Applied Computer Vision

David Vernon
Carnegie Mellon University Africa

vernon@cmu.edu
www.vernon.eu

Lecture 22

Video Image Processing

Object Tracking I:

Exhaustive Search, Mean Shift, Optical Flow

Tracking

- Used in video surveillance, sports video analysis, vehicle guidance systems, etc.

- A hard task because objects
  - may be undergoing complex motion
  - may change shape
  - may be occluded
  - may change appearance due to lighting/weather
  - may physically change appearance

- Approaches considered:
  - Exhaustive search
  - Mean Shift
  - Dense optical flow
  - Feature-based optical flow

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

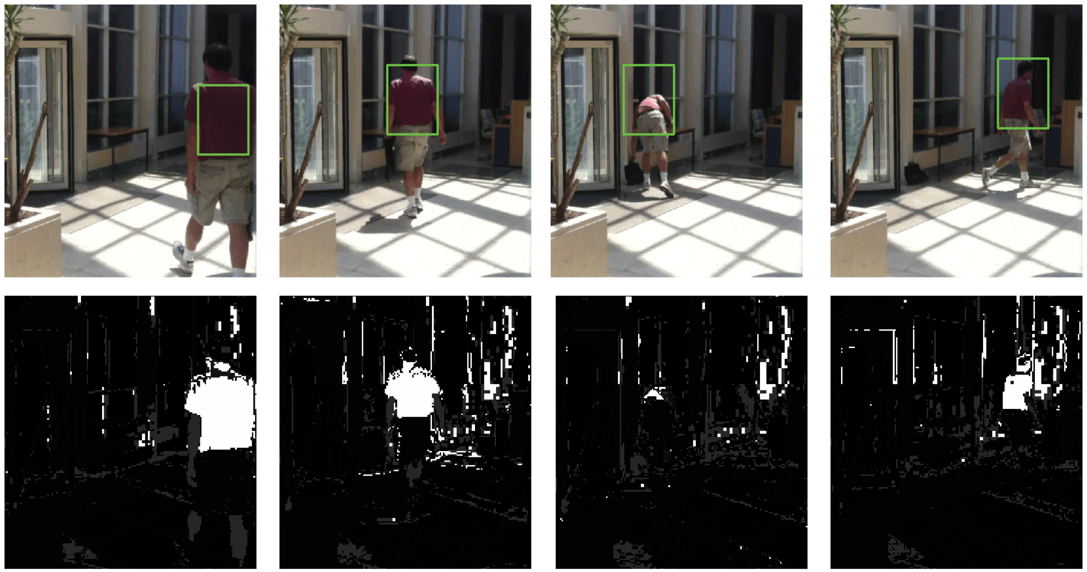
Exhaustive Search

- Extract object to be tracked from frame
- Compare in all possible positions in future frame(s)
  - Use a similarity metric, e.g. normalised cross correlation
  - Pick the best match
- Need extra degrees of freedom for scale and orientation
- May fail if object motion is too complex
- Template matching and chamfer matching support this type of tracking

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Mean Shift

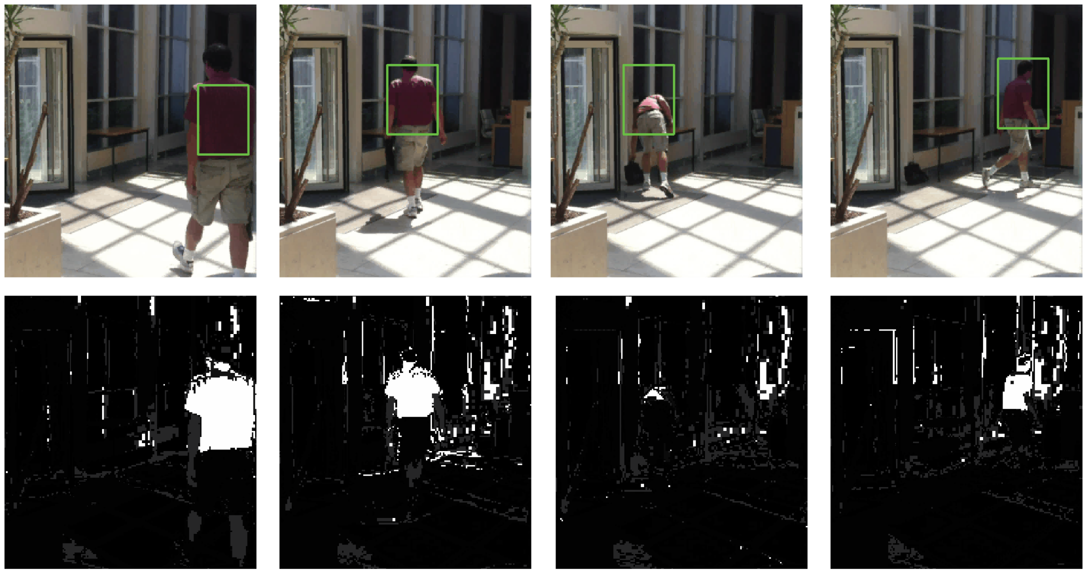
- Back-projects a histogram of the object into the current frame
- Searches for a region of the same size within the back projection looking for the highest (weighted) sum
- Uses hill climbing (gradient ascent) to iteratively look for the (local) maximum



Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

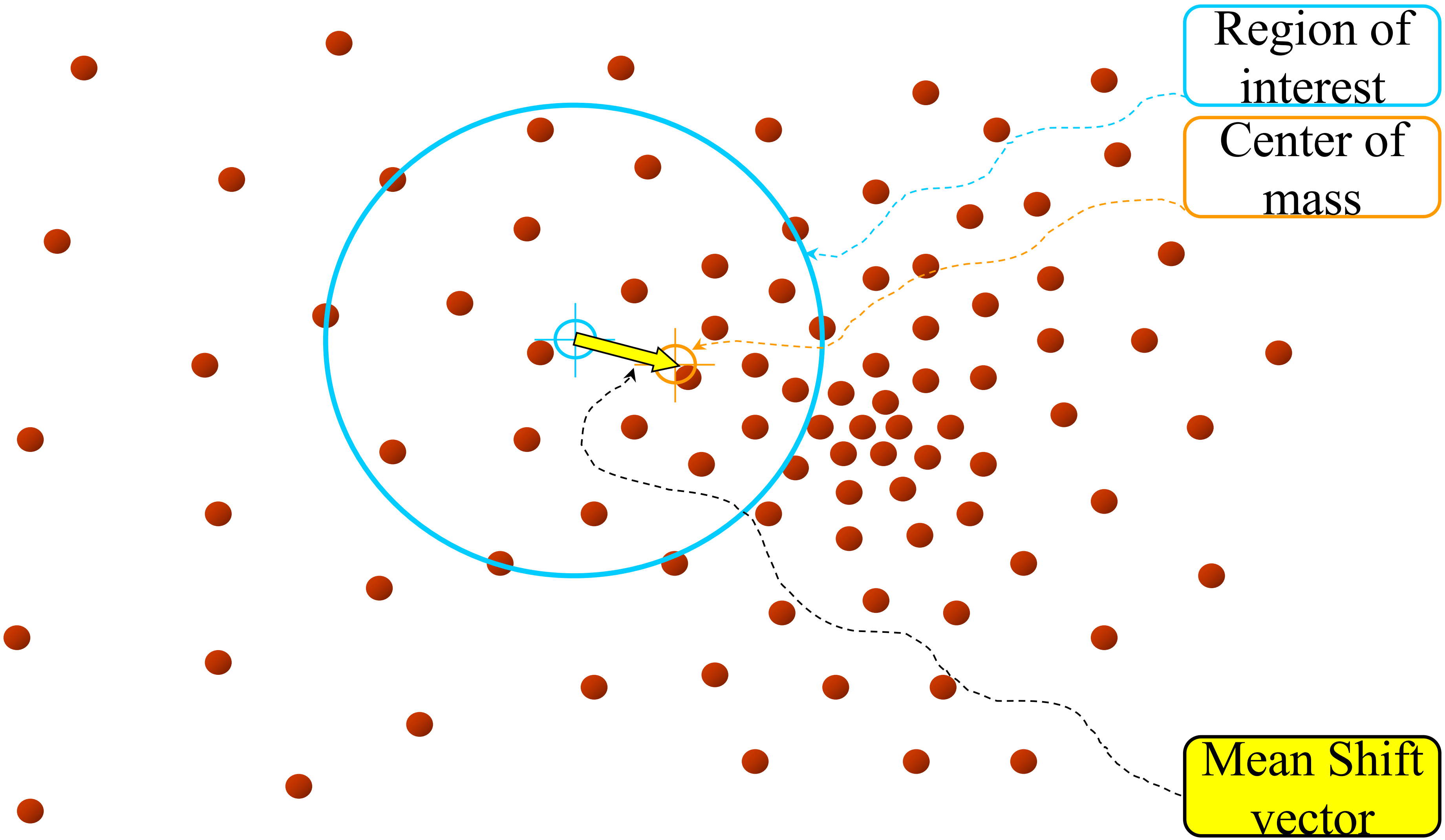
Mean Shift

- Requires a target image
- Requires initial estimate of the target location



Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

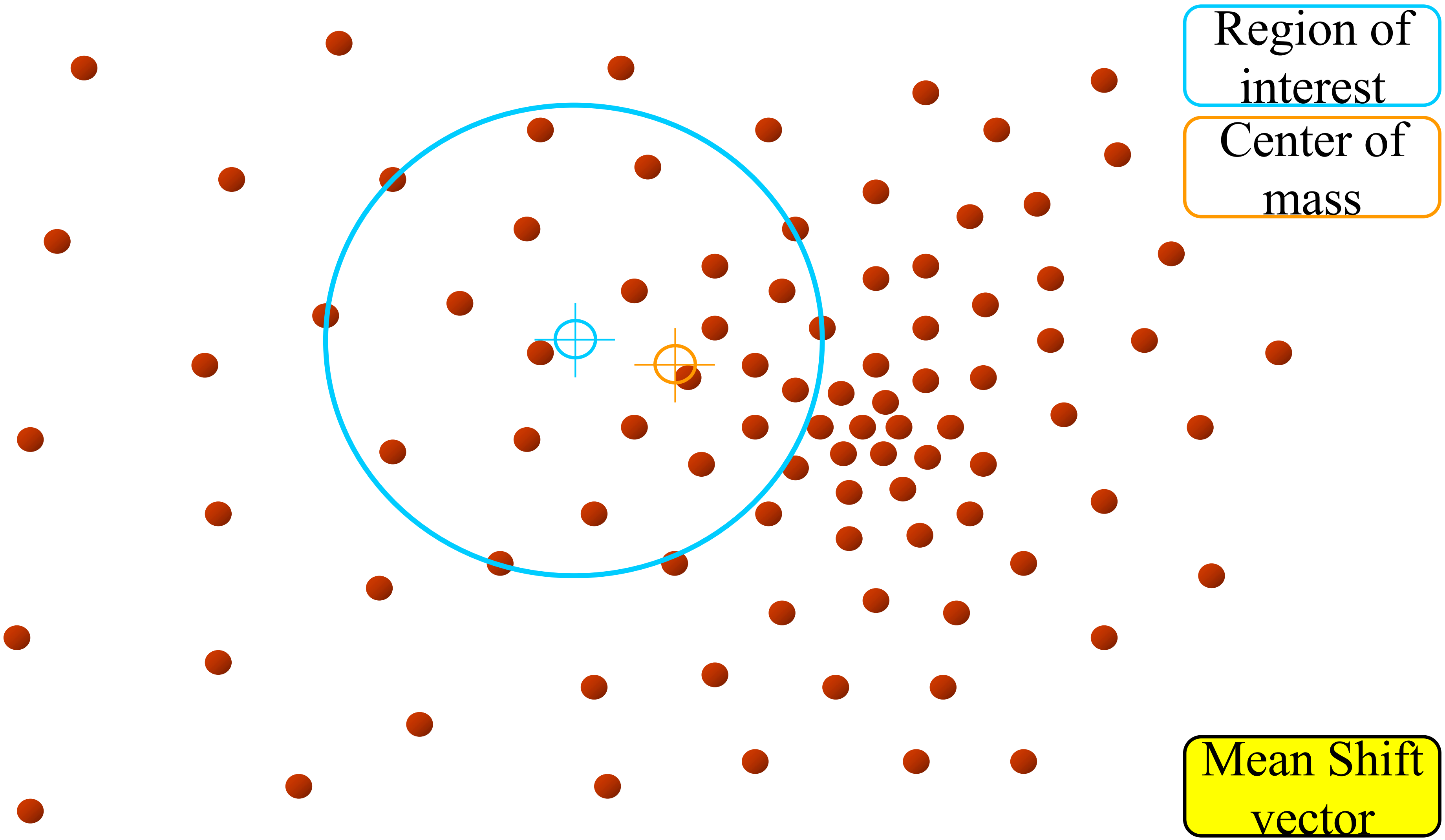
Mean Shift



Slide by Y. Ukrainitz & B. Sarel

Credit: Markus Vincze, Technische Universität Wien

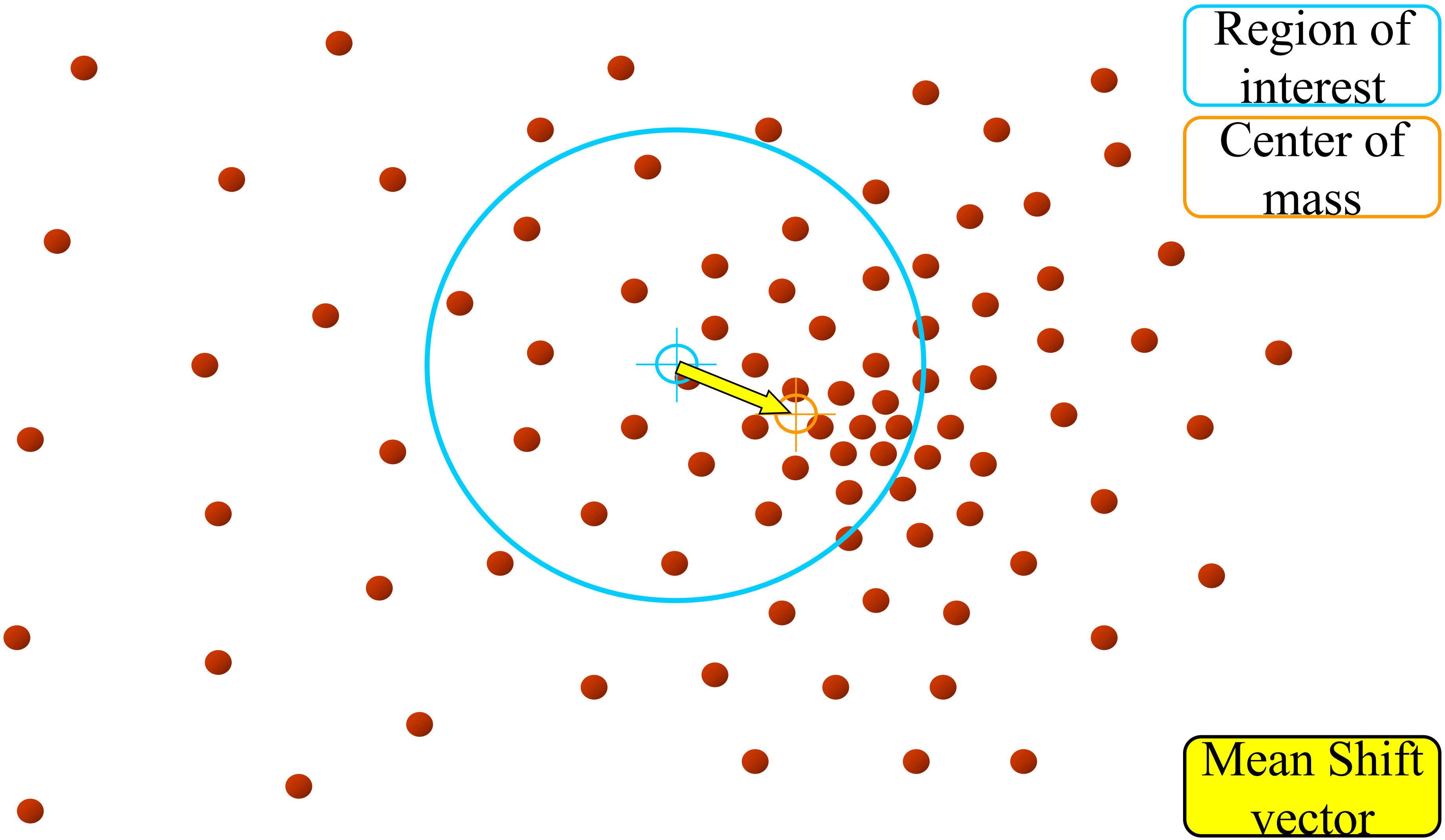
Mean Shift



Slide by Y. Ukrainitz & B. Sarel

Credit: Markus Vincze, Technische Universität Wien

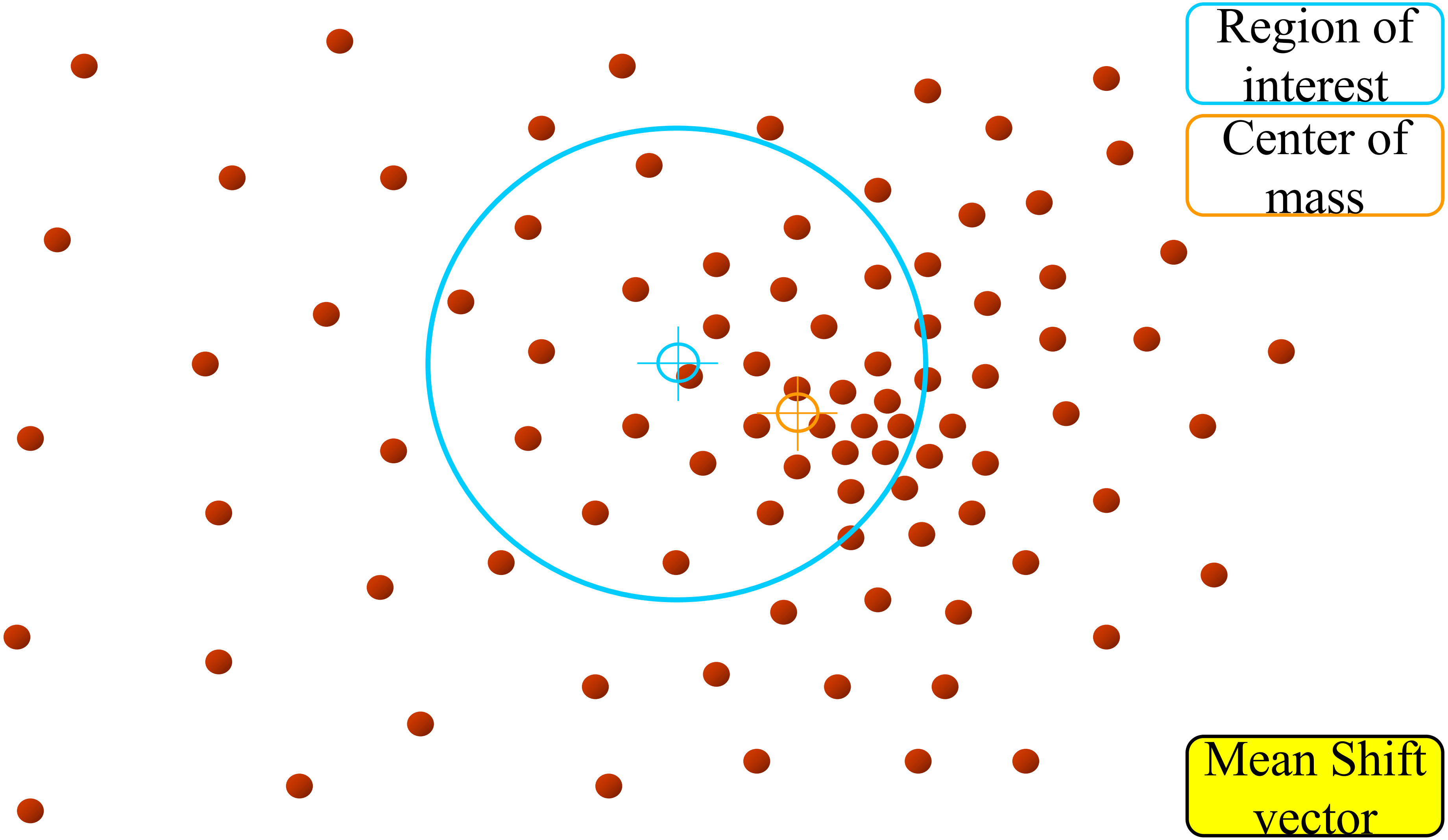
Mean Shift



Slide by Y. Ukrainitz & B. Sarel

Credit: Markus Vincze, Technische Universität Wien

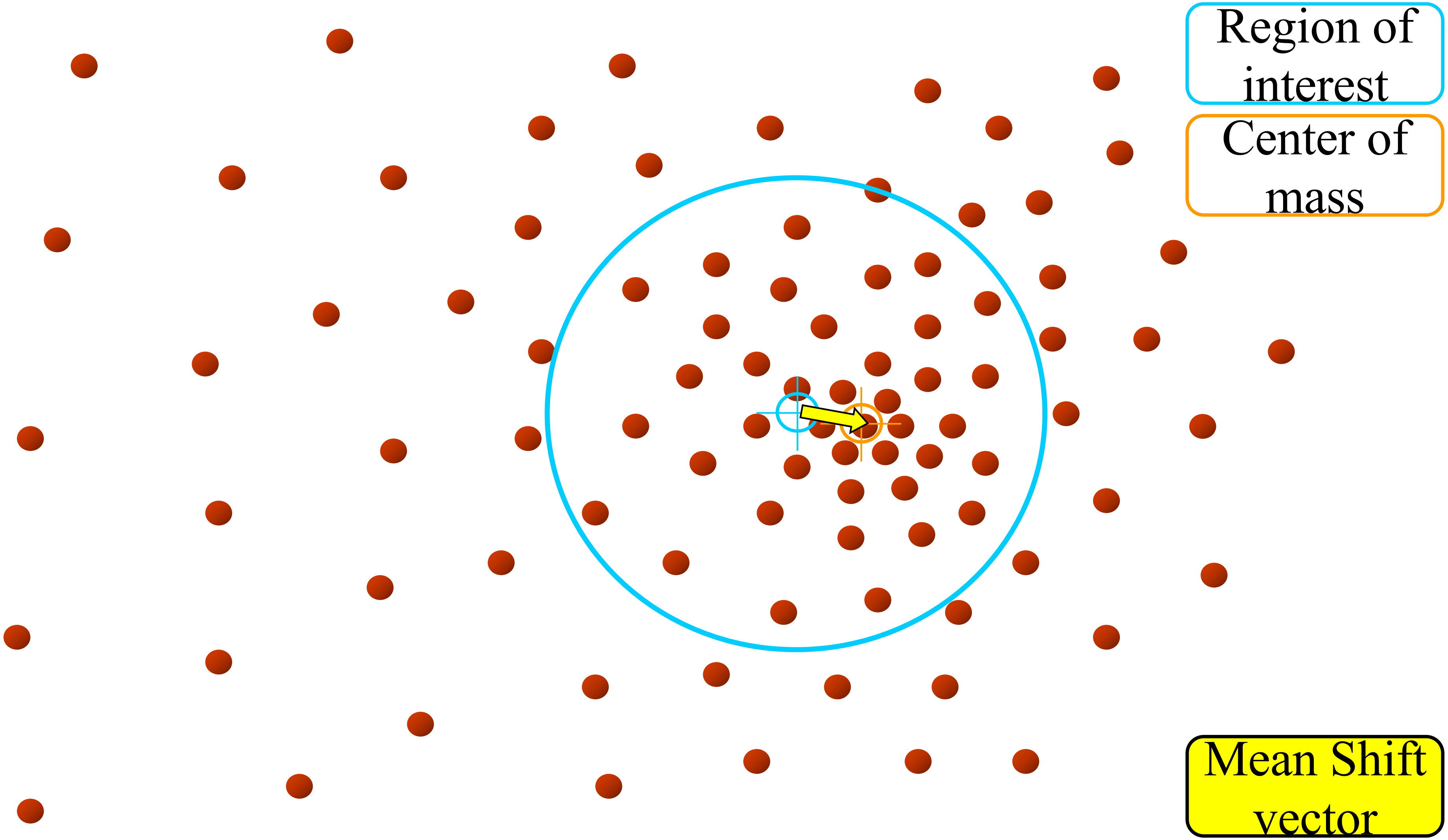
Mean Shift



Slide by Y. Ukrainitz & B. Sarel

Credit: Markus Vincze, Technische Universität Wien

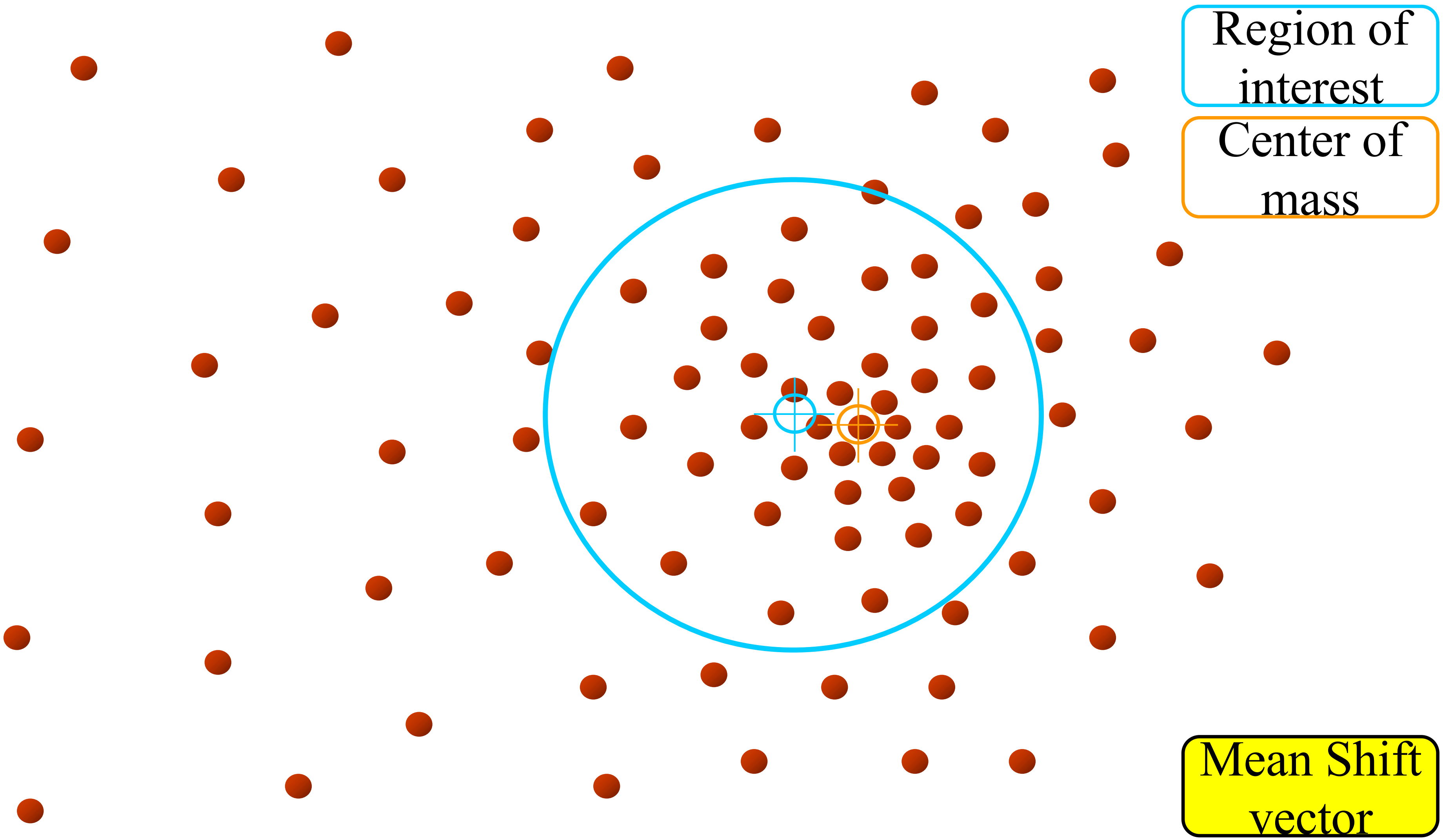
Mean Shift



Slide by Y. Ukrainitz & B. Sarel

Credit: Markus Vincze, Technische Universität Wien

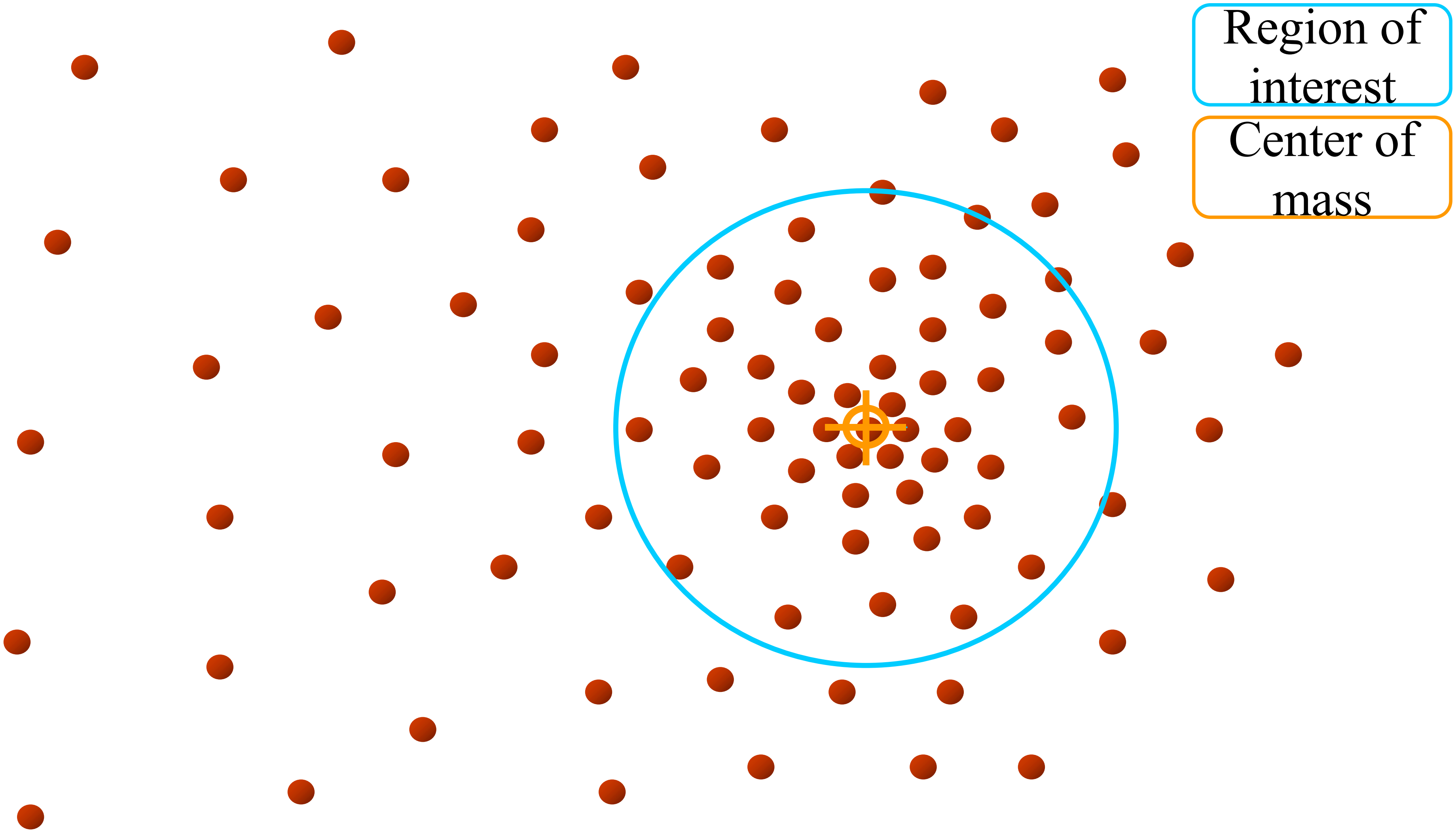
Mean Shift



Slide by Y. Ukrainitz & B. Sarel

Credit: Markus Vincze, Technische Universität Wien

Mean Shift



Slide by Y. Ukrainitz & B. Sarel

Credit: Markus Vincze, Technische Universität Wien

Mean Shift - Algorithm

Searches an end point (*modes*) for a given set of points

1. Select kernel

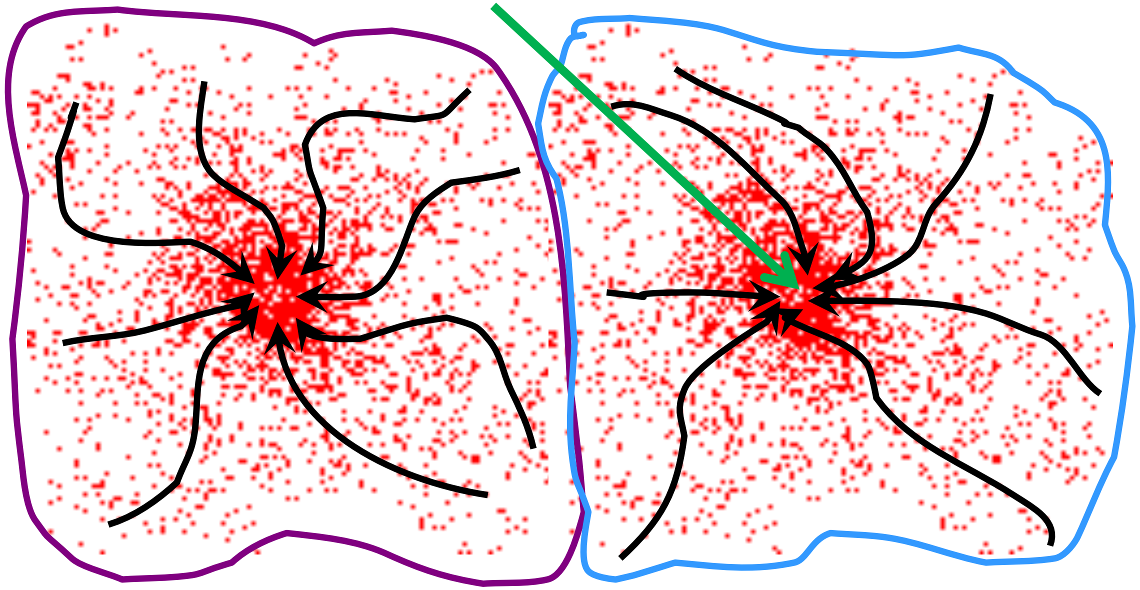
2. For each point:
   a) Set a window around this point
   b) Calculate the mean of the data in this window
   c) Shift the center of the window to new mean
   d) Repeat steps b) and c) until convergence in end point (no more shifts)

3. Assign point to a cluster that leads to the same mode (end point)

Credit: Markus Vincze, Technische Universität Wien

Density as Attraction Basin

- Attraction basin: region, in which all paths lead to the same end point (mode)
- Cluster: all data points in attraction basin of a mode (end point)



