EE392J Final Project

Gunnlaugur Thor Briem
Mar. 13 2000

Content-Based Occlusion-Adaptive Mesh Design for Motion Compensation

ABSTRACT

For mesh-based motion compensation, image warping and other image processing tasks involving mesh structure, two major issues are robustness to occlusion, and the need for an initial mesh conforming to objects and features in the image signal. In this report I describe an automated mesh-design procedure which aligns the mesh to image content and to the boundaries of occlusion regions, and does not allow mesh nodes or their displacement estimates to be placed in regions of background-to-be-covered (BTBC) or uncovered background (UB).

1 INTRODUCTION

Mesh-based methods have been proposed as a promising alternative for the block-correlation methods that currently dominate video codecs. Block-based methods suffer from discontinuities and blocking artifacts due to the assumption of a translational motion model. Generalized block matching allows for other motion models, but still exhibits discontinuities at block boundaries.

Mesh-based methods are specifically designed as continuity-preserving transforms, usually as piecewise affine warps of the image plane. The disadvantage to these methods, however, is that their efficacy is quite dependent on the conformance of the mesh elements to regions of similar motion in the image, i.e. the edges of the mesh need to fall on the boundaries between regions of motion, introducing the problem of motion segmentation – content-based processing. In morphing for computer graphics, this need has been met by human intervention, which is obviously undesirable in a video codec (and is also the reason high-quality video morphing is very costly in man-hours).

It is therefore desirable to apply an automated process for generating meshes that conform to image features sufficiently for motion compensation to be performed without disturbing failed-segmentation artifacts. Such a system is described here.

2 OCCLUSION ADAPTATION

In a rough categorization with respect to the transition between two frames, there are two types of occlusion regions: in the previous frame, the background to be covered (BTBC), and in the subsequent frame, the uncovered background (UB). These regions are generally estimated as “model failure” regions, i.e. where motion-compensated prediction fails.

Thus for our purposes we'll estimate the BTBC region in frame k as the region of maximum motion compensation error using a dense motion estimate from frame $k+1$ to frame k, and then exclude that region from mesh node creation (since points in that region have no correspondence in the following frame). When our mesh is used for motion compensation, equivalently, node motion vectors must be prohibited from entering the UB region, because by definition it consists of points that have no correspondence in the previous frame.

On the other hand, regions adjacent to the BTBC and UB regions *do* have point correspondences, thus we would like our mesh segmentation to separate them effectively from the occlusion regions. This means we need mesh edges to conform closely to the boundary of the occlusion regions (as well as to boundaries between different motion regions in the unoccluded image).

3 MESH DESIGN

Here I will describe the procedure for deriving a mesh from two frames. It may be split in two: the BTBC detection and the node point generation and meshing.

3.1 BTBC Region Detection

The BTBC regions in frame k are all (nontrivial) clusters of pixels that are occluded in frame $k+1$ (i.e. are not correlated with any part of frame $k+1$). Note that this includes also regions that are moving offscreen as opposed to being covered by other image objects.

Following the abovementioned guideline of estimating the BTBC as the region of model failure in a backwards motion compensated prediction from frame $k+1$, we first want to obtain a dense motion field to use for this initial prediction. My method of doing that[1] was to first perform block correlation at a courser scale, then interpolate the resulting estimates and pass them through 10 iterations of Horn and Schunck's well-known algorithm. The smoothness constraint parameter c was chosen as 55, by computing the motion-compensated prediction from the motion estimates for a range of different c, for a representative frame pair from the sequence under consideration (“salesman”), and choosing the one that resulted in the largest PSNR between the MC-predicted frame $k+1$ and the actual one. The curve is shown in figure 1.

[1]partly because of a pet peeve with Lucas-Kanade's method

The computational load associated with the initial block correlation motion estimates is considerable, even when computed at relatively large intervals. Therefore I first compute the *change detection mask* (CDM), an estimate of which regions of the image actually underwent a large change in intensity between the two frames. It is assumed that regions of low change in intensity will correspond to regions of zero motion, hence we simply set the motion vectors equal to zero there to save computational effort.

The CDM is determined by first thresholding the direct frame difference $f2 - f1$ and applying a postprocessing procedure: a median-filter with a 5×5 kernel, morphological closing with a 3×3 kernel, morphological opening with a 3×3 kernel, and elimination of contiguous pixel ranges smaller than a threshold (25 pixels in my experiments). This effectively gets rid of much of the inevitably large amount of binary noise in the thresholded frame difference.

Now block correlation is performed at small intervals in the CDM, and the resulting coarsely sampled motion vector field is interpolated to a dense field and passed through N iterations of the Horn-Schunck algorithm. The "optimal" smoothness parameter c is somewhat hand-wavingly determined as 31 for the "salesman" sequence by graphing the prediction PSNR for a range of smoothness parameters as shown in figure 1.

