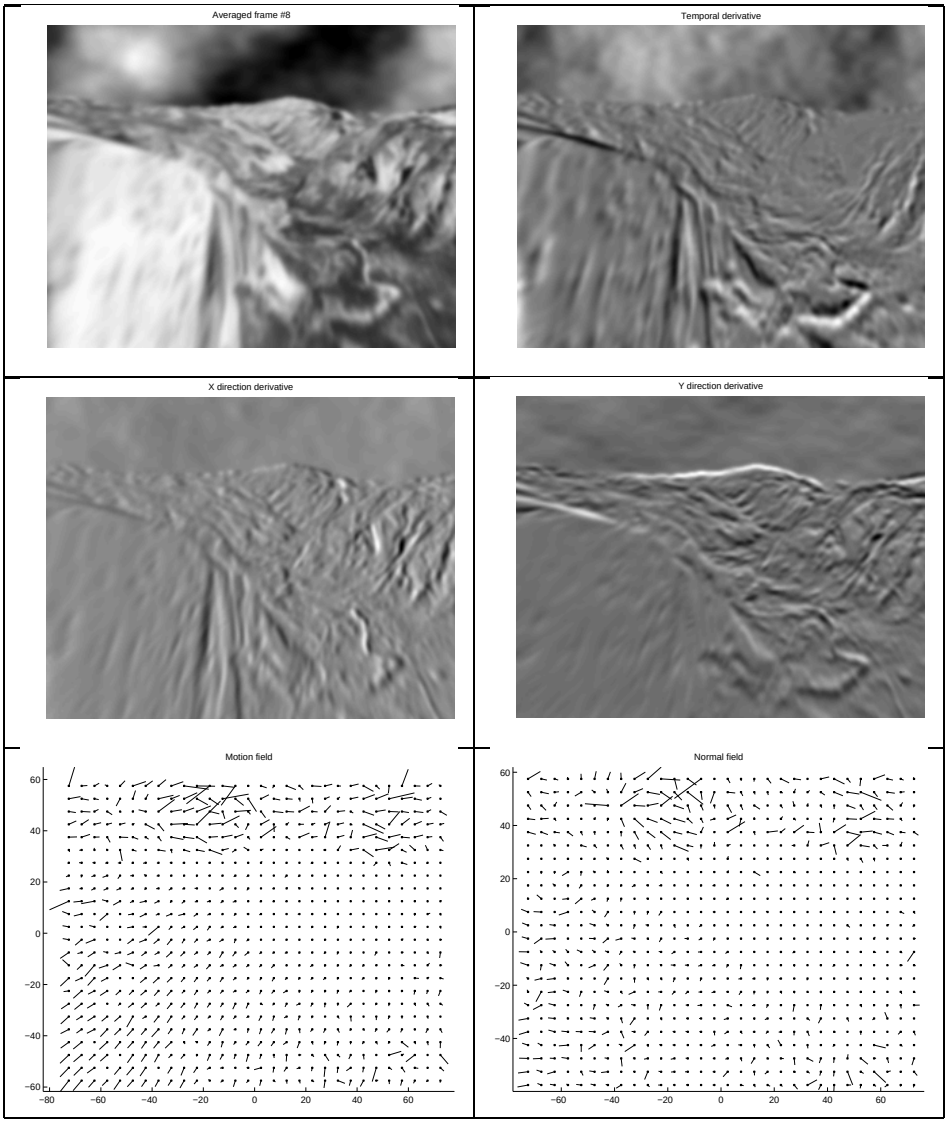


Figure 3: Results of problem 4

Appendix: Matlab code for problem 3 & 4

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                 COPYRIGHT (C) LJ 2000
% DESCRIPTION
%  Main routine for hw #11 problem 3.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Clear up.
clear;
close all;

% Ask for parameters.
%askforinput;
setparameters;

% Position 8 vertices of the cube.
Rc = rotationfromangle(rc(1),rc(2),rc(3));
c = position8vertices(tc, Rc);

% Check validity of settings.
if (0 == isvalidsetting(c, bg))
    disp('>>Error: invalid settings');
    return;
end

% Construct 6 facets.
facets = computefacets(c);