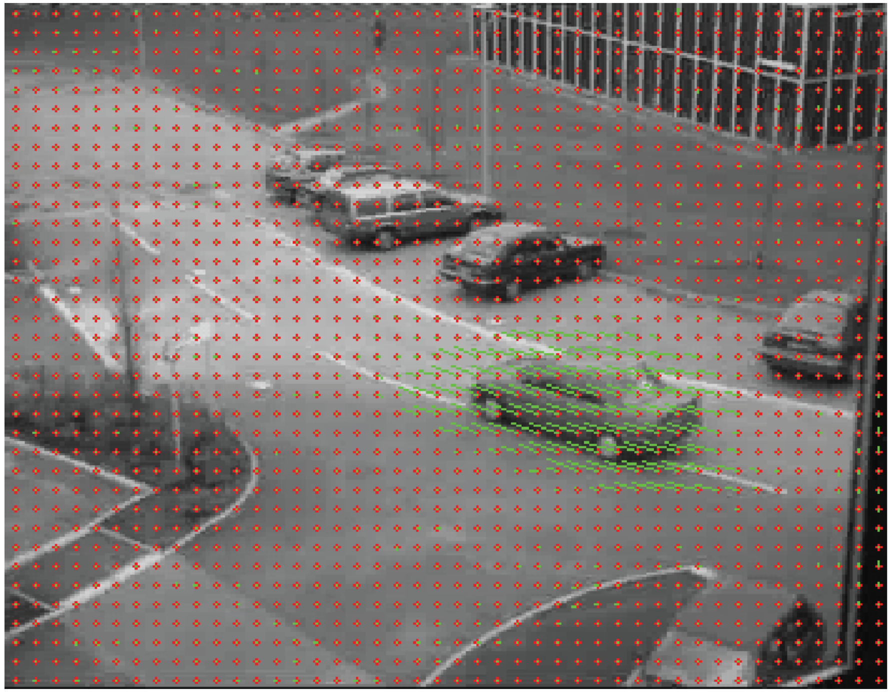
Slide by Y. Ukrainitz & B. Sarel

Credit: Markus Vincze, Technische Universität Wien

Dense Optical Flow

Compute a motion field (known as optical flow) for the entire image

- Direction & Magnitude



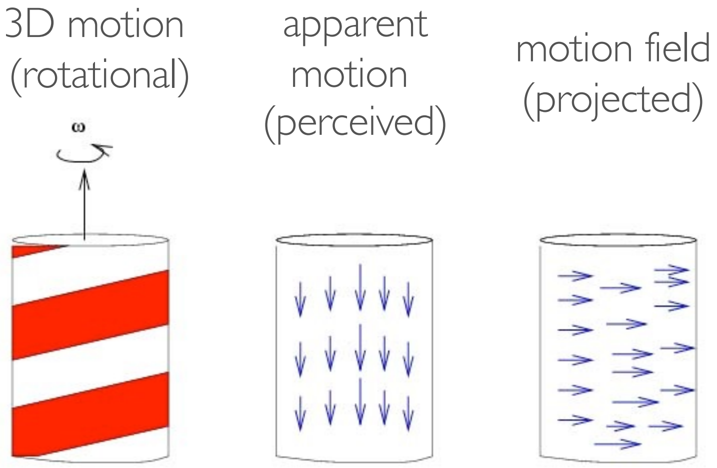
© Reproduced by permission of Dr. James Ferryman, University of Reading

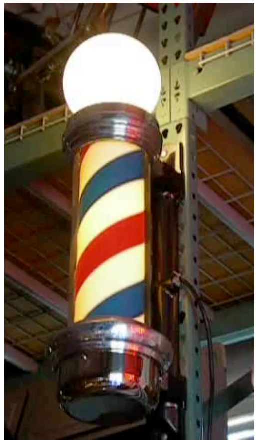
Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Dense Optical Flow

- We can estimate the motion field from images, but what we estimate will be related to the apparent motion

- What is the motion field of an object moving in a dark room? or of a uniform object on a similar background?



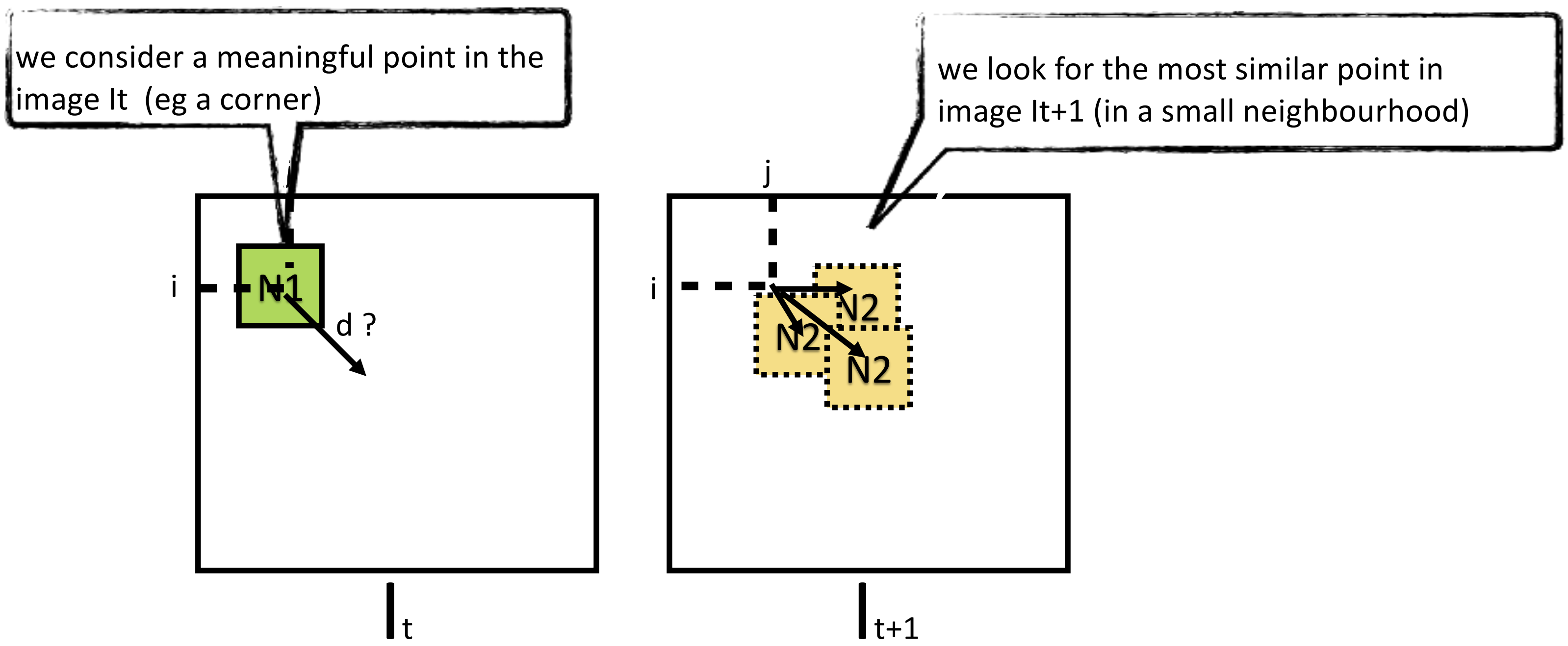


Credit: Francesca Odone, University of Genova

Dense Optical Flow

- Two different images of the same scene, this time acquired by the same camera at adjacent temporal instants

- Prior: in this case we may assume the displacement *d* to be very small



Credit: Francesca Odone, University of Genova

Dense Optical Flow

- We start from an assumption on the image brightness constancy

- A classical derivative-based algorithm: Lucas Kanade

$$\frac{dI}{dt} = 0$$

total image brightness is constant

$$\frac{d(I(x,y,t))}{dt} = \frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = 0$$

y component of motion (velocity)

x component of motion (velocity)

Total (or full) derivative

$$(\nabla I)^{\top}\mathbf{u} + I_t = 0$$

(see https://en.wikipedia.org/wiki/Total_derivative)

visual motion (i.e. optical flow)

Credit: Francesca Odone, University of Genova

Dense Optical Flow

Alternative derivation

From frame to frame, image appearance does not change (i.e. is constant)

$$f_t(i,j) = f_{t+\Delta t}(i + \Delta i, j + \Delta j)$$

If we assume the motion from frame to frame is small, we can use the approximation:

$$f_{t+\Delta t}(i + \Delta i, j + \Delta j) = f_t(i,j) + \frac{\partial f}{\partial i}\Delta i + \frac{\partial f}{\partial j}\Delta j + \frac{\partial f}{\partial t}\Delta t$$

In effect, we approximate the next frame by adding estimates of motion in both directions and in time to to the image at time t

Hence:

$$\frac{\partial f}{\partial i}\Delta i + \frac{\partial f}{\partial j}\Delta j + \frac{\partial f}{\partial t}\Delta t = 0$$

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Dense Optical Flow

Alternative derivation

Dividing across by Δt:

$$\frac{\partial f}{\partial i}\frac{\Delta i}{\Delta t} + \frac{\partial f}{\partial j}\frac{\Delta j}{\Delta t} + \frac{\partial f}{\partial t} = 0$$

Reorganizing:

$$\begin{bmatrix} \frac{\partial f}{\partial i} & \frac{\partial f}{\partial j} \end{bmatrix} \begin{bmatrix} \frac{\Delta i}{\Delta t} \\ \frac{\Delta j}{\Delta t} \end{bmatrix} = -\frac{\partial f}{\partial t}$$

$$(\nabla I)^\top \mathbf{u} + I_t = 0$$

Dense Optical Flow

- The optical flow is a vector field subject to the constraint

$$(\nabla I)^\top \mathbf{u} + I_t = 0$$

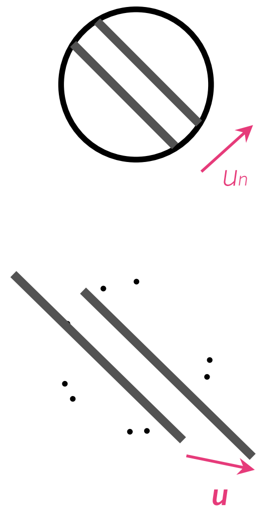
- Notice that **one** constraint is not enough to compute the optical flow (**2** unknowns)

Credit: Francesca Odone, University of Genova

Dense Optical Flow

- The image brightness constancy equation allows us to determine the optical flow component parallel to the spatial image gradient

- Analytically $$u_n = \frac{(\nabla I)^\top \mathbf{u}}{||\nabla I||} = \frac{-I_t}{||\nabla I||}$$



Credit: Francesca Odone, University of Genova

Dense Optical Flow

- Many algorithms start from the idea of adding constraints to the underdetermined system obtained by the brightness constancy equation

- We will see a simple way of doing so: the Lucas-Kanade algorithm

  – Assumption: **u** is constant in a small neighbourhood of a point

Credit: Francesca Odone, University of Genova

Dense Optical Flow

- The assumption allows us to obtain a system of equations with one equation for each point in the neighbourhood

$$(\nabla I(\mathbf{x}_i, t))^\top \mathbf{u} + I_t(\mathbf{x}_i, t) = 0 \quad \mathbf{x}_i \in N$$

- We then obtain a linear system $A\mathbf{u}=\mathbf{b}$ with

$$A = \begin{bmatrix} \nabla I(\mathbf{x}_1, t)^\top \\ \nabla I(\mathbf{x}_2, t)^\top \\ \vdots \\ \nabla I(\mathbf{x}_m, t)^\top \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} -I_t(\mathbf{x}_1, t) \\ -I_t(\mathbf{x}_2, t) \\ \vdots \\ -I_t(\mathbf{x}_m, t) \end{bmatrix}$$

m elements in the *N* neighbourhood

Credit: Francesca Odone, University of Genova

Dense Optical Flow

- The linear system may be solved with the pseudo-inverse

$$\mathbf{u} = A^{\dagger}\mathbf{b} \quad \text{with} \quad A^{\dagger} = (A^{\top}A)^{-1}A^{\top}$$

- Notice that the inversion of the matrix will be ill-posed if the matrix is not full rank (i.e. all the equations are not linearly independent)

- The matrix is full rank in the proximity of corners (points who do not suffer from the aperture problem)

  This is why Lucas Kanade is often implemented as a *sparse* algorithm (after a corner detection stage)

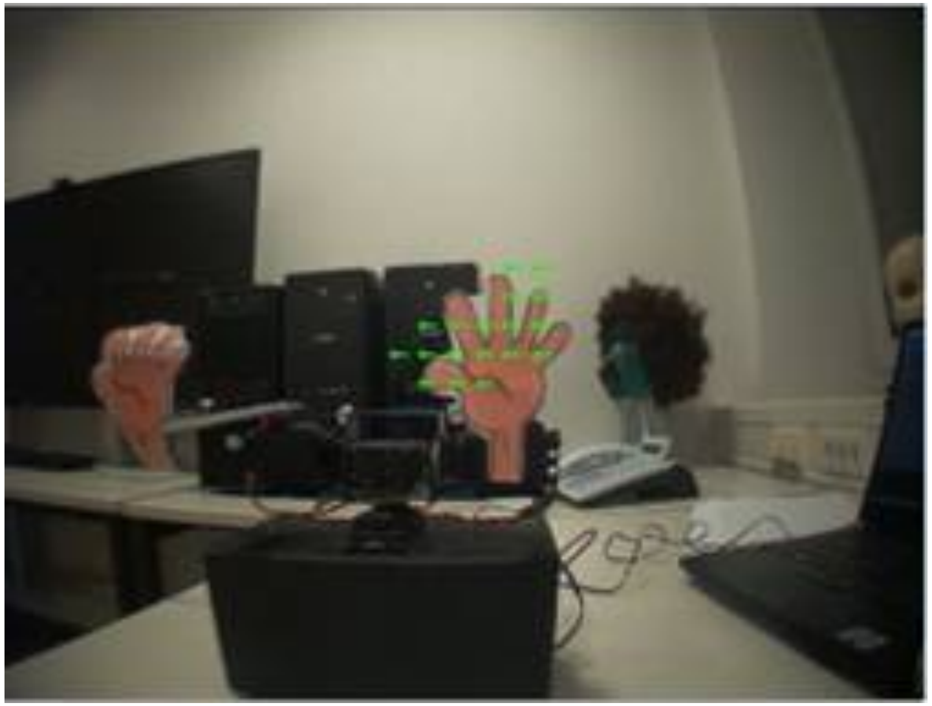
Credit: Francesca Odone, University of Genova

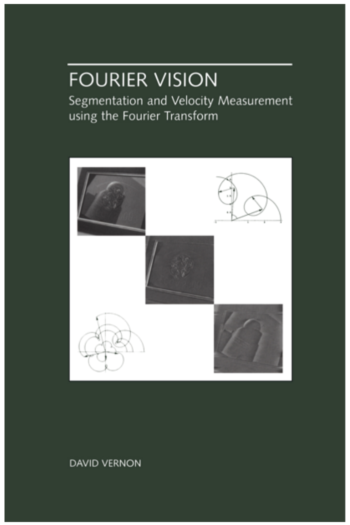
Feature-based Optical Flow

- We cannot accurately compute optical flow for constant regions or along edges
- Often better to compute optical flow just for features (e.g. Lucas Kanade feature tracker)

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Dense Optical Flow





D. Vernon, "Computation of Instantaneous Optical Flow using the Phase of Fourier Components", Image and Vision Computing, Vol. 17, No. 3-4, pp. 189-198, 1999.

D. Vernon, Fourier Vision, The Springer International Series in Engineering and Computer Science, Vol. 623 (ISBN: 978-0-7923-7413-8)

Dense Optical Flow

A function $f(x, y)$ translating with velocity (v_x, v_y) can be written

$$f(x - v_x\delta t, y - v_y\delta t)$$

The shift property of the Fourier transform states:

$$\mathcal{F}\left(f(x - v_x\delta t, y - v_y\delta t)\right) = |\mathsf{F}(\omega_x, \omega_y)|\, e^{i\phi(\omega_x,\omega_y)} e^{-i(\omega_x v_x \delta t + \omega_y v_y \delta t)}$$

Thus, a spatial shift of $(v_x\delta t, v_y\delta t)$ of an image,

i.e. $f(x, y)$ shifted to $f(x - v_x\delta t, y - v_y\delta t)$

only produces a change in the phase of the Fourier components:

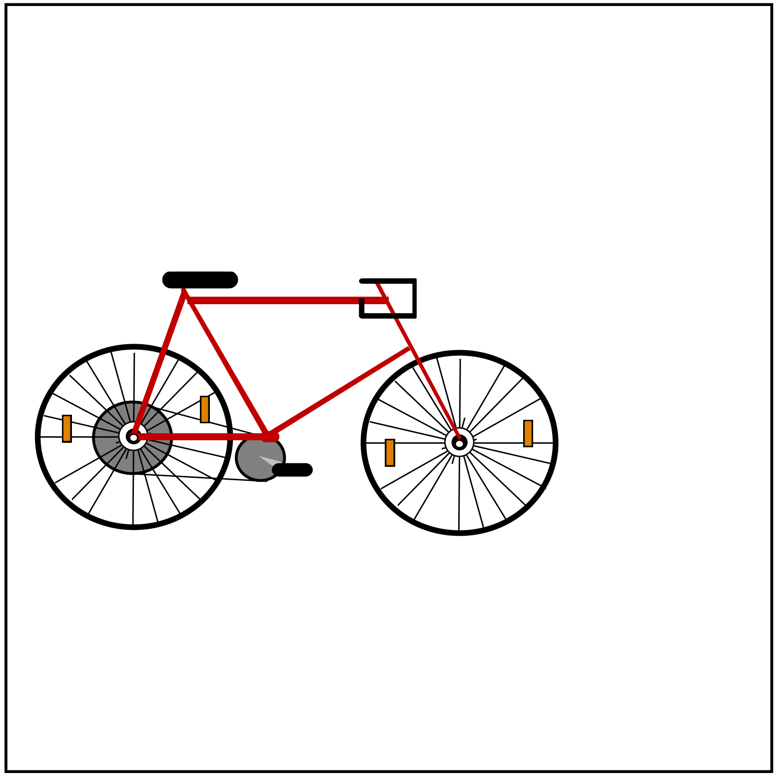
$$e^{-i(\omega_x v_x \delta t + \omega_y v_y \delta t)}$$