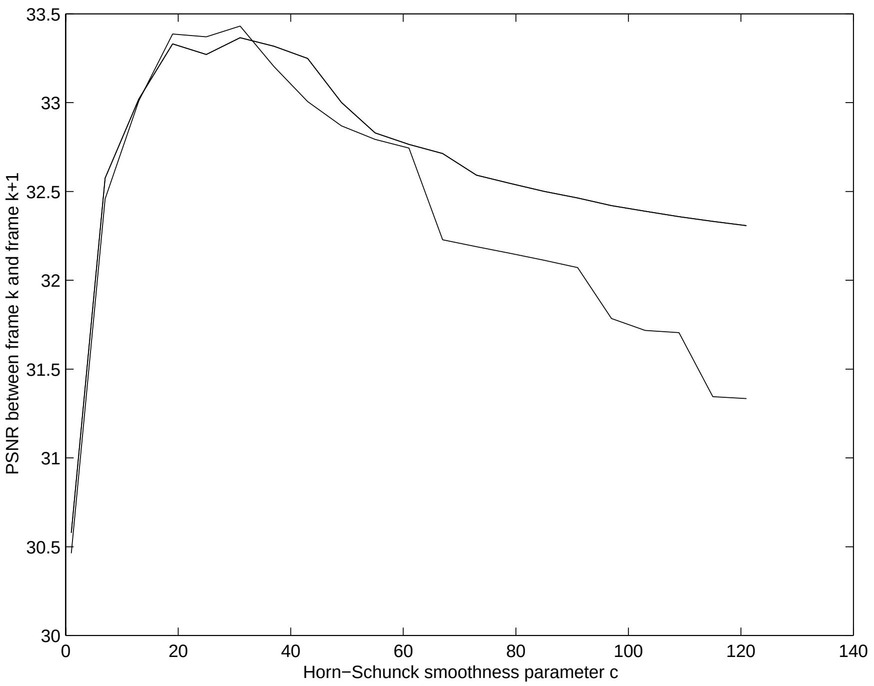

Figure 1: Horn-Schunck motion compensated prediction PSNR as a function of c

7 This smooth, dense motion estimate is used to MC-predict frame k from frame $k + 1$, and the resulting DFD is regarded as an indicator of model failure. It is thresholded and postprocessed in the same way as the change detection mask before, and henceforth called the BTBC region.

3.2 Node generation and meshing

The node point generation algorithm relies on a combination of the intensity gradient of frame k and the energy in the displaced frame difference produced by motion-compensating frame k from frame $k+1$ using the dense motion estimate generated during BTBC region detection above. At each iteration, one node point is generated at the highest-gradient "legal" point left over, where "legal" is defined by the iteration. More precisely:

1. Initially, all pixels are defined "legal" except for those in the previously computed BTBC, and the 1-norm of the image gradient is computed at all points:

   $$C(x,y) = |D_x I(x,y)| + |D_y I(x,y)|$$

   This is used as merit function for selecting node points, so that they, and thus the edges between them, tend to coincide with spatial boundaries in the image. See figure 6

2. The mean-square DFD is computed:

   $$\mathrm{DFD}_{\mathrm{avg}} = \frac{\sum \mathrm{DFD}^2}{K}$$

   where the summation is over all currently legal points.

3. The highest-C legal point is chosen as a node, with the constraint that it must not be closer to any other node than a given threshold (I made it 3.2 pixels).

4. A circle is grown around this node point, just large enough for the encircled energy in the DFD to be greater than $\mathrm{DFD}_{\mathrm{avg}}$. All pixels within this circle are made illegal. This makes sure nodes are not chosen too close together.

5. Loop from 2) until the required number of node points (which I made 200) is reached, or there are no more legal points.

Finally, the obtained set of node points is simply meshed using Delaunay triangularization. Node points are first added along the border, so as to make the mesh cover the whole image area.

4 IMPLEMENTATION

This procedure was implemented in Matlab and tried out on a frame pair from the "salesman" sequence. The original frames are shown in figures 2 and 3. The code is supplied in the appendix. The following figures illustrate the procedure and give an example of the results.

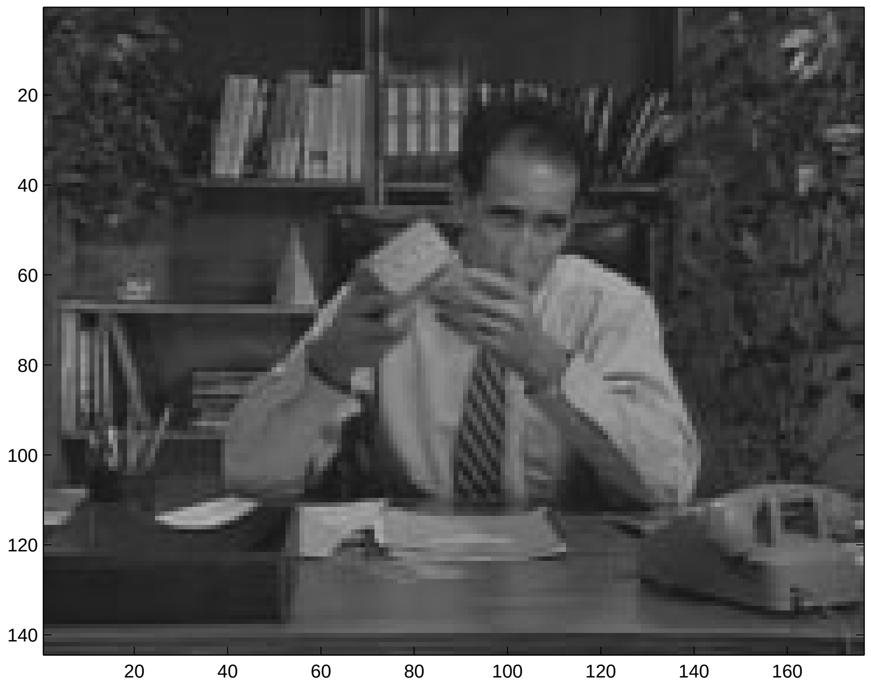

Figure 2: Former frame from “salesman”