% For every grid point on image plane compute motion flow.
hnd = waitbar(0, 'C o m p u t i n g  M o t i o n  F i e l d . . .');
MF_u = zeros(H,W); MF_v = zeros(H,W);
for m=1:H
    for n=1:W
        x=-(n-((W-1)/2)-1)*w; y=-(m-((H-1)/2)-1)*h;
        % Find object point.
        objpnt = findobjpnt(x, y, facets, bg, -tc, inv(Rc));
        % Compute motion flow.
        [MF_u(m,n), MF_v(m,n)] = computemotionflow(objpnt, to, ro);
    end
    waitbar(m/H);
end
close(hnd);

% Draw motion field.
drawmotionfield(MF_u, MF_v, h, w, c);
str = sprintf(['Motion Field: To[%5.2f,%5.2f,%5.2f], ',...
```

```
        'Ro[%5.2f,%5.2f,%5.2f]\n',...
        'bg[%5.2f,%5.2f,%5.2f], ',...
        'Tc[%5.2f,%5.2f,%5.2f], ',...
        'Rc[%5.2f,%5.2f,%5.2f]'],...
    to(1), to(2), to(3),...
    ro(1), ro(2), ro(3),...
    bg(1), bg(2), bg(3),...
    tc(1), tc(2), tc(3),...
    rc(1), rc(2), rc(3));
title(str);

% Generate random gradient direction.
GD = rand(H,W)*2*pi;
GD_u = cos(GD);
GD_v = sin(GD);

% Project motion flow onto gradient direction.
MF_GD = (MF_u.*GD_u+MF_v.*GD_v);
MF_GD_u = MF_GD.*GD_u;
MF_GD_v = MF_GD.*GD_v;
drawmotionfield(MF_GD_u, MF_GD_v, h, w, c);
str = sprintf(['Normal Motion Field: To[%5.2f,%5.2f,%5.2f], ',...
        'Ro[%5.2f,%5.2f,%5.2f]\n',...
        'bg[%5.2f,%5.2f,%5.2f], ',...
        'Tc[%5.2f,%5.2f,%5.2f], ',...
        'Rc[%5.2f,%5.2f,%5.2f]'],...
    to(1), to(2), to(3),...
    ro(1), ro(2), ro(3),...
    bg(1), bg(2), bg(3),...
    tc(1), tc(2), tc(3),...
    rc(1), rc(2), rc(3));
title(str);

%----------------------------------The  End---------------------------------%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                          COPYRIGHT (C) LJ 2000
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Background plane.
bg = [10,0.5,0.8]';
% Cube position.
tc = [0.5,0,5]';
rc = [0.5,0.6,-0.4]';
% Observer motion.
to = [0.05,0.03,0.6]';
ro = [0,0,0]';
```

```
ro = [0.005,0.01,-0.02]';
% Grid size for motion field.
h=0.05; w=0.05;
% Grid number for motion field.
H=21; W=21;

%----------------------------------The  End---------------------------------%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                          COPYRIGHT (C) LJ 2000
% R=rotationfromangle(a,b,c)
%
% INPUT
%  a, b, c : rotation angles around x-, y-, and z-axis.
%
% OUTPUT
%  R : 3x3 rotation matrix.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function R=rotationfromangle(a,b,c)
% ------------------------------------------------------------------------ %

Rx = [
    1, 0, 0;
    0, cos(a), -sin(a);
    0, sin(a), cos(a)];
Ry = [
    cos(b), 0, -sin(b);
    0, 1, 0;
    sin(b), 0, cos(b)];
Rz = [
    cos(c), -sin(c), 0;
    sin(c), cos(c), 0;
    0, 0, 1];
R = Rz*Ry*Rx;

%----------------------------------The  End---------------------------------%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                          COPYRIGHT (C) LJ 2000
% c=position8vertices(T, R)
%
% INPUT
%  T : 1x3 translation.
%  R : 3x3 rotation matrix.
```

```
%
% OUTPUT
%  c : 3x8 vertices.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function c=position8vertices(T, R)
% ------------------------------------------------------------------------ %

% vertices 1 -- 8
c = [
    +1, +1, +1;
    +1, +1, -1;
    +1, -1, -1;
    +1, -1, +1;
    -1, -1, +1;
    -1, -1, -1;
    -1, +1, -1;
    -1, +1, +1]';

c = R * c;
c = c + T*ones(1,8);

%----------------------------------The  End-----------------------------------%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                          COPYRIGHT (C) LJ 2000
% b=isvalidsetting(c, bg)
%
% INPUT
%  c : 3x8 vertices.
%  bg: [Z0, p, q] for background plane Z=Z0+pX+qY.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function b=isvalidsetting(c, bg)
% ------------------------------------------------------------------------ %

% Is the background plane before camera? (Z0>0?)
% Is all 8 vertices of the cube in front of camera? (Zc>=0?)
% Is all 8 vertices of the cube in front of the background plane?  (Z<=Z(X,Y)?)