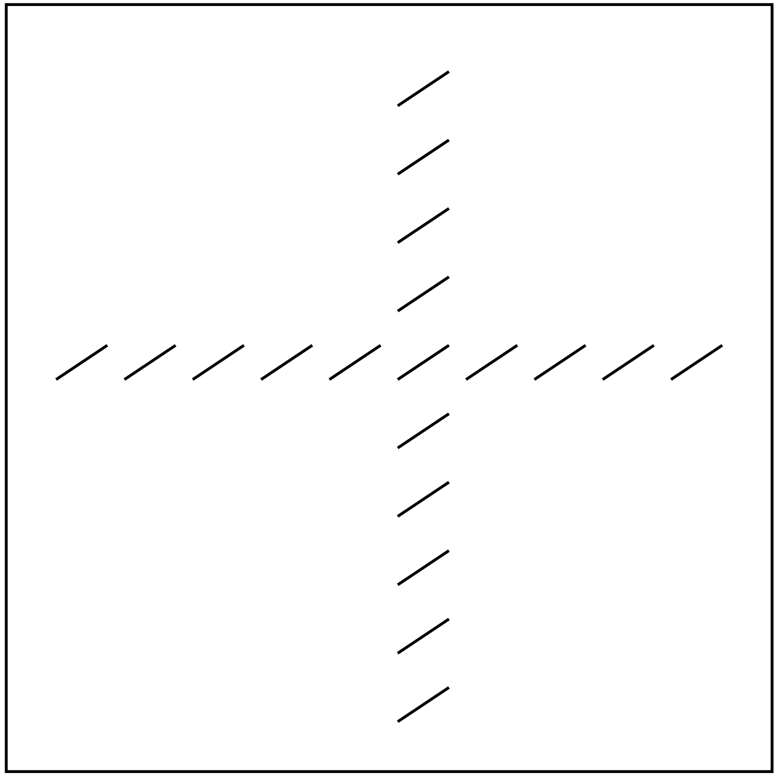
Dense Optical Flow

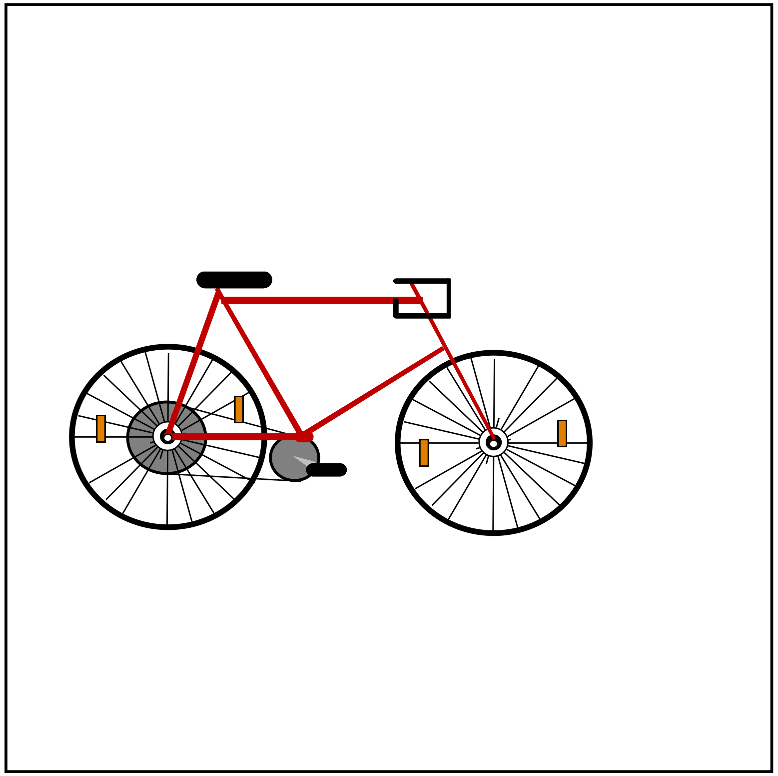
The change in phase corresponds to a rotation of the Fourier phasors

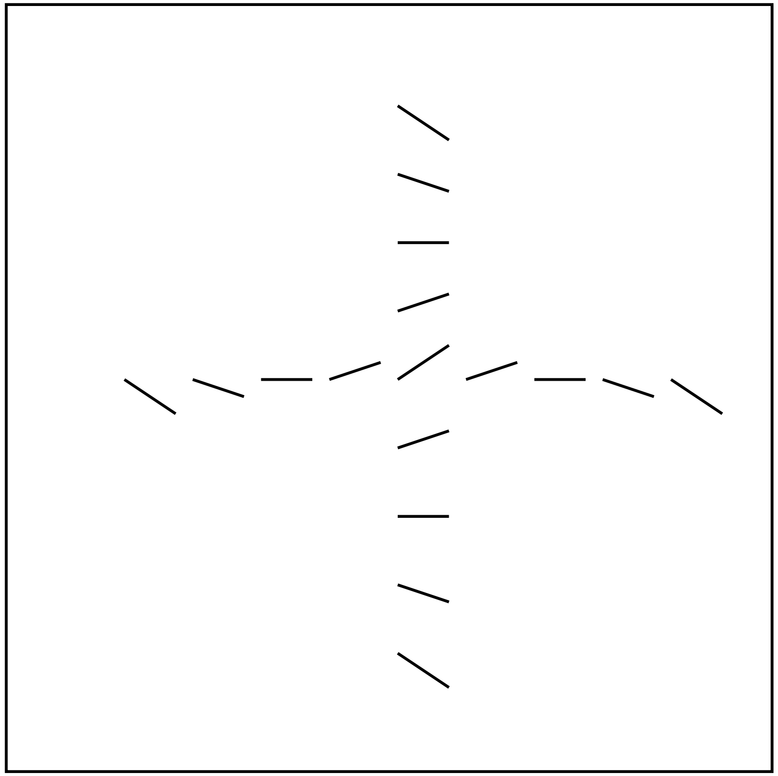
The angle of rotation is determined by both

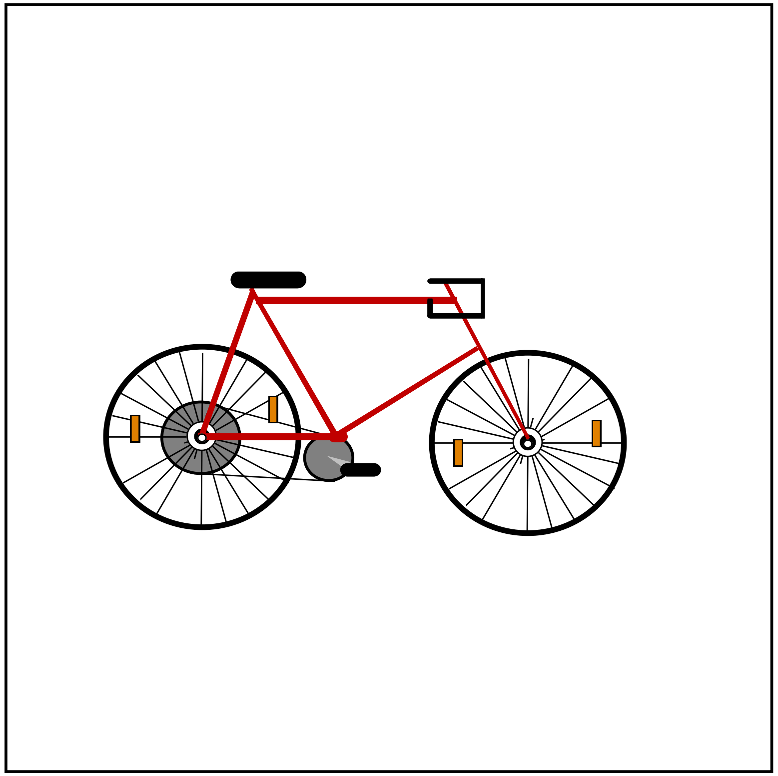
- the velocity
- the spatial frequency

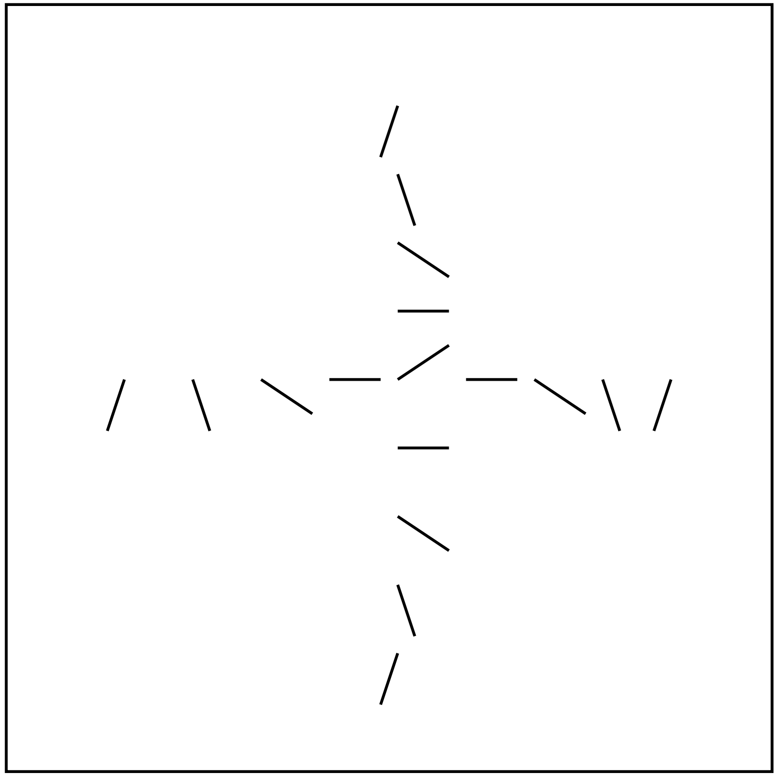


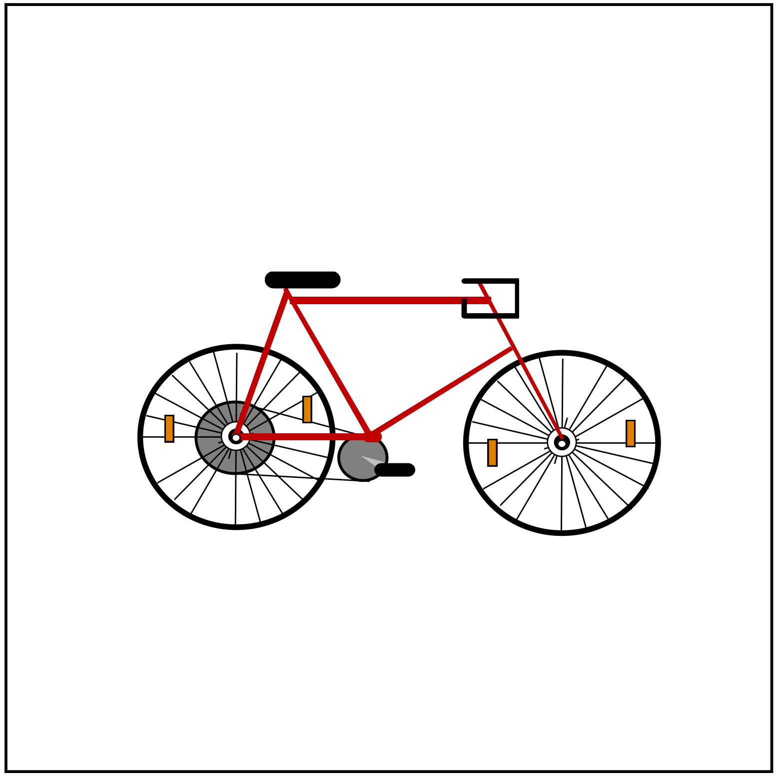


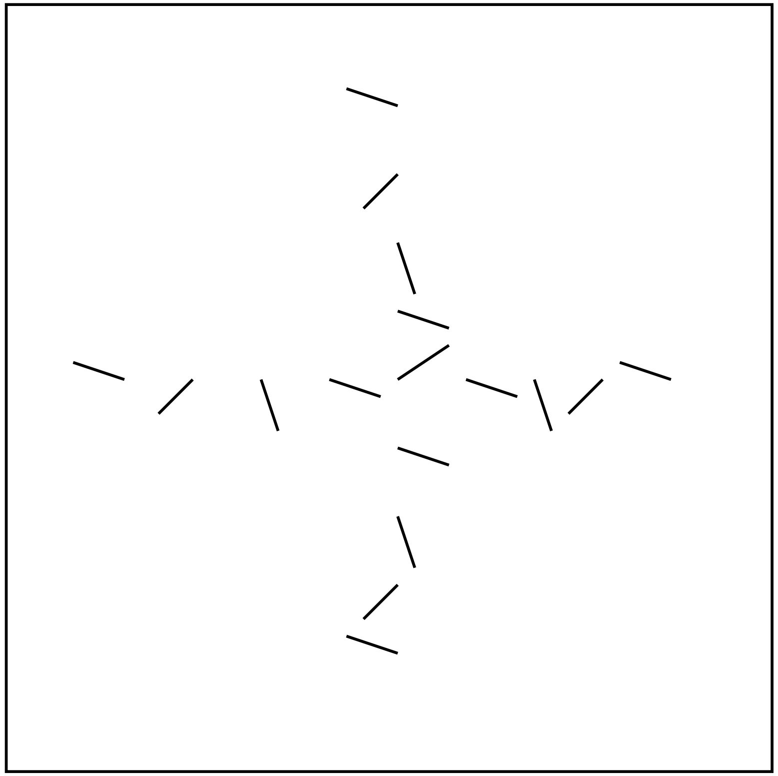












Dense Optical Flow

- To estimate the velocity of an image translating with constant velocity we just need to identify the phase shift $e^{-i(\omega_x v_x \delta t + \omega_y v_y \delta t)}$

Rotation

Phase of image at time t

- Noting that

$$
\begin{aligned}
e^{i\phi_{t+\delta t}(\omega_x,\omega_y)} &= e^{-i(\omega_x v_x \delta t + \omega_y v_y \delta t)} e^{i\phi_t(\omega_x,\omega_y)} \\
&= e^{i(\phi_t(\omega_x,\omega_y) - (\omega_x v_x \delta t + \omega_y v_y \delta t))}
\end{aligned}
$$

Phase of shifted image at time $t + \delta t$

- Hence:

$$
\phi_{t+\delta t}(\omega_x, \omega_y) = \phi_t(\omega_x, \omega_y) - (\omega_x v_x \delta t + \omega_y v_y \delta t)
$$

Dense Optical Flow

- Rearranging:

$$v_y = \frac{1}{\omega_y \delta t}\left(\phi_t(\omega_x, \omega_y) - \phi_{t+\delta t}(\omega_x, \omega_y) - \omega_x v_x \delta t\right)$$

- Treat this as a Hough transform defined on v_x and v_y
  - For each spatial frequency (ω_x, ω_y)
    - Determine phase $\phi_t(\omega_x, \omega_y)$ for image acquired at time t
    - Determine phase $\phi_{t+\delta t}(\omega_x, \omega_y)$ for image acquired at time $t + \delta t$
    - For all values of v_x, compute v_y [knowing $\omega_x, \omega_y, \phi_t(\omega_x, \omega_y), \phi_{t+\delta t}(\omega_x, \omega_y)$]
      - Increment the Hough accumulator for that v_x and v_y
  - Local maxima in this Hough correspond to the required velocity

Dense Optical Flow

- Note that

$$v_y = \frac{1}{\omega_y \delta t}\left(\phi_t(\omega_x, \omega_y) - \phi_{t+\delta t}(\omega_x, \omega_y) - \omega_x v_x \delta t\right)$$

is degenerate when $\omega_y = 0$

- In that case, use an alternative re-arrangement

$$v_x = \frac{1}{\omega_x \delta t}\left(\phi_t(\omega_x, \omega_y) - \phi_{t+\delta t}(\omega_x, \omega_y)\right)$$

```
/* compute optical flow at coordinates i and j in images f1(i,j) and f2(i,j) */
/* where i and j are the coordinates of the centre of a 64 x 64 pixel region */
/* i and j effectively sample the image with a sampling period sp            */
/* (sp = 10 pixels in the results presented in this chapter)                 */
/* The dimensions of f(i,j) are assumed to be given by variables d_i and d_j */

initial_i = 32; final_j = d_i - 32;
initial_j = 32; final_j = d_j - 32;

for (i = initial_i; i < final_i; i = i + sp)
   for (j = initial_j; j < final_j; j = j + sp)

      extract 64x64 pixel regions f1' and f2', centred at i,j, from f1 and f2

      apodized/window f1' and f2' by computing

         g1(x,y) = f1'(x,y) x G(x,y)  // G(x,y) is a 64x64 pixel Gaussian
         g2(x,y) = f2'(x,y) x G(x,y)  // = 0.5 at nw/8 pixels from centre
                                      // (n=1, 2, 3 and w = 64)

      compute G1(wx, wy), the Fourier transform of g1(x,y)
      compute G2(wx, wy), the Fourier transform of g2(x,y)

      compute the phases of G1 and G2:  P1(wx, wy) and P2(wx, wy)

      for (wx = inital_wx; wx < final_wx; wx = wx + 1)      // -32 < wx < 32
         for (wy = inital_k; wy < final_wy; wy = wy + 1)    // -32 < wy < 32

            compute phase difference: pd = P1(wx, wy) - P2(wx, wy);

            if (wy != 0)
               for (vx = 0; vx < 10; vx = vx + 0.1)   // vx is the x velocity
                  vy = (pd - (wx * vx)) / wy
                  Hough_accumulator[vx,vy] += 1
            else if (wx != 0)
               vx = pd / wx;
               for (vy = 0; vy < 10; vy = vy + 0.1)   // vy is the y velocity
                  Hough_accumulator[vx,vy] += 1
```

Dense Optical Flow

Apply the technique

- on a local, windowed, apodized basis
- to compute the velocity of that window from the phase change