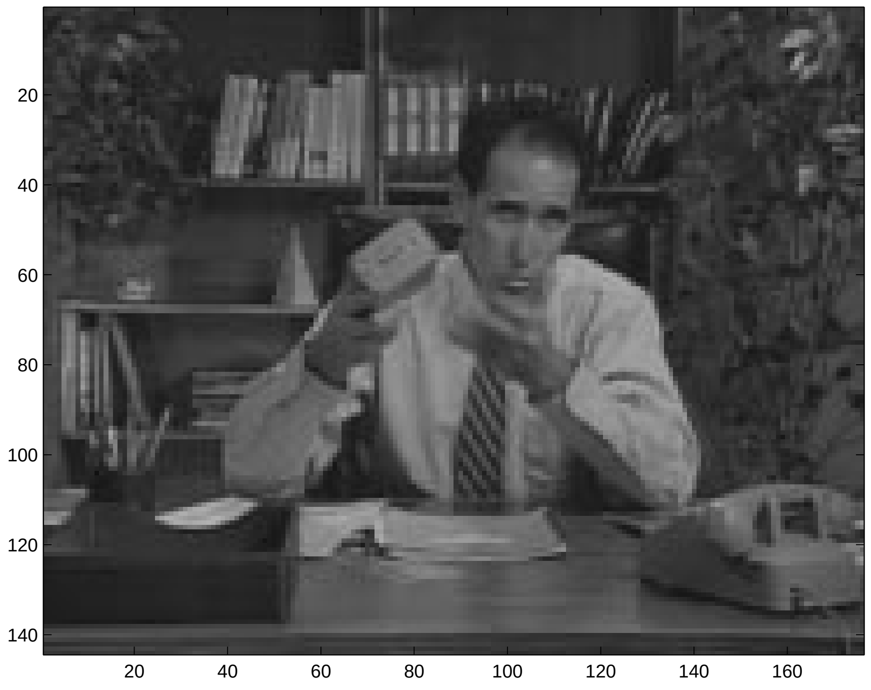

Figure 3: Latter frame from “salesman”

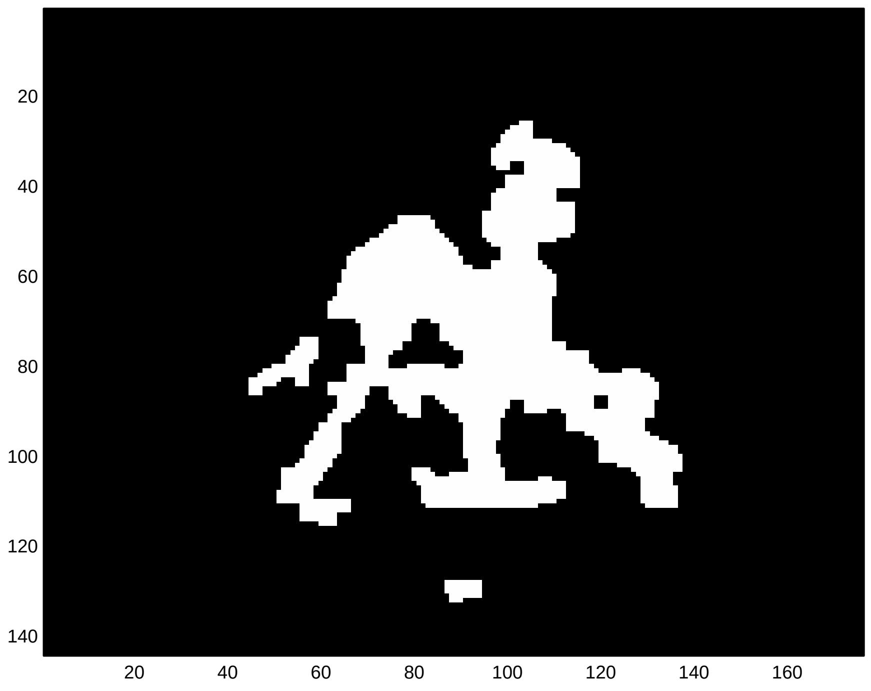

Figure 4: Change detection mask for a frame pair from the salesman sequence

Figure 4 shows the CDM. Note the cleanliness due to the careful postprocessing stage . . . the thresholded frame difference is, of course, quite dirty.

Figure 5 shows the estimated BTBC regions. Note, again, the cleanliness. This is important for a medium-scaled mesh to work out well.

Figure 6 shows the merit function for node point selection, i.e. the 1-norm of the former frame intensity.

Figure 7 shows the pattern of legal and illegal points (white are illegal) following the node point selection procedure. The illegal circles are smallest where temporal dependence is highest, thus promoting the densest node point allocation.

Figure 8 shows the "distance to nearest node point" measure, which is kept track of during the node selection procedure for performance, rather than computing the distance to all node points in each iteration.

Finally, figures 9 and 10 show the final node allocations and resulting Delaunay triangulation. Note the successfull conformance to image features, most readily apparent on the arms of the salesman himself, and also note the higher density of node points at his fast-moving hands, except in the estimated BTBC region, which gets no nodes.