Z0=bg(1); p=bg(2); q=bg(3);

if (Z0<=0)
    b=0;
    disp('>>Error: background behind camera');
```

```
    return;
end

Zc = c(3,:);
if (sum(Zc>=0) < 8)
    b=0;
    disp('>>Error: one or moer vertices behind camera');
    return;
end

XYc = c(1:2, :);
Zxy = Z0 + [p, q]*XYc;
if (sum(Zc<=Zxy)<8)
    b=0;
    disp('>>Error: one or more vertices behind background');
    return;
end

b=1;
return;

%---------------------------------The  End---------------------------------%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                          COPYRIGHT (C) LJ 2000
% facets=computefacets(c)
%
% INPUT
%  c : 3x8 vertices
%
% OUTPUT
%  facets : 7x6 information about facets.  Each column contains # of 4 vertic-
%           es, and 3 coef for facet plane equation (aX+bY+cZ=1).
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function facets=computefacets(c)
% ------------------------------------------------------------------------ %

facets = [
    1,2,3,4,0,0,0;
    1,2,7,8,0,0,0;
    1,4,5,8,0,0,0;
    2,3,6,7,0,0,0;
    3,4,5,6,0,0,0;
    5,6,7,8,0,0,0]';
```

```
for i=1:6
    facets(5:7,i) = inv(c(:,facets(1:3,i))') * [1,1,1]';
end

%----------------------------------The  End---------------------------------%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                          COPYRIGHT (C) LJ 2000
% objpnt = findobjpnt(x, y, facets, bg, Tc, Rc)
%
% INPUT
%  x, y : point on image plane.
%  facets : 7x6 information about facets.  Each column contains # of 4 vertic-
%           es, and 3 coef for facet plane equation (aX+bY+cZ=1).
%  bg : [Z0, p, q]' for background plane S.T. Z=Z0+pX+qY.
%  Tc, Rc : Inverse cube position translation and rotation.
%
% OUTPUT
%  objpnt : 1x3 [X,Y,Z] on either one of 6 facets of the cube or background.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function objpnt = findobjpnt(x, y, facets, bg, Tc, Rc)
% ------------------------------------------------------------------------ %

% Find (X,Y,Z) on background.
Z0=bg(1); p=bg(2); q=bg(3);
Z = Z0/(1-p*x-q*y);

for i=1:6 %facets
    % Find intersection point.
    a=facets(5,i); b=facets(6,i); c=facets(7,i);
    Znew = 1/(a*x+b*y+c);
    Xnew = x*Znew;
    Ynew = y*Znew;
    % Determine if on cube.
    objpnt_ = Rc*([Xnew, Ynew, Znew]' + Tc);
    X_=objpnt_(1); Y_=objpnt_(2); Z_=objpnt_(3);
    diff = max([abs(X_)-1, abs(Y_)-1, abs(Z_)-1]);
    if (diff < eps*10000) % if yes.
        % Pick the nearest one.
        Z = min(Znew, Z);
    end
end

objpnt = [x*Z,y*Z,Z]';
```

```
%----------------------------------The  End---------------------------------%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                          COPYRIGHT (C) LJ 2000
% [u, v] = computemotionflow(objpnt, to, ro)
%
% INPUT
%  objpnt : [X,Y,Z]'.
%  to : translation motion of observer.
%  ro : rotation motion of observer.
%
% OUTPUT
%  u, v : motion flow at image point (x,y,1), where x=X/Z, y=Y/Z.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [u,v] = computemotionflow(objpnt, to, ro)
% ------------------------------------------------------------------------ %

v3d = -to - cross(ro, objpnt); % 3d world velocity.
X = objpnt(1); Y = objpnt(2); Z = objpnt(3);
X_ = v3d(1); Y_ = v3d(2); Z_ = v3d(3);
u = X_/Z - X*Z_/(Z*Z);
v = Y_/Z - Y*Z_/(Z*Z);