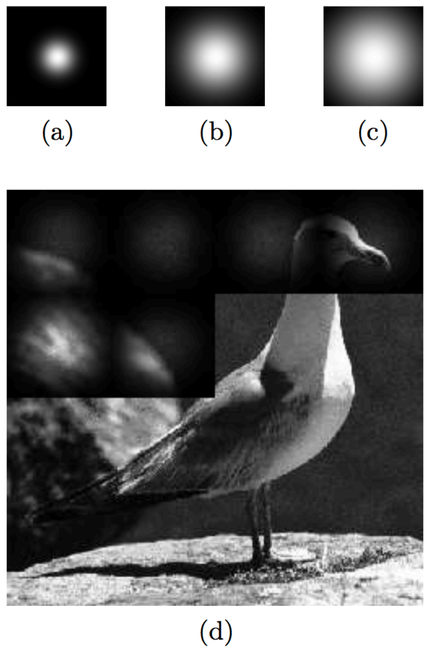
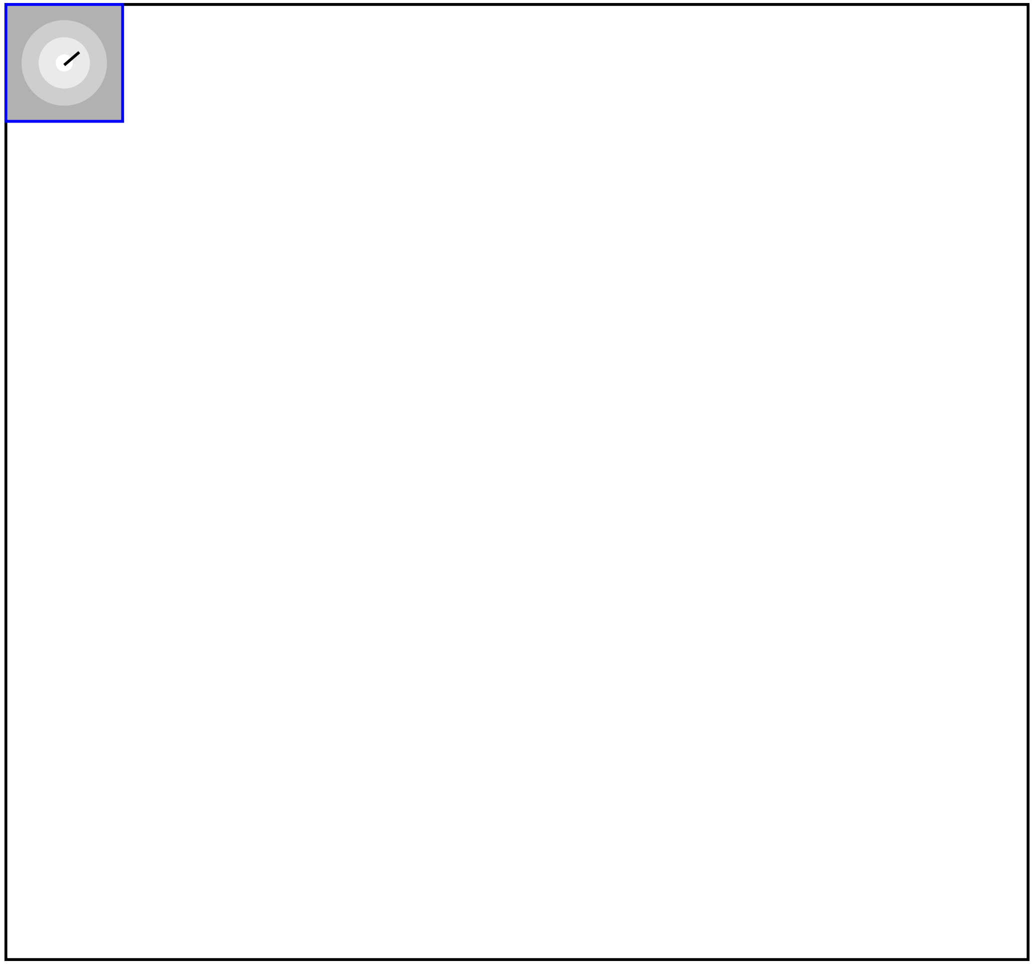
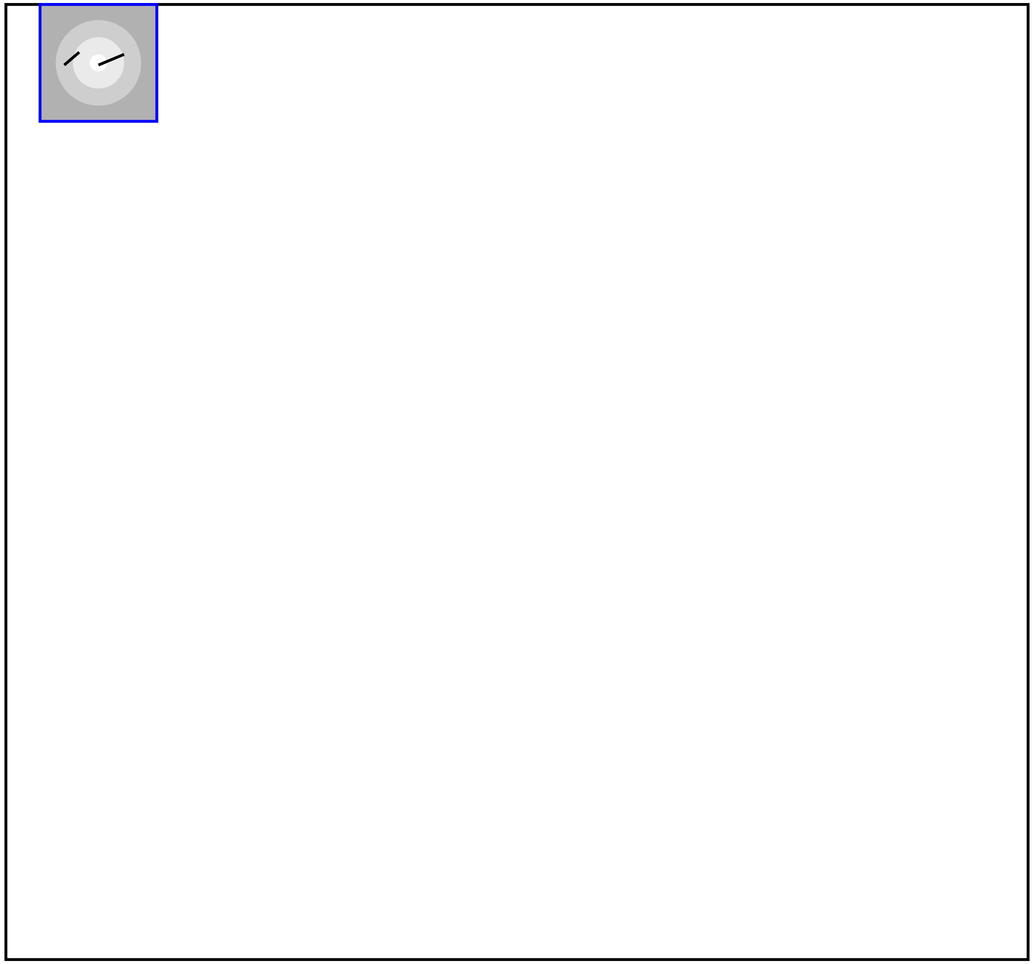


Figure 7.1. Gaussian windowing functions: (a), (b), (c) 50% weighting at $\frac{w}{8}$, $\frac{2w}{8}$, $\frac{3w}{8}$ pixels from the region centre, respectively (w is the size of the window); (d) each region in the image is multiplied by the windowing function before its Fourier transform is computed and its velocity estimated.