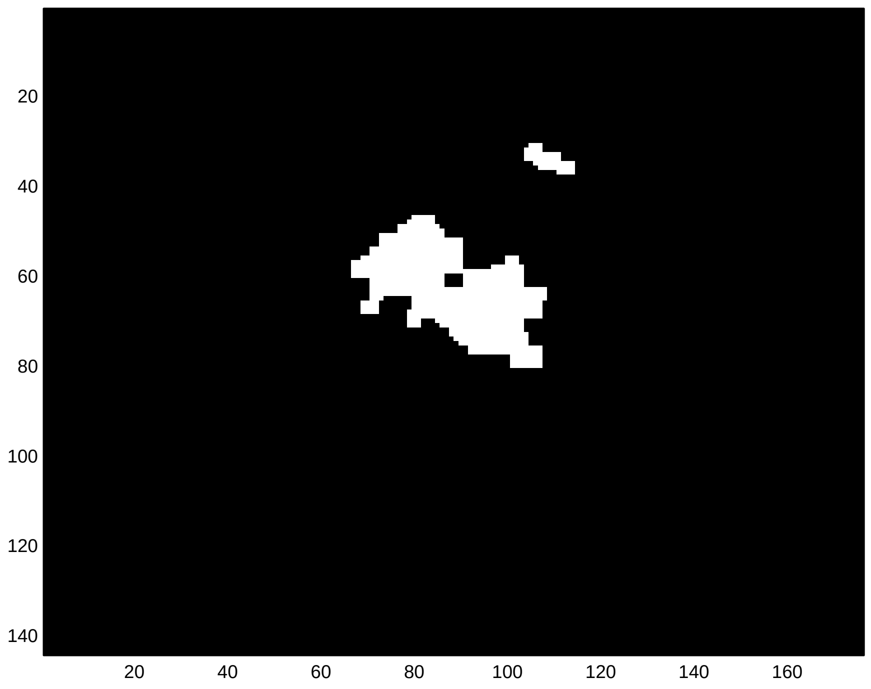

Figure 5: Estimated BTBC region

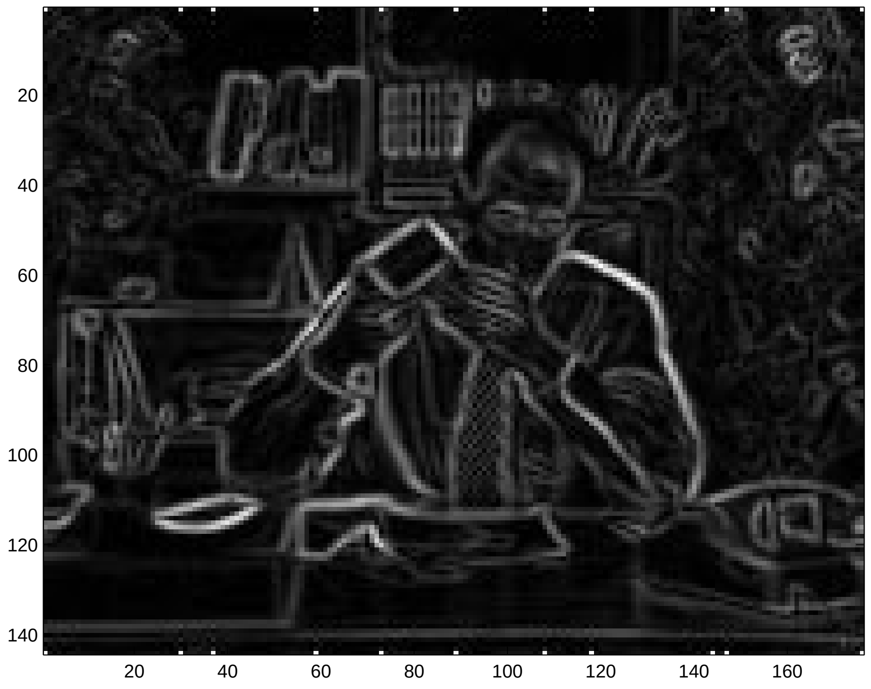

Figure 6: Merit function (1-norm of gradient) over the image

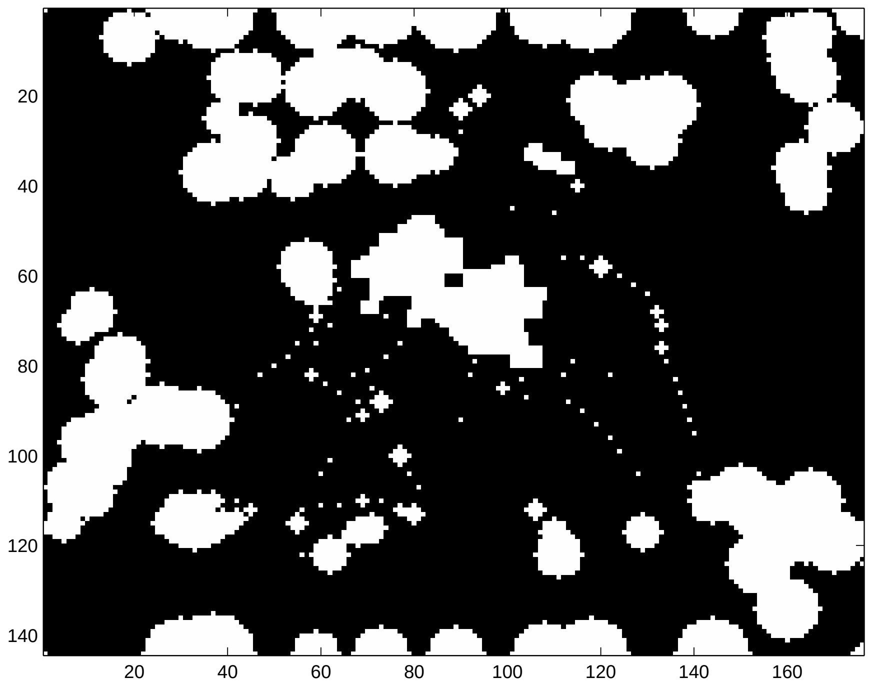