%----------------------------------The  End---------------------------------%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                          COPYRIGHT (C) LJ 2000
% drawmotionfield(MF_u, MF_v)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function drawmotionfield(MF_u, MF_v, h, w, c)
% ------------------------------------------------------------------------ %

figure;
axis equal;
[H, W] = size(MF_u);

h_=h/20; w_=w/20;

for m=1:H
    for n=1:W
        x=-(n-((W-1)/2)-1)*w; y=-(m-((H-1)/2)-1)*h;
        hnd = line([x, x+MF_u(m,n)], [y, y+MF_v(m,n)]);
```

```
        set(hnd,'color','red');
        hnd = line([x+w_,x+w_,x-w_,x-w_,x+w_], [y+h_,y-h_,y-h_,y+h_,y+h_]);
        set(hnd,'color','black');
    end
end

%return;

ci = c(1:2,:)./[c(3,:); c(3,:)];
tmp=[
    1,2,3,4,1;
    5,6,7,8,5;
    1,4,5,8,1;
    2,3,6,7,2];
for i=1:4
    hnd=line(ci(1,tmp(i,:)), ci(2,tmp(i,:)));
    set(hnd,'color','blue');
end

%----------------------------------The  End---------------------------------%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                          COPYRIGHT (C) LJ 2000
%  Main routine for HW #11 problem 4
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;
close all;

% Load 15 images.
for i=2:16
    I(:,:,i-1)=imread(sprintf('%02d.tif',i));
end
I = double(I);

% Apply spatiotemporal Gaussian filter with delta=1.5 pixel-frames, and kernel
% size 11 x 11 x 11.
g = (exp(-[-5:5].^2/(2*1.5*1.5))/sqrt(2*pi))';

I=filter(g,1,I,[],3);
I=I(:,:,11:15); % the useful frames.
I=filter(g,1,I,[],1); I(1:10,:,:)=[];
I=filter(g,1,I,[],2); I(:,1:10,:)=[]; % trim imprecise part.

% Compute spatiotemporal derivatives.
d = [-1,8,0,-8,1]/12;
```

```
% Temporal derivatives.
It=filter(d,1,I,[],3);
It=It(:,:,5);
% Spatial derivatives.
Ix=filter(d,1,I(:,:,3),[],2);
Iy=filter(d,1,I(:,:,3),[],1);
% Trim imprecise part so as to alian Ix, Iy, It.
Ix(1:4,:)=[]; Ix(:,1:4)=[];
Iy(1:4,:)=[]; Iy(:,1:4)=[];
It(1:2,:)=[]; It(:,1:2)=[];

% For points with large spatial derivatives, compute weighted LS solution of
%  motion flow (u,v).
% Preparation.
w=[0.0625,0.25,0.375,0.25,0.0625];
W=w'*w;
W=W*W;
W=reshape(W,1,25);
W2=diag(W);
% Compute motion flow for each point.
[M,N]=size(Ix);
MF_u=zeros(M,N); MF_v=zeros(M,N); % motion field
NF_u=zeros(M,N); NF_v=zeros(M,N); % normal field
h=10; w=10;
for m=3:h:M-2
    for n=3:w:N-2
       if (norm([Ix(m,n); Iy(m,n)]) > 1) % only for large derivatives.
            Ax=reshape(Ix(m-2:m+2, n-2:n+2), 1, 25);
            Ay=reshape(Iy(m-2:m+2, n-2:n+2), 1, 25);
            A = [Ax; Ay]';
            b = -reshape(It(m-2:m+2, n-2:n+2), 25, 1);
            MF = pinv(A'*W2*A)*A'*W2*b;
            MF_u(m,n)=MF(1); MF_v(m,n)=MF(2);

            s = -It(m,n)/norm([Ix(m,n); Iy(m,n)]);
            NF_u(m,n) = s*Ix(m,n)/norm([Ix(m,n); Iy(m,n)]);
            NF_v(m,n) = s*Iy(m,n)/norm([Ix(m,n); Iy(m,n)]);
        end
    end
end
MF_u = MF_u(3:h:M-2, 3:h:N-2);
MF_v = MF_v(3:w:M-2, 3:w:N-2);
NF_u = NF_u(3:h:M-2, 3:h:N-2);
NF_v = NF_v(3:w:M-2, 3:w:N-2);

% Draw motion field.
drawmotionfield(MF_u, -MF_v, 5, 5);
%quiver([1:size(MF_u,2)], [size(MF_u,1):-1:1], MF_u, -MF_v);
```

```
title('Motion field');
drawmotionfield(NF_u, -NF_v, 5, 5);
title('Normal field');

%---------------------------------The  End---------------------------------%
```