Dense Optical Flow

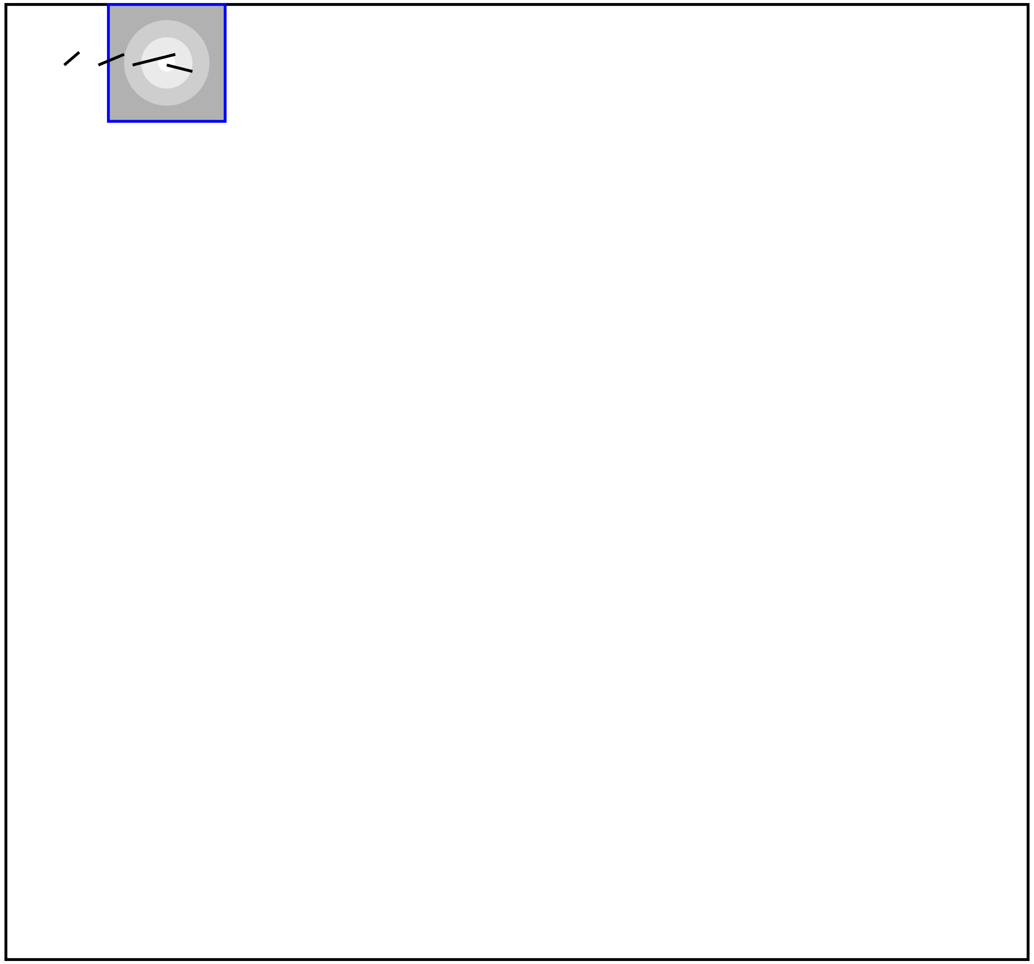


Dense Optical Flow

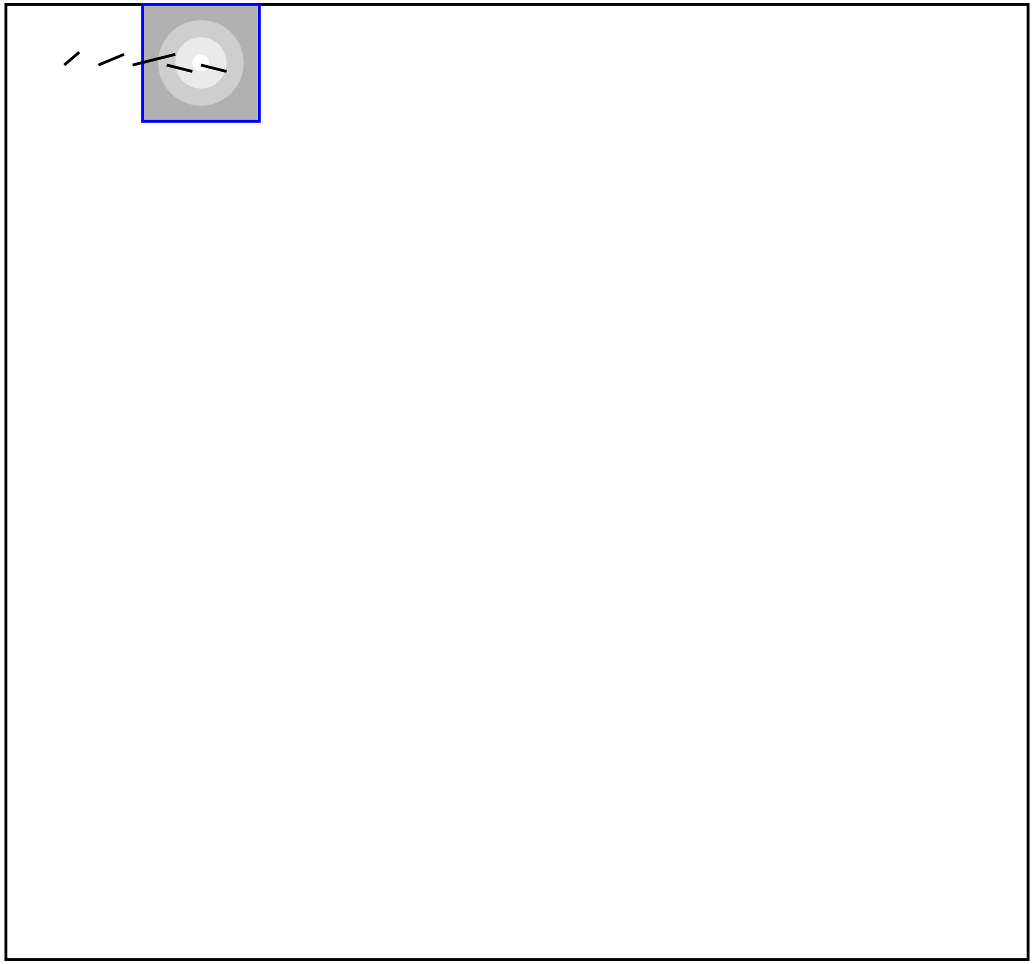


Dense Optical Flow

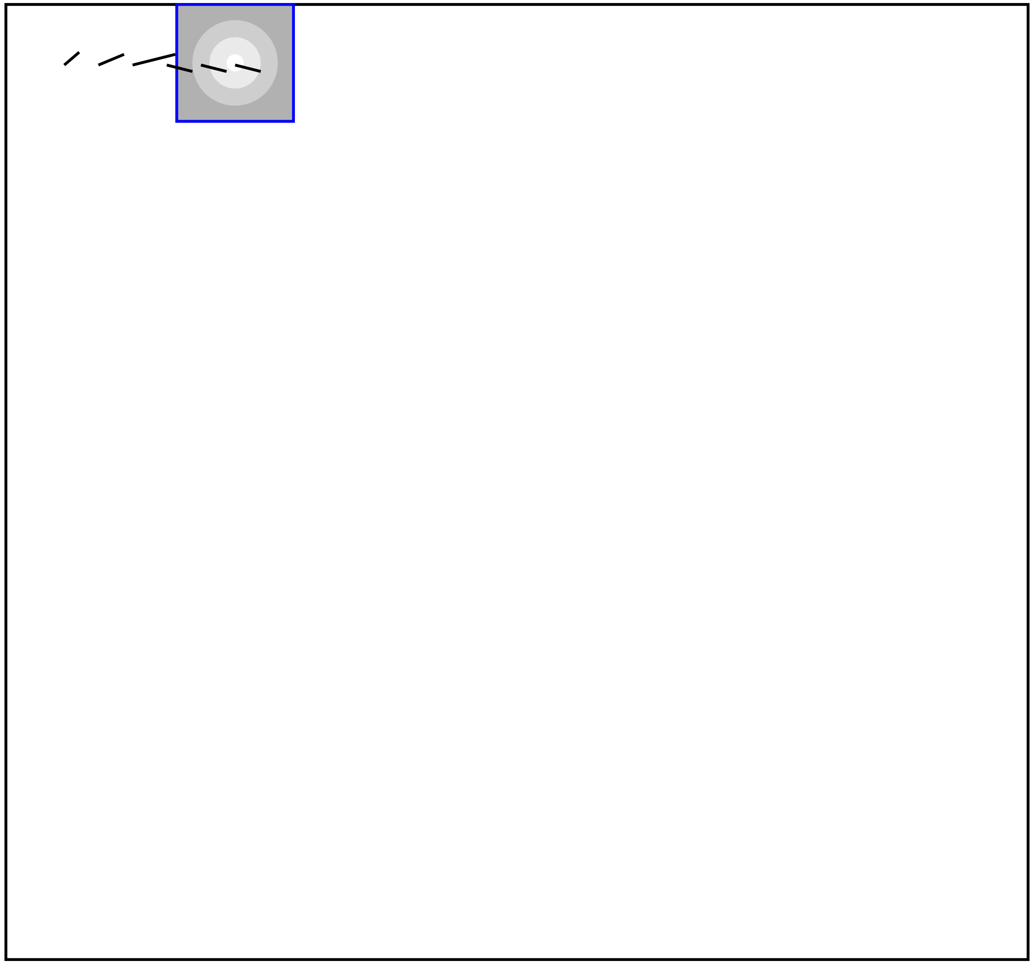
Dense Optical Flow



Dense Optical Flow



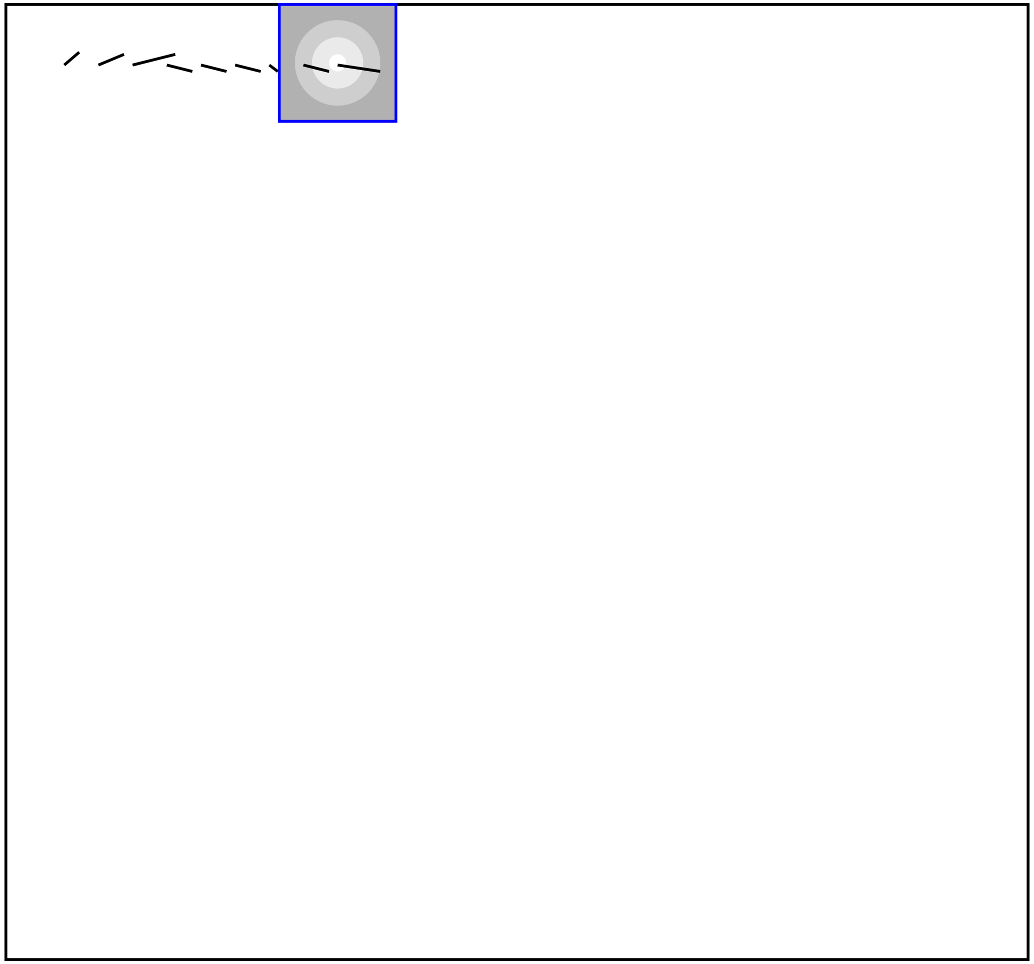
Dense Optical Flow



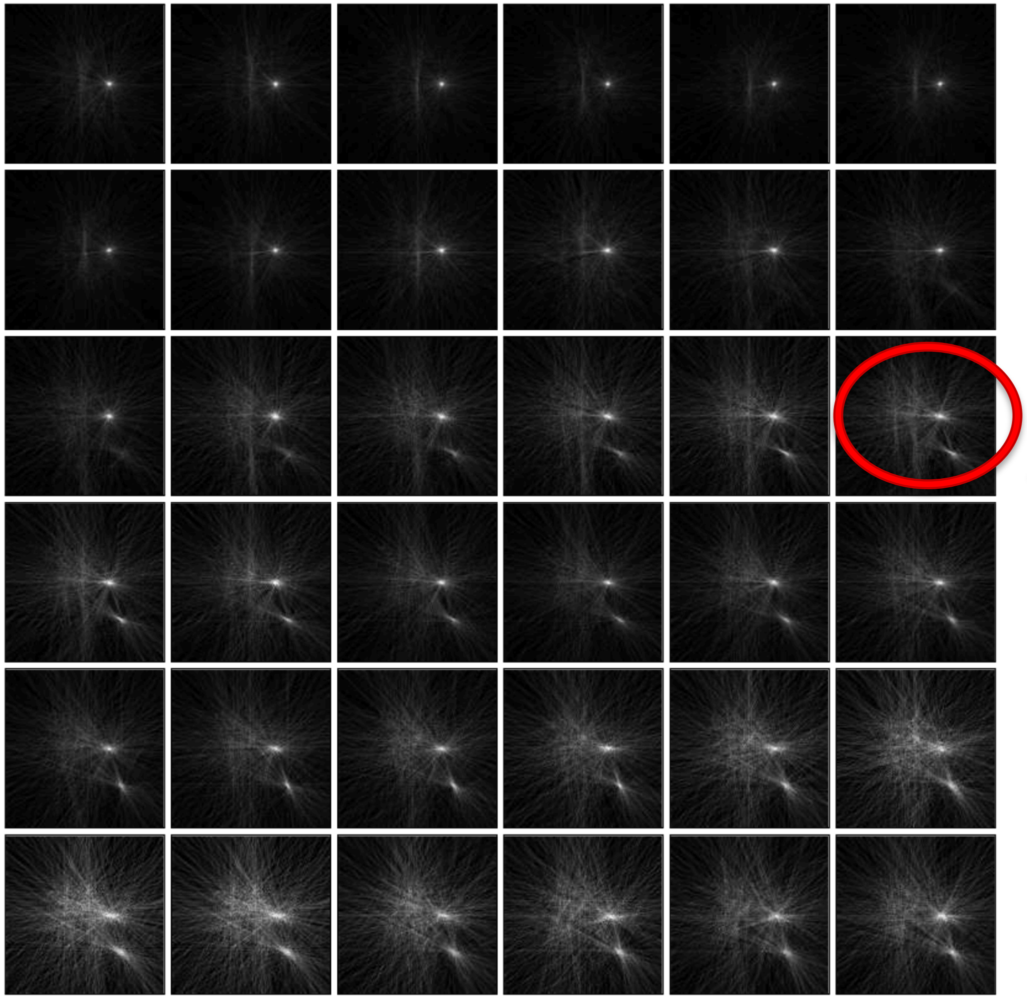
Dense Optical Flow

Dense Optical Flow

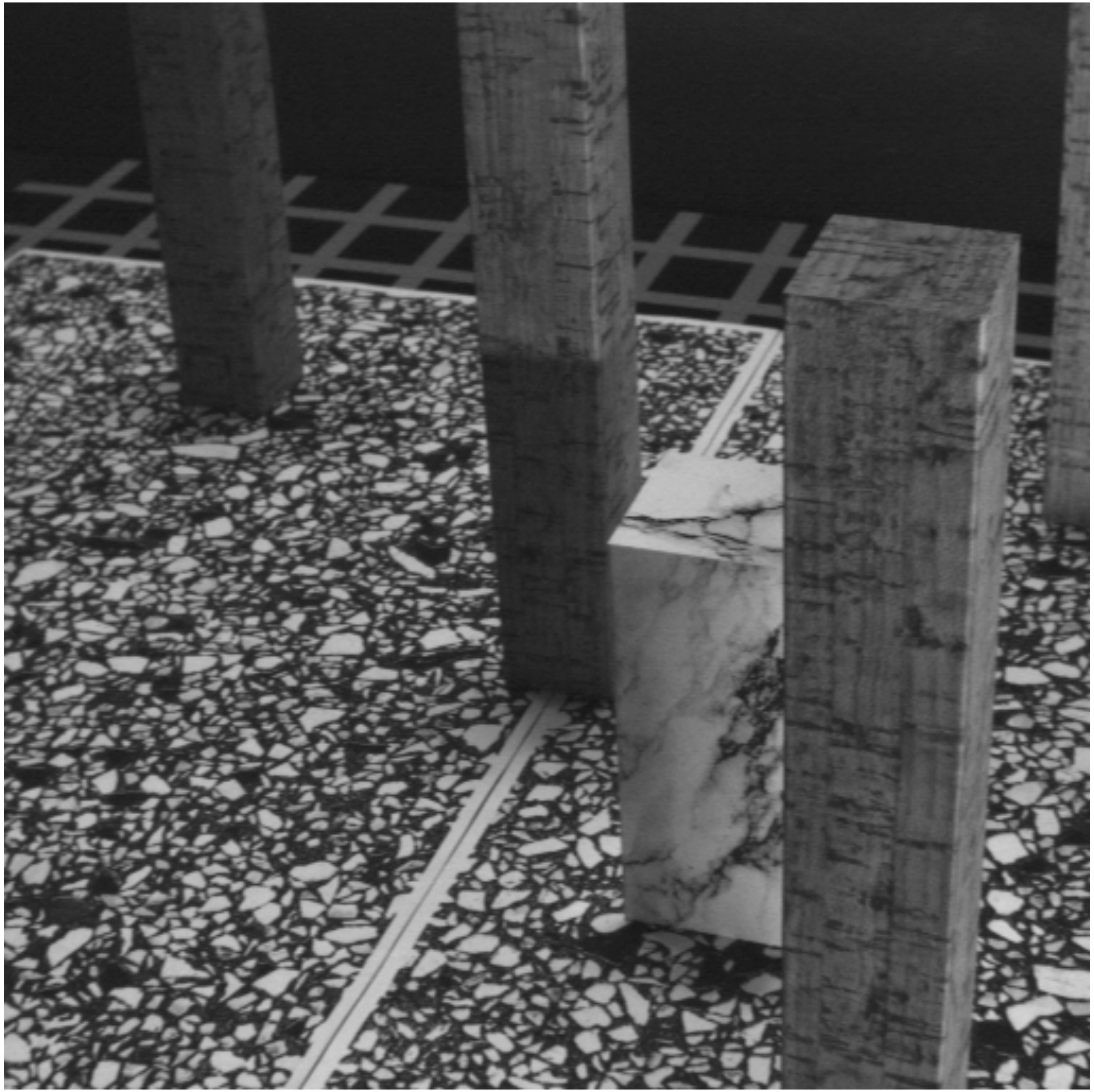
Dense Optical Flow



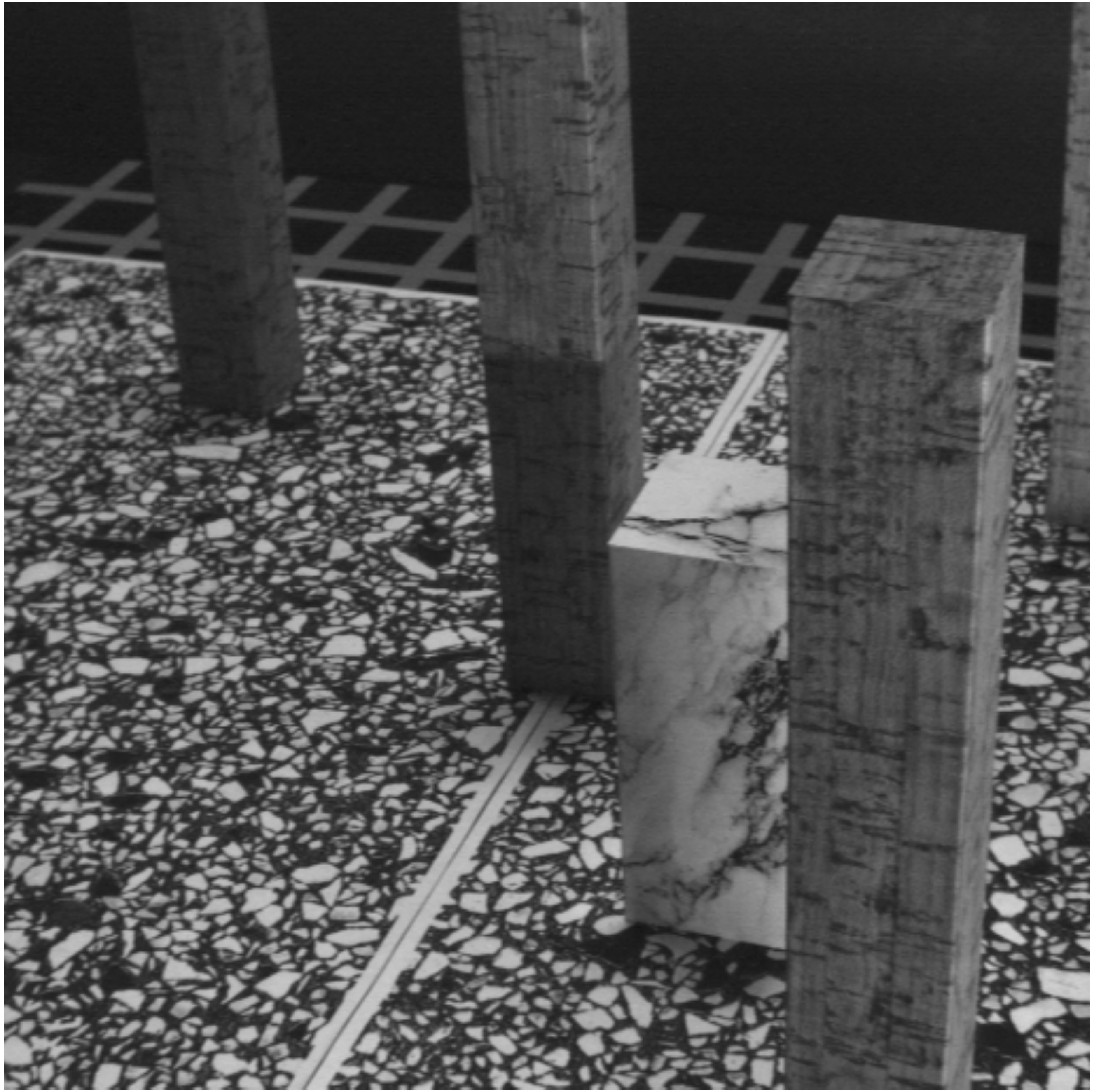
Dense Optical Flow



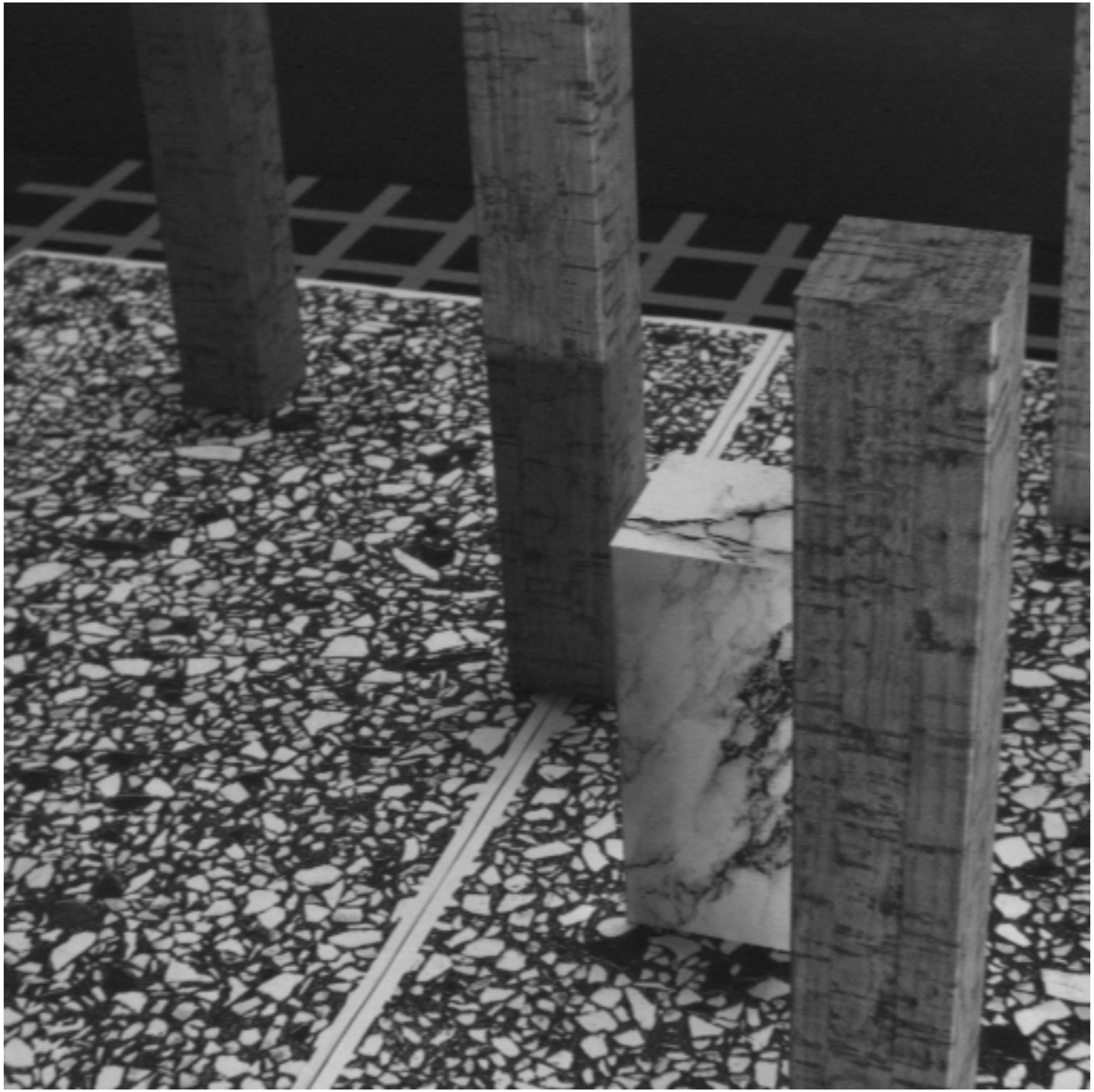
Two velocities because the window is over an occluding boundary