Figure 7: Illegal node regions at end of node selection procedure

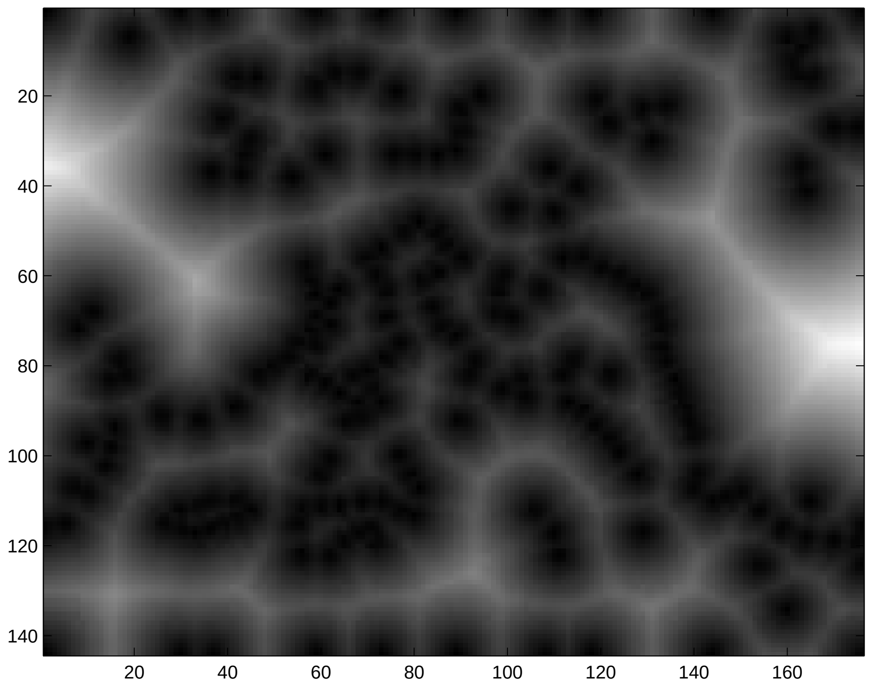

Figure 8: Distance to nearest node point

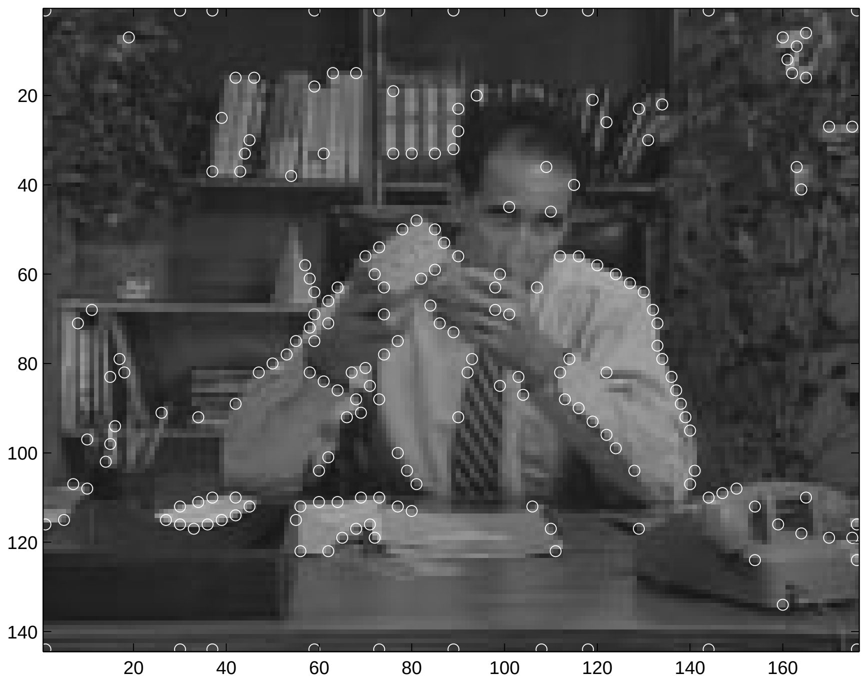

Figure 9: The salesman image (former frame) with the selected node points overlaid