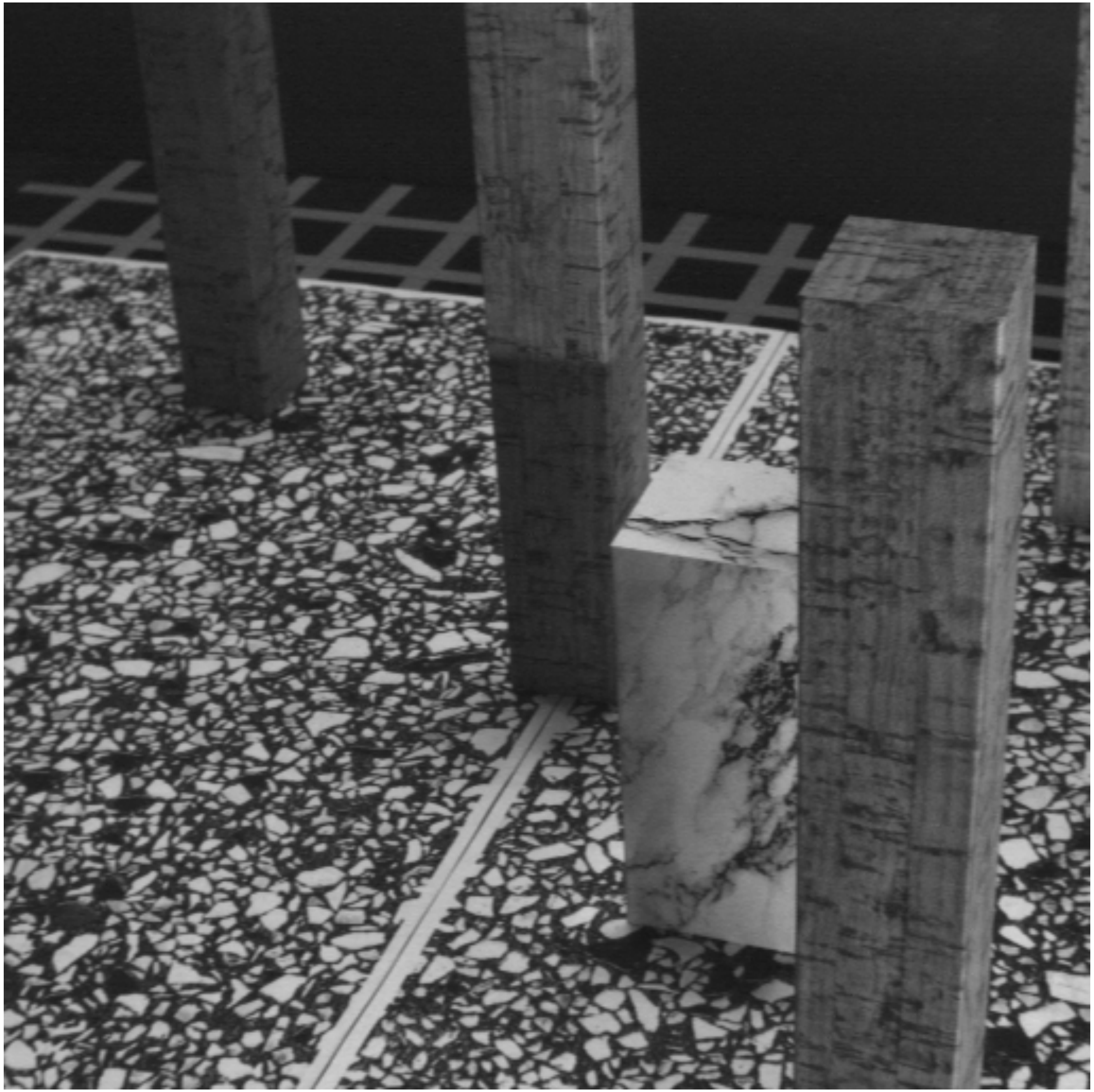
Otte and Nagel's Benchmark Sequence



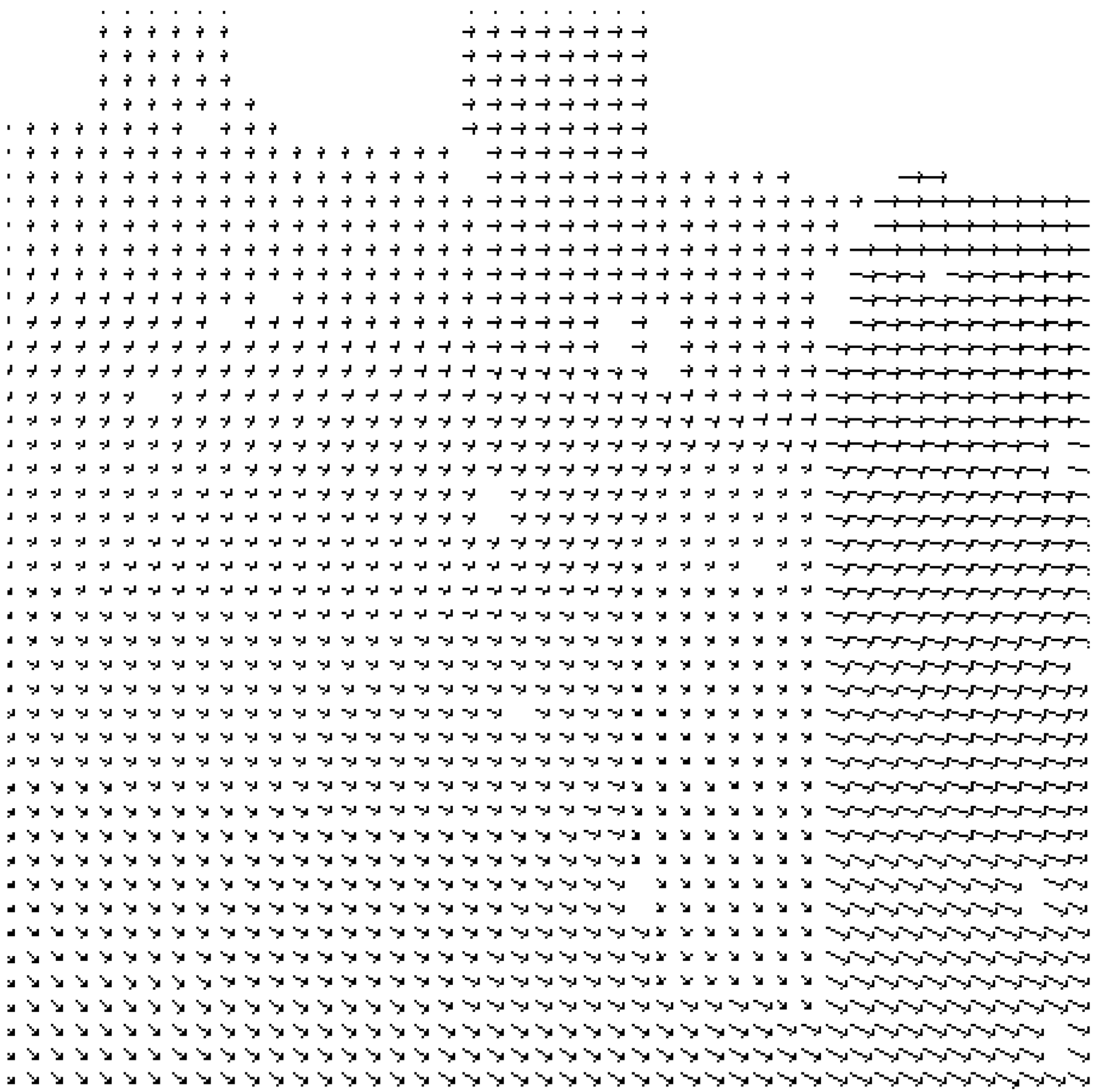
Otte and Nagel's Benchmark Sequence



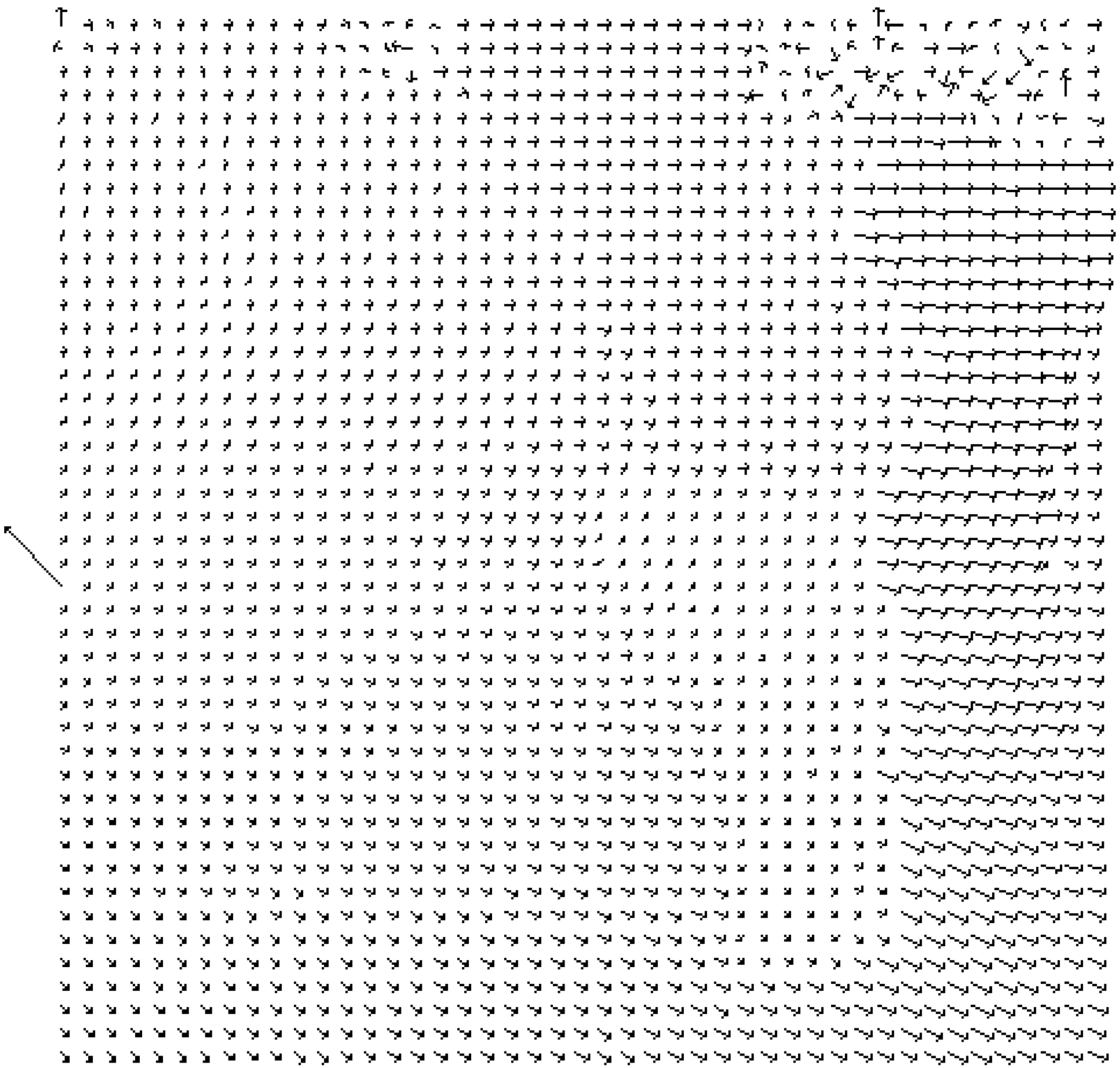
Otte and Nagel's Benchmark Sequence



Otte and Nagel's Benchmark Sequence

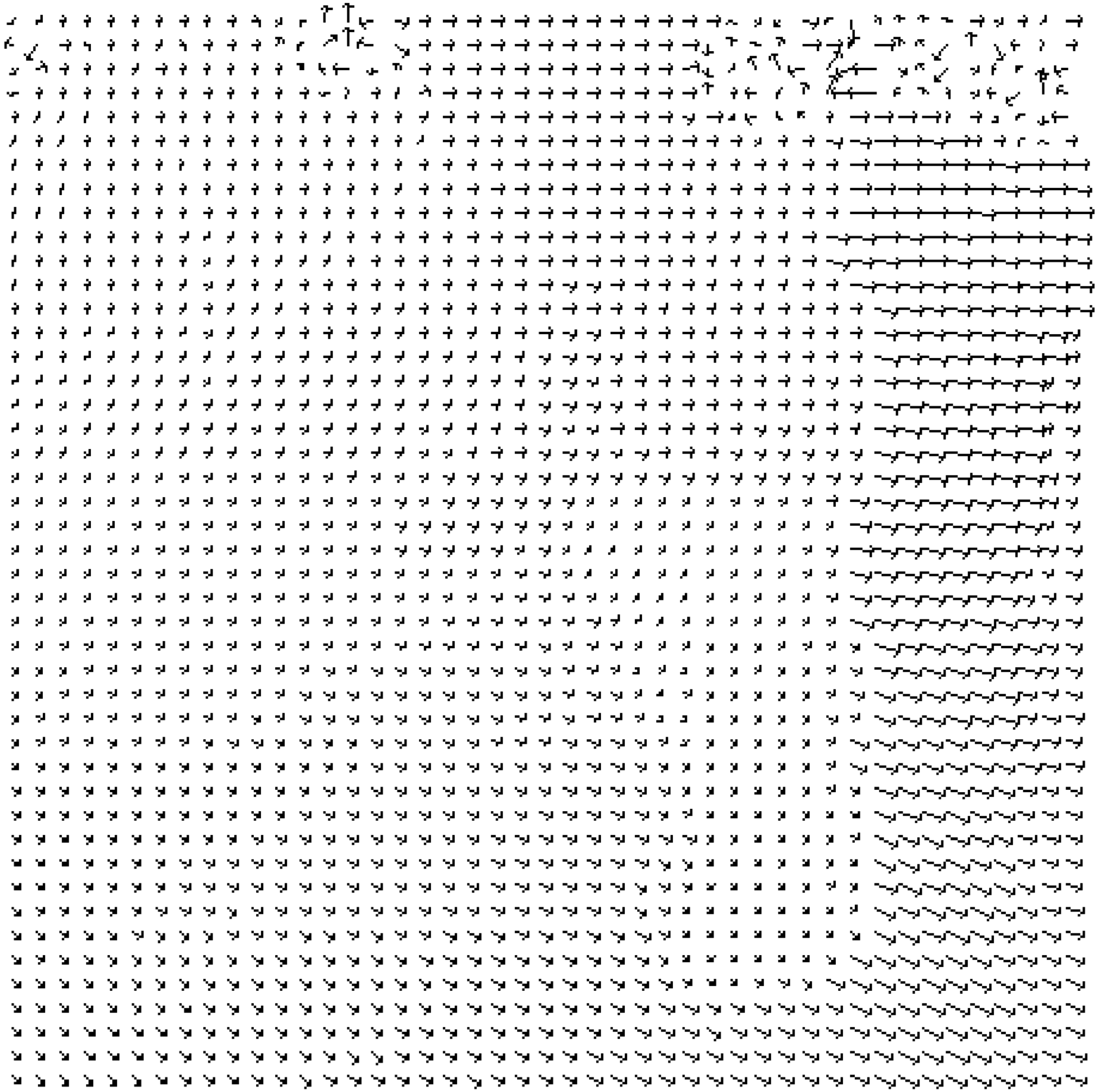


Ground Truth optical flow field for Otte and Nagel's Benchmark Sequence



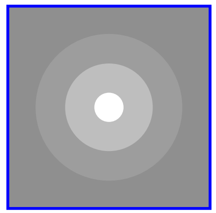
Computed optical flow field for Otte and Nagel's Benchmark Sequence

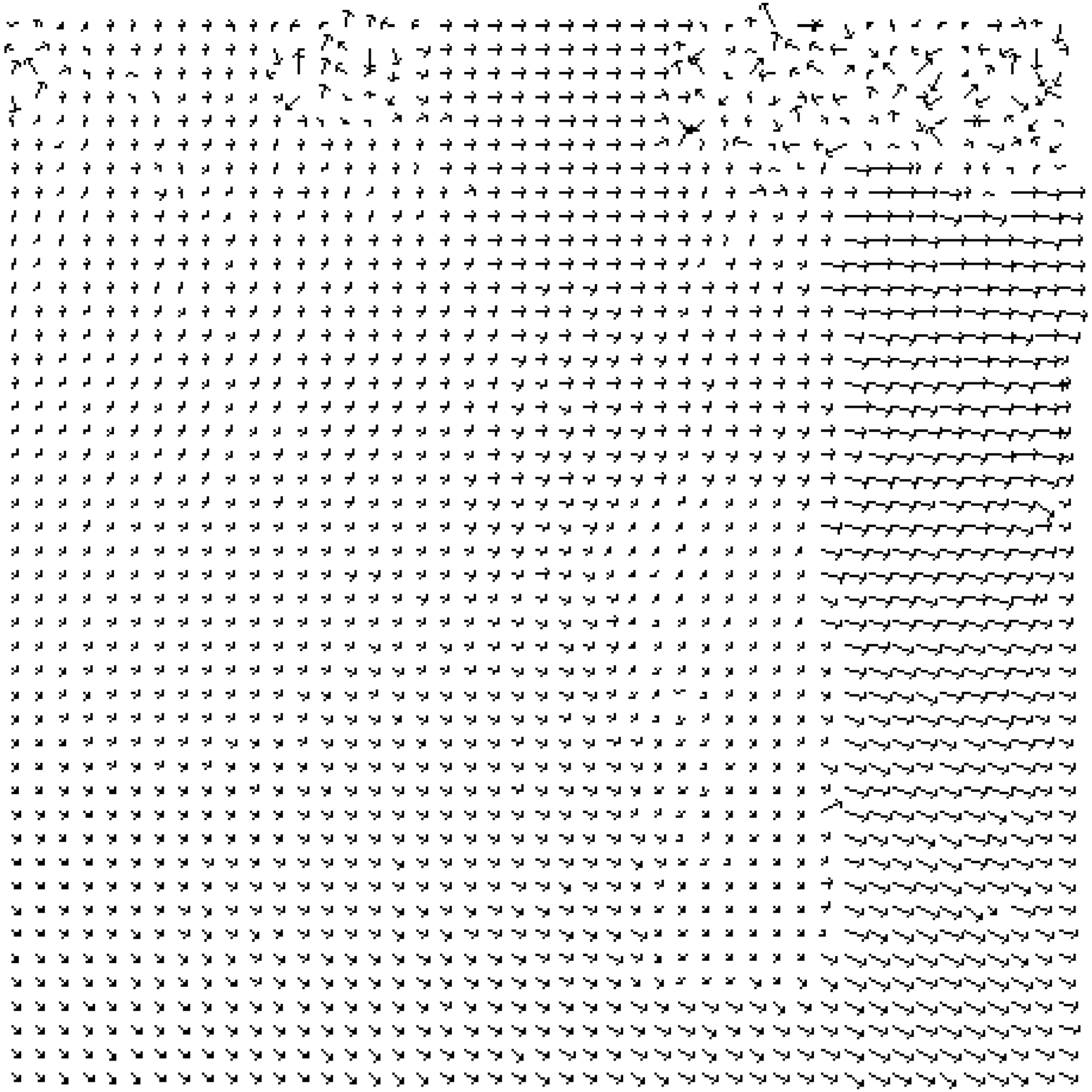
Windowing function 50% weighting at 3w/8 pixels from centre



Computed optical flow field for Otte and Nagel's Benchmark Sequence

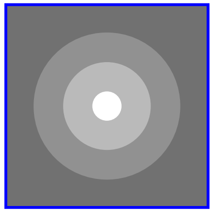
Windowing function 50% weighting at 2w/8 pixels from centre