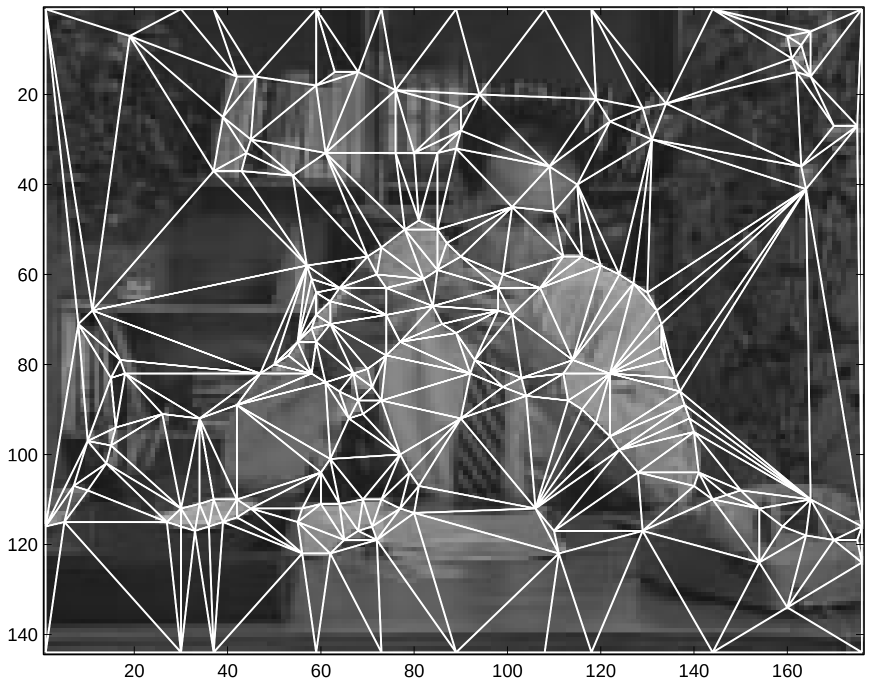

Figure 10: The salesman image with the designed mesh overlaid

5 CONCLUSION

The effectiveness of the mesh design algorithm under consideration has been demonstrated by example, as well as argued by reason. The algorithm is robust to noisy images due to the morphological cleanup procedure, and execution speed (in an unoptimized Matlab implementation) is acceptable, approx. one minute in total on a standard desktop PC — excluding the initial block motion estimation, which is the dominant factor in my experiments.

There is much of interest to try out in connection with this method, in particular piecewise affine motion compensation over a long video sequence with a mesh that updates to conform to changing image content, is interesting. This would involve removing mesh points that get predicted into an occlusion region, and creating new ones on uncovered-background regions. This raises various issues such as intelligent control of the number of node points and criteria for designing the new parts of the mesh in the absence of prior information about the just-unoccluded area. Alternatively, a new mesh can be designed at each frame, but this holds little promise for video coding efficiency. Applications in semi-automatic image warping and morphing are easy to anticipate, though, and this begs the question of how to design two meshes separately with constraints so as to determine feature correspondences automatically.

This project was based heavily on the first-cited work in the references, my work being mostly an implementation thereof (albeit with some changes and improvements), rather than an independent "contribution to knowledge."

6 REFERENCES

1. Y. Altunbasak and A. M. Tekalp "Occlusion-adaptive, Content-Based Mesh Design and Forward Tracking". *IEEE Trans. Image Processing*, vol. 6, no. 9, pp. 1270-1280, Sept. 1994

2. Y. Altunbasak and A. M. Tekalp "Closed-form Connectivity-Preserving Solutions for Motion Compensation using 2-D meshes." *IEEE Trans. Image Processing*, vol. 6, no. 9, pp. 1255-1269, Sept. 1994

3. A. M. Tekalp, P. van Beek, C. Toklu, B. Günsel "Two-Dimensional Mesh-Based Visual-Object Representation for Interactive Synthetic/Natural Digital Video." *Proc. IEEE*, vol. 86, no. 6, pp. 1029-1051, June 1998 *IEEE Trans. Image Processing*, vol. 6, no. 9, pp. 1255-1269, Sept. 1994

4. J. Apostolopoulos, and S. Wee, *Lecture notes in EE392J*, Stanford University, Jan.-Mar. 2000.

7 APPENDIX - CODE

bcme.m

```
% [U,V] = BCME(frame1,frame2,x_coords,y_coords,B,r)
% block correlation motion estimation with blocksize 2B+1
% at the coordinates x_coords,y_coords (vectors, not
% meshgrid-type things) over a search range of +-r pixels
% in each dimension, with the MAD criterion. By default
% B=7 and r=15.
function [mu,mv] = bcme(f1,f2,x,y,B,r)

if(size(f1)~=size(f2)),
   error('Frames need to be the same size!');
end;

if(length(size(f1))~=2),
   error('Frames need to be 2-D matrices!');
end;

if(nargin<4),
   error('Need at least 4 arguments!');
end;

if(nargin<5),
   B=7;
end;

if(nargin<6),
   r=7;
end;

NX=size(f1,2)
NY=size(f1,1)
nX=length(x);
nY=length(y);

if(max(x)>NX-B-r | max(y)>NY-B-r | min(x)<1+B+r | min(y)<1+B+r),
error('Given coordinates would make me hit the boundaries!');
end;

mu=zeros(length(y),length(x));
mv=zeros(length(y),length(x));
score=zeros(r+r+1,r+r+1);
[gwx,gwy]=meshgrid(-r:r,-r:r);
```

```
igausswind=exp(-(gwx.*gwx+gwy.*gwy)/(16*B^2));
igausswind=max(max(igausswind))./igausswind;
% Loop over the blocks
disp(['We''ll be done at i=' num2str(nX)]);
for i=1:nX,
   disp(['i=' num2str(i)]);
   for j=1:nY,
      xc=x(i)-B:x(i)+B; yc=y(j)-B:y(j)+B;
      currmin=1e10;
      for u=-r:r,
         for v=-r:r,
            absdiffo=sum(sum(abs(f2(yc+v,xc+u)-f1(yc,xc))))*igausswind(u+r+1,v+r+1);
            if(absdiffo<currmin),
               currmin=absdiffo;
               currminu=u;
               currminv=v;
            end;
         end;
      end;
      mu(j,i)=currminu;
      mv(j,i)=currminv;
   end;
end;
```

detectregion.m

```
% region = detectregion(f,stddevthresh,minregionsize)
% Creates a boolean map of regions whose intensity deviation from
% the mean exceeds stddevthresh standard deviations. The regions
% are median-filtered, morphologically closed and opened, and
% those that are smaller than minregionsize pixels are thrown out.
function x = detectregion(f,stddevthresh,minregionsize)

x=abs(f-mean2(f))>(std2(f)*stddevthresh);
x=bwmorph(x,'close');
x=bwmorph(x,'open');
[labels,num]=bwlabel(x);
for i=1:num,
regioninds=find(labels==i);
if(length(regioninds)<minregionsize),
x(regioninds)=0;
end;
end;
```