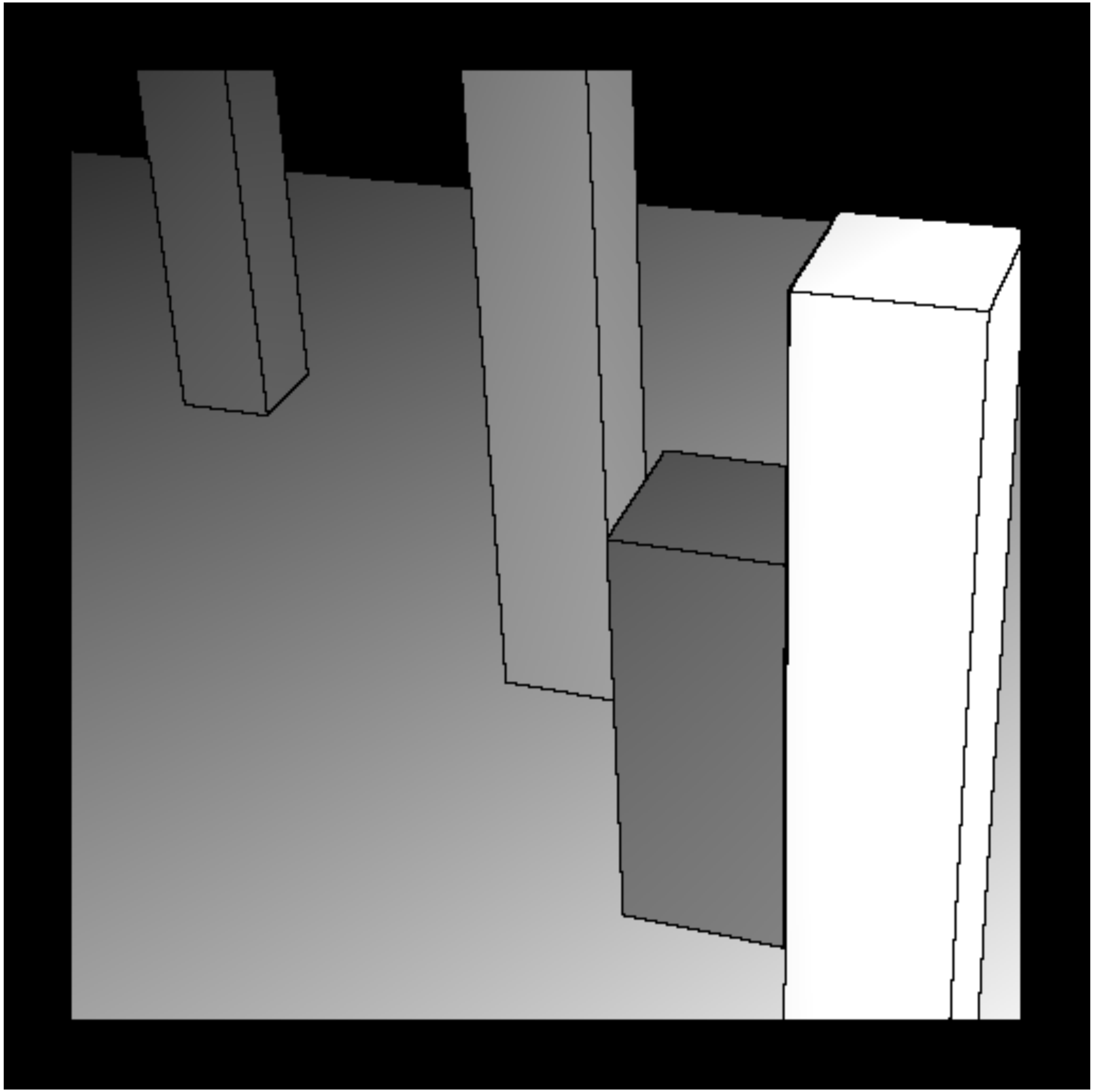




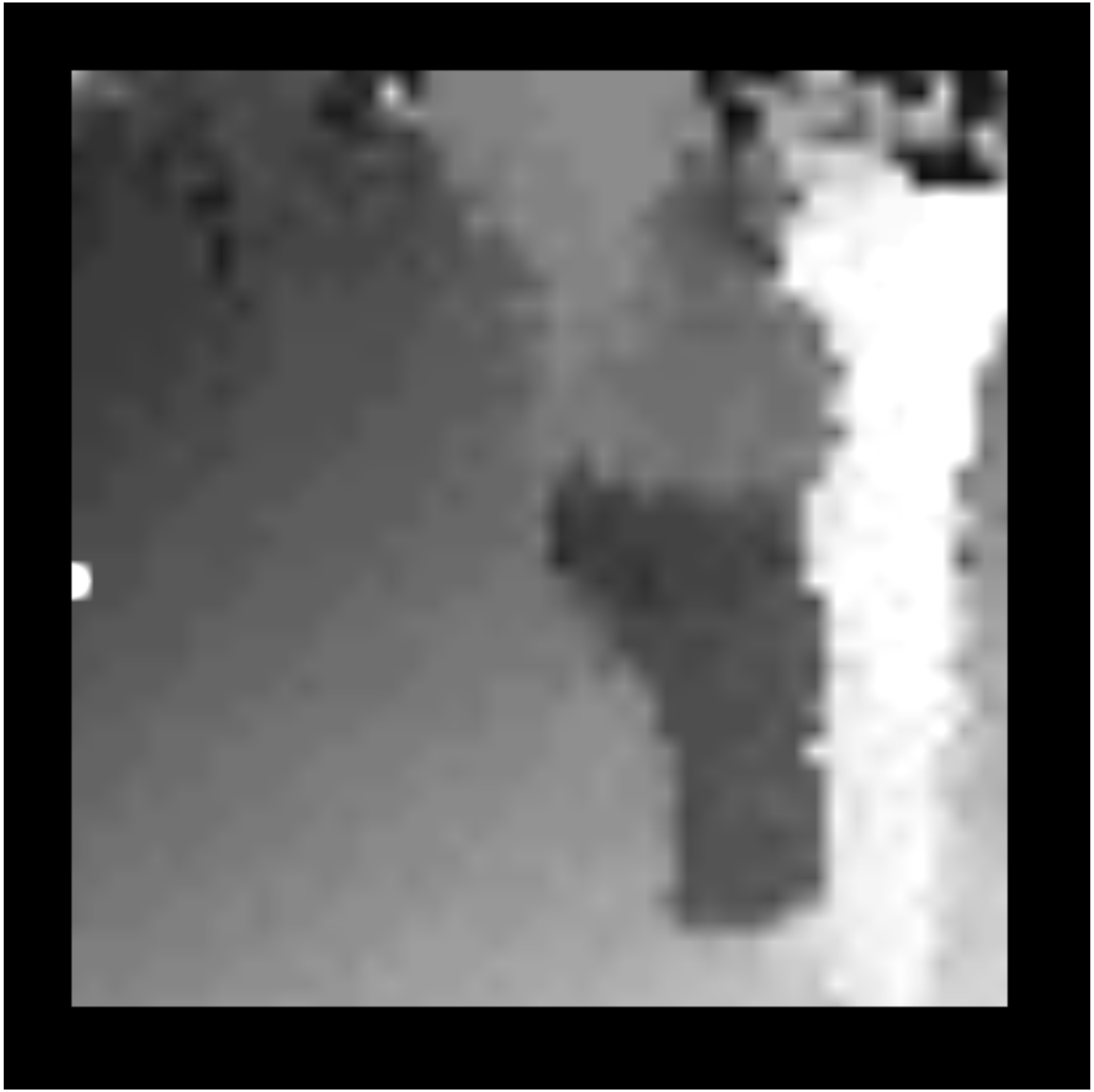
Computed optical flow field for Otte and Nagel's Benchmark Sequence

Windowing function 50% weighting at w/8 pixels from centre





Ground Truth flow magnitude for Otte and Nagel's Benchmark Sequence



Computed
flow magnitude for Otte and Nagel's Benchmark Sequence

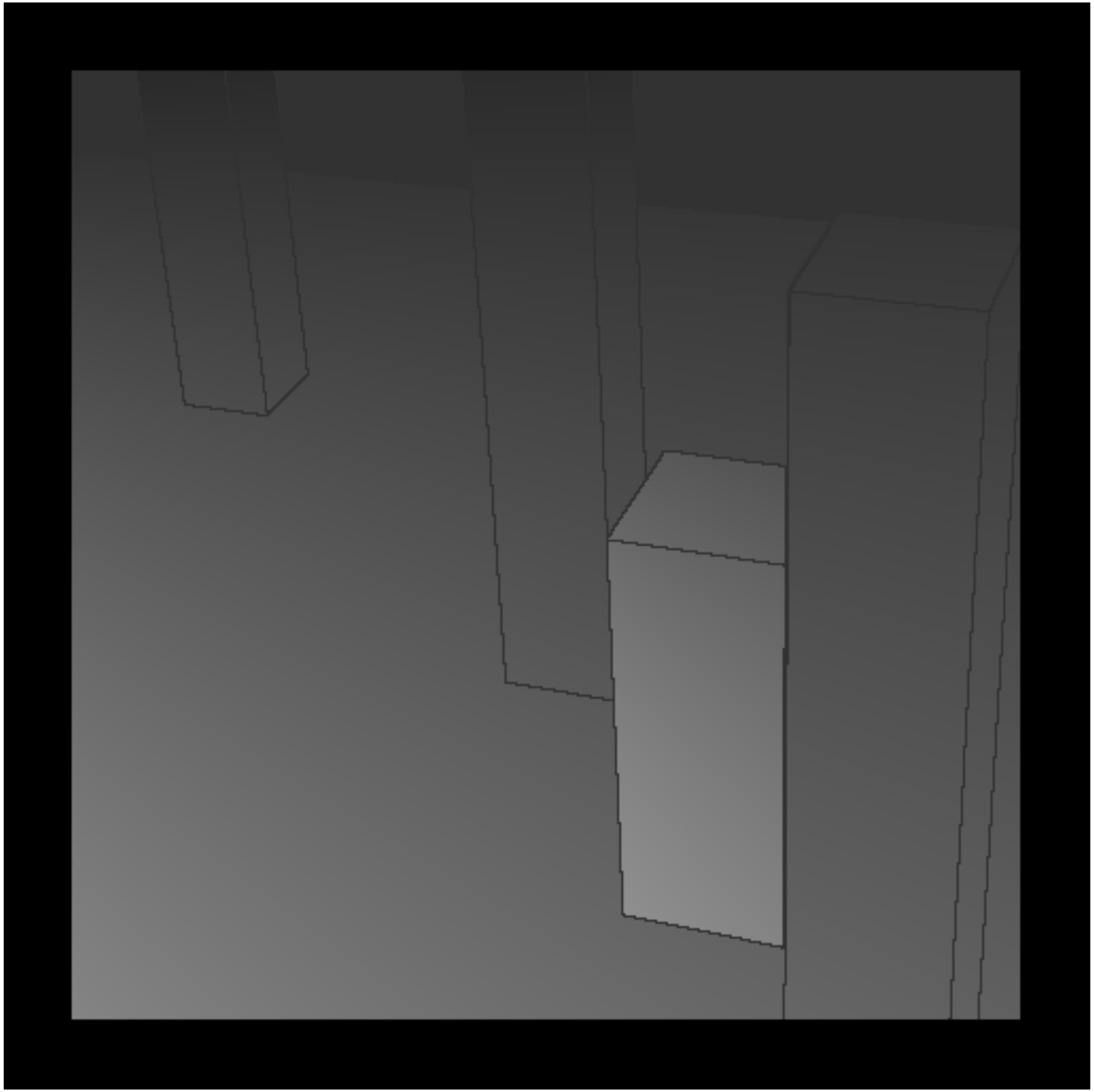
Windowing function 50% weighting at 3w/8 pixels from centre

Computed
flow magnitude for Otte and Nagel's Benchmark Sequence

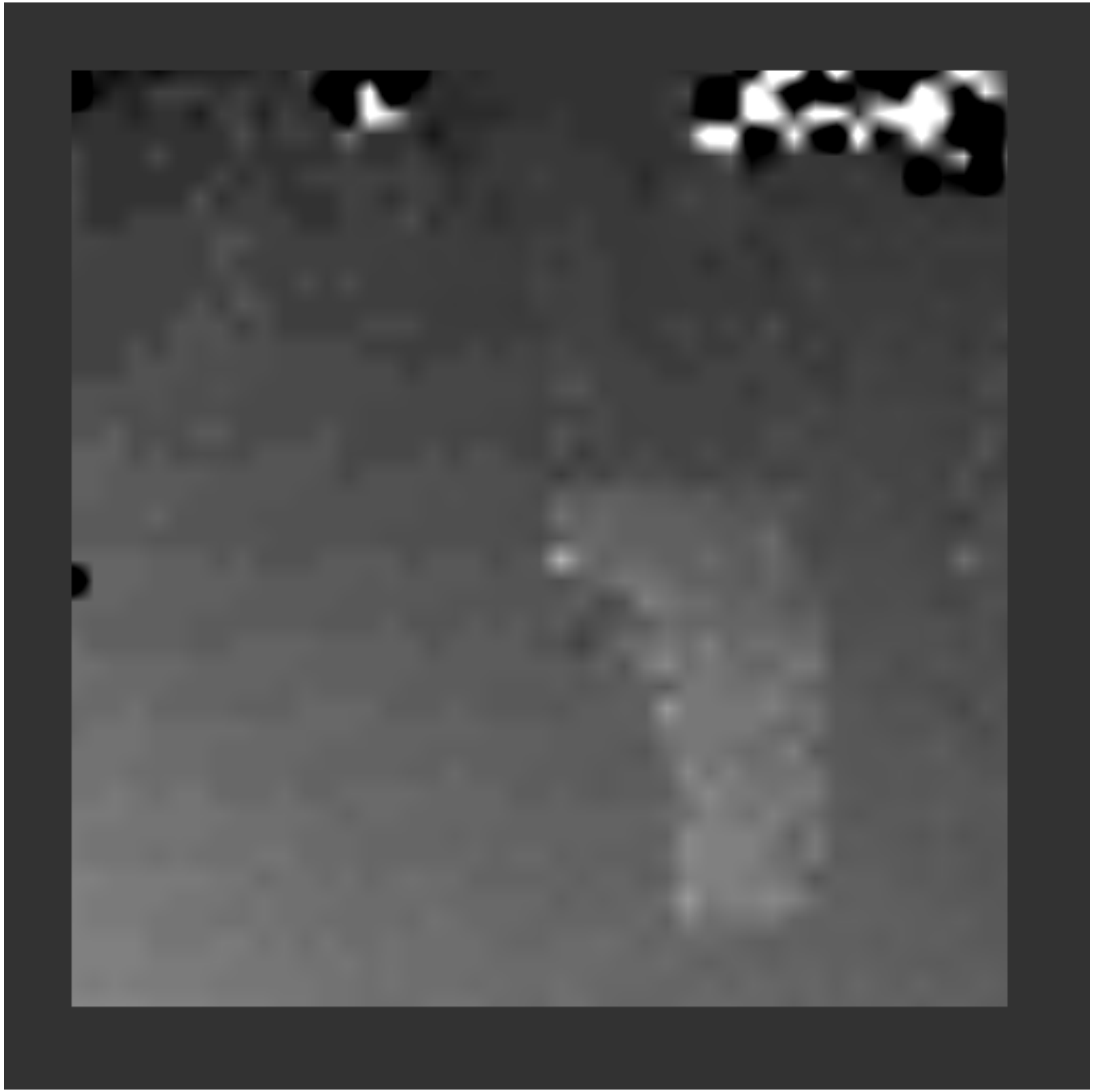
Windowing function 50% weighting at 2w/8 pixels from centre

Computed
flow magnitude for Otte and Nagel's Benchmark Sequence

Windowing function
50% weighting at
w/8 pixels from centre

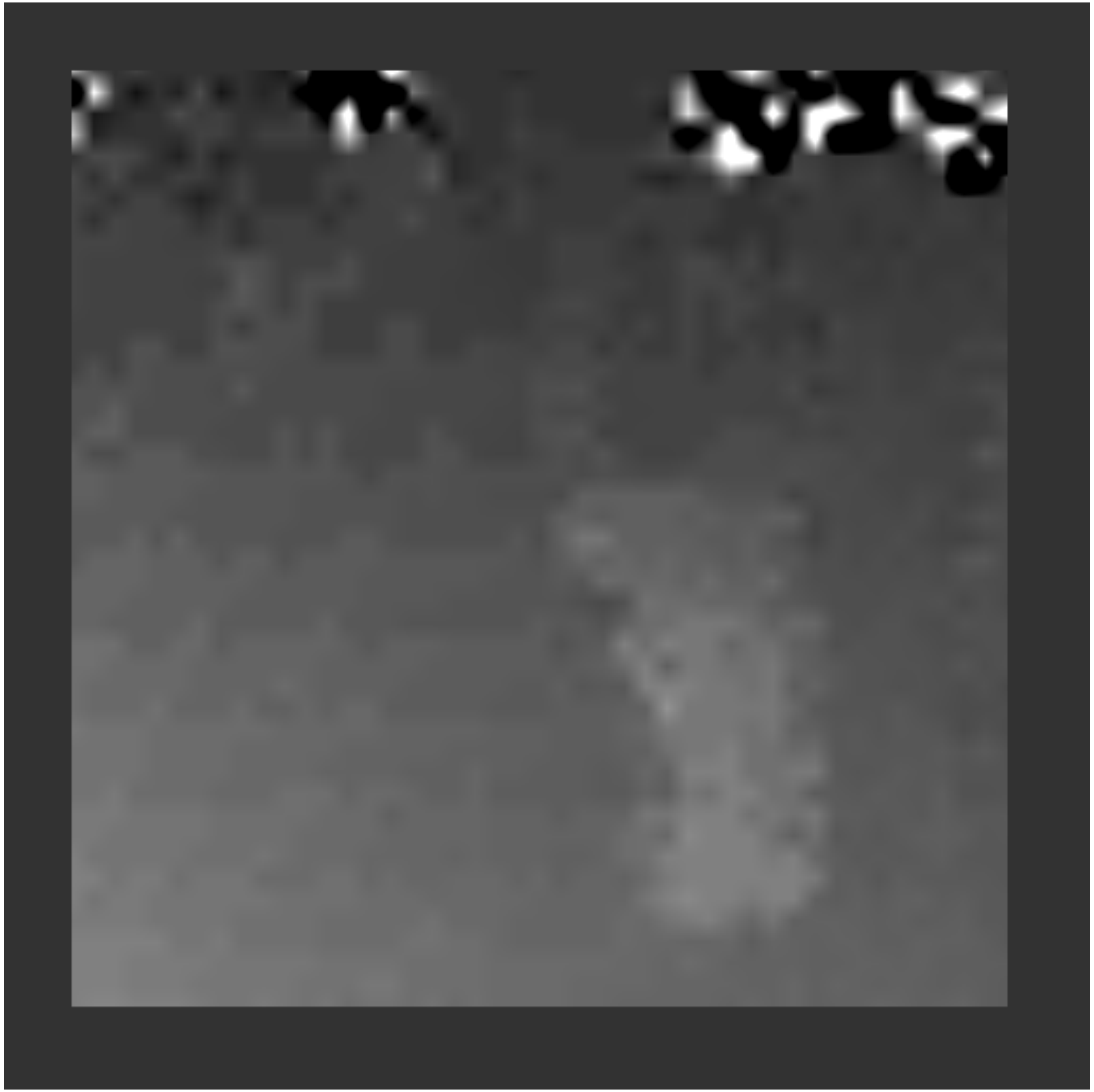


Ground Truth flow direction for Otte and Nagel's Benchmark Sequence