drawmesh.m

```
function drawmesh(tri,xn,yn,col,linewidth)

if(nargin<5),
   linewidth=1.5;
if(nargin<4),
      col=[1 1 1];
      if(nargin<3),
         error('Need at least 3 arguments!');
      end;
   end;
end;

line([[xn(tri(:,1))';xn(tri(:,2))'] [xn(tri(:,2))';xn(tri(:,3))']   ...
      [xn(tri(:,3))';xn(tri(:,1))']],[[yn(tri(:,1))';yn(tri(:,2))'] ...
      [yn(tri(:,2))';yn(tri(:,3))'] [yn(tri(:,3))';yn(tri(:,1))']], ...
      'LineWidth',linewidth,'Color',col);
```

est_gradient.m

```
% [Sx,Sy] = est_gradient(f);
% Estimates the image intensity gradient at each pixel
function [Sx,Sy] = est_gradient(f),

[NY,NX]=size(f);

% Filter for spatial gradient estimation
[fx,fy]=meshgrid(-1:1,-1:1);
filto=exp(-(fx.^2+fy.^2)); filto=filto/sum(sum(filto));
f=filter2(filto,[f(1,1) f(1,:) f(1,end);          ...
          f(:,1) f f(:,end);                      ...
          f(end,1) f(end,:) f(end,end)],'valid');

% Estimate spatial gradient
Sx=[diff(f(:,1:2)')' 0.5*(f(:,3:end)-f(:,1:end-2))  diff(f(:,end-1:end)')'];
Sy=[diff(f(1:2,:));  0.5*(f(3:end,:)-f(1:end-2,:)); diff(f(end-1:end,:))];
```

hsme.m

```
% [mu,mv] = hsme(f1,f2,mu,mv,N,c)
% Performs dense motion estimation by N Horn-Schunck relaxations
% with smoothness emphasis c. Default N is 10, default c is 15.
% mu and mv are initial motion estimates, computed for example
% using block matching and interpolation.
function [mu,mv] = hsme(f1,f2,mu,mv,N,c)

% Sanity checks
if(nargin<4),
error('Need at least 4 arguments');
end;

if(nargin<5),
N=10;
end;

if(nargin<6),
c=15;
end;

% Sanity checking
[NY,NX]=size(f1);
if(size(f2)~=[NY,NX]),
error('Hey dude, frames must be the same size!');
end;

% Temporal derivative estimation
St=f2-f1;

% Filter for spatial gradient estimation
[fx,fy]=meshgrid(-2:2,-2:2);
filto=exp(-(fx.^2+fy.^2)/5); filto=filto/sum(sum(filto));
f1=filter2(filto,f1);

% Estimate spatial gradient
Sx=[diff(f1(:,1:2)')' 0.5*(f1(:,3:end)-f1(:,1:end-2)) diff(f1(:,end-1:end)')'];
Sy=[diff(f1(1:2,:));0.5*(f1(3:end,:)-f1(1:end-2,:));diff(f1(end-1:end,:))];
SxSx=Sx.*Sx; SxSy=Sx.*Sy; SySy=Sy.*Sy; SxSt=Sx.*St; SySt=Sy.*St;

if(N>0),
c=c*c;
normsqrgrad=SxSx+SySy;
idenom=1./(c+normsqrgrad);
end;

[fx,fy]=meshgrid(-1:1,-1:1);
filt=exp(-(fx.^2+fy.^2)); filt=filt/sum(sum(filt));
while(N>0),
% filter with a fairly sharp 3x3 gaussian kernel
muf=filter2(filt,[mu(1,1)   mu(1,:)   mu(1,end);
                          mu(:,1)   mu        mu(:,end);
                          mu(end,1) mu(end,:) mu(end,end)],'valid');
mvf=filter2(filt,[mv(1,1)   mv(1,:)   mv(1,end);
                          mv(:,1)   mv        mv(:,end);
                          mv(end,1) mv(end,:) mv(end,end)],'valid');

mu=muf-(SxSx.*muf+SxSy.*mvf+SxSt).*idenom;
mv=mvf-(SxSy.*muf+SySy.*mvf+SySt).*idenom;

N=N-1;
end;
```

loadem.m

```
%[y1,u1,v1]=read_frame_qcif('seq/qcif_sequences/mthr_dotr.qcif',nf1);
%[y2,u2,v2]=read_frame_qcif('seq/qcif_sequences/mthr_dotr.qcif',nf2);
[y1,u1,v1]=read_frame_qcif('seq/salesman.qcif',nf1);
[y2,u2,v2]=read_frame_qcif('seq/salesman.qcif',nf2);
[NY,NX]=size(y1);
u1b=interp2(u1,1); u1b=[u1b u1b(:,end); u1b(end,:) u1b(end,end)];
v1b=interp2(v1,1); v1b=[v1b v1b(:,end); v1b(end,:) v1b(end,end)];
u2b=interp2(u2,1); u2b=[u2b u2b(:,end); u2b(end,:) u2b(end,end)];
v2b=interp2(v2,1); v2b=[v2b v2b(:,end); v2b(end,:) v2b(end,end)];
```

makemesh.m

```
% makemesh(xn,yn) -- makes a triangle map and an adjacency matrix for a
%                    Delaunay triangulation of the given set of nodes.
%                    This is blatant redundancy -- tri and adj express
%                    the same information, but they have different
%                    utility for different purposes.
function [tri,adj] = makemesh(xn,yn)

tri=delaunay(xn,yn);
numnodes=length(xn);
[N,dummy]=size(tri);
adj=spalloc(numnodes,numnodes,3*N); % never more than 3N edges
for i=1:N,
   adj(tri(N,1),tri(N,2))=1;
   adj(tri(N,2),tri(N,3))=1;
   adj(tri(N,3),tri(N,1))=1;
   adj(tri(N,3),tri(N,2))=1;
   adj(tri(N,1),tri(N,3))=1;
   adj(tri(N,2),tri(N,1))=1;
end;
```

mcp.m

```
% MCP(f1,mu,mv) - motion-compensated prediction of next
%                 frame given the motion vectors (mu,mv).
function f2 = mcp(f1,mu,mv)

[NY,NX]=size(f1);
[xg,yg]=meshgrid(1:NX,1:NY);
xgp=xg-mu;
ygp=yg-mv;
xgp(xgp<1)=1; xgp(xgp>NX)=NX;
ygp(ygp<1)=1; ygp(ygp>NY)=NY;

f2 = griddata(xgp,ygp,f1,xg,yg,'linear');
f2(isnan(f2))=0;
```

me.m

```
% out=me(f1,f2) returns a motion vector estimate produced
% by block correlation on the change detection mask, followed
% by Horn-Schunck iterations.
function [uh,vh] = me(f1,f2)

[NY,NX]=size(f1);

% Compute change detection mask
cdm  = compute_cdm(f1,f2,25);

% Do block correlation motion estimation on the mask
B=7;  % block size for BC ME
r=15; % search range for BC ME
spars=4; % sparsity of BC ME
[ix,iy]=find(cdm~=0);
xmin=max(1+B+r,min(ix)-spars); xmax=min(NX-B-r,max(ix)+spars);
ymin=max(1+B+r,min(iy)-spars); ymax=min(NY-B-r,max(iy)+spars);
xb=xmin:spars:xmax; yb=ymin:spars:ymax;
[ub,vb] = bcme(f1,f2,xb,yb,B,r);

% Refine the motion estimate to a dense one by Horn-Schunck
ud=interp2(xb,yb,ub,1:NX,(1:NY)','cubic');
```

```
vd=interp2(xb,yb,vb,1:NX,(1:NY)','cubic');
ud(isnan(ud))=0; vd(isnan(vd))=0;
[uh,vh]=hsme(f1,f2,ud,vd,10,31);
```

selectnodes.m

```
% [xn,yn] = selectnodes(f1,f2,mu,mv,N);
% selects up to N nodes for a mesh representation based
% on f1, f2 and the motion vectors mu,mv from f1 to f2.
function [xn,yn]=selectnodes(f1,f2,mu,mv,N,cdm),

[NY,NX]=size(f1);
[xg,yg]=meshgrid(1:NX,1:NY);
mindist2=10;

% Initialization
%marked=zeros(size(f1)); % label all pixels 'unmarked'
marked=cdm;
xn=[]; yn=[]; % start with no node points
dist2tonearest=repmat(inf,size(f1)); % sqr distance to nearest node point
indlegal=1:(NY*NX);

% precompute node selection function
[Sx,Sy]=est_gradient(f1);
C=abs(Sx)+abs(Sy);

% Modify C to enforce screen edge node selections
NXnodes=sqrt(N)/2; NYnodes=NXnodes*3/4;
horizedgelocs=round(linspace(1,NX,round(NXnodes)));
vertedgelocs=round(linspace(1,NY,round(NYnodes)));
C(1,horizedgelocs)=inf;
C(end,horizedgelocs)=inf;
C(1,vertedgelocs)=inf;
C(end,vertedgelocs)=inf;

% Precompute motion compensated prediction
f2mcp=mcp(f1,mu,mv);

while(length(xn)<N)
% index unmarked pixels
unmarked=find(marked==0);

% Compute the DFD
DFD=f2mcp-f2;
DFDunmarked=DFD(unmarked);
DFDunmarkedenergy = DFDunmarked.*DFDunmarked;
```

```
DFDenergy=DFD.*DFD;
DFDavg = mean(DFDunmarkedenergy);

% Find max-C point that's not too close to someone else.
indlegal=find(dist2tonearest>=mindist2);
if(isempty(indlegal))
break; % we're done, no more nodes to add
   end;

[maxC,indmaxC]=max(C(indlegal));
indchoose=indlegal(min(indmaxC));

% Add it to the node list
xn=[xn;xg(indchoose)]; yn=[yn;yg(indchoose)];

% Grow a circle around it till encircled DFDenergy exceeds DFDavg
incircle=indchoose; radiustimes2=0; radiussqr=0;
distsqr=(xg-xn(end)).^2+(yg-yn(end)).^2;
while(sum(DFDenergy(incircle))<=DFDavg)
radiussqr=radiussqr+radiustimes2+1;
radiustimes2 = radiustimes2+2;
incircle=find(distsqr<=radiussqr);
end;
marked(incircle)=1;
dist2tonearest=min(dist2tonearest,(xg-xn(end)).^2+(yg-yn(end)).^2);
end;

figure(3);clf;hold off;
image(double(marked)*255);

figure(4);clf;hold off;
imagesc(sqrt(dist2tonearest));

figure(5);clf;hold off;
imagesc(C);
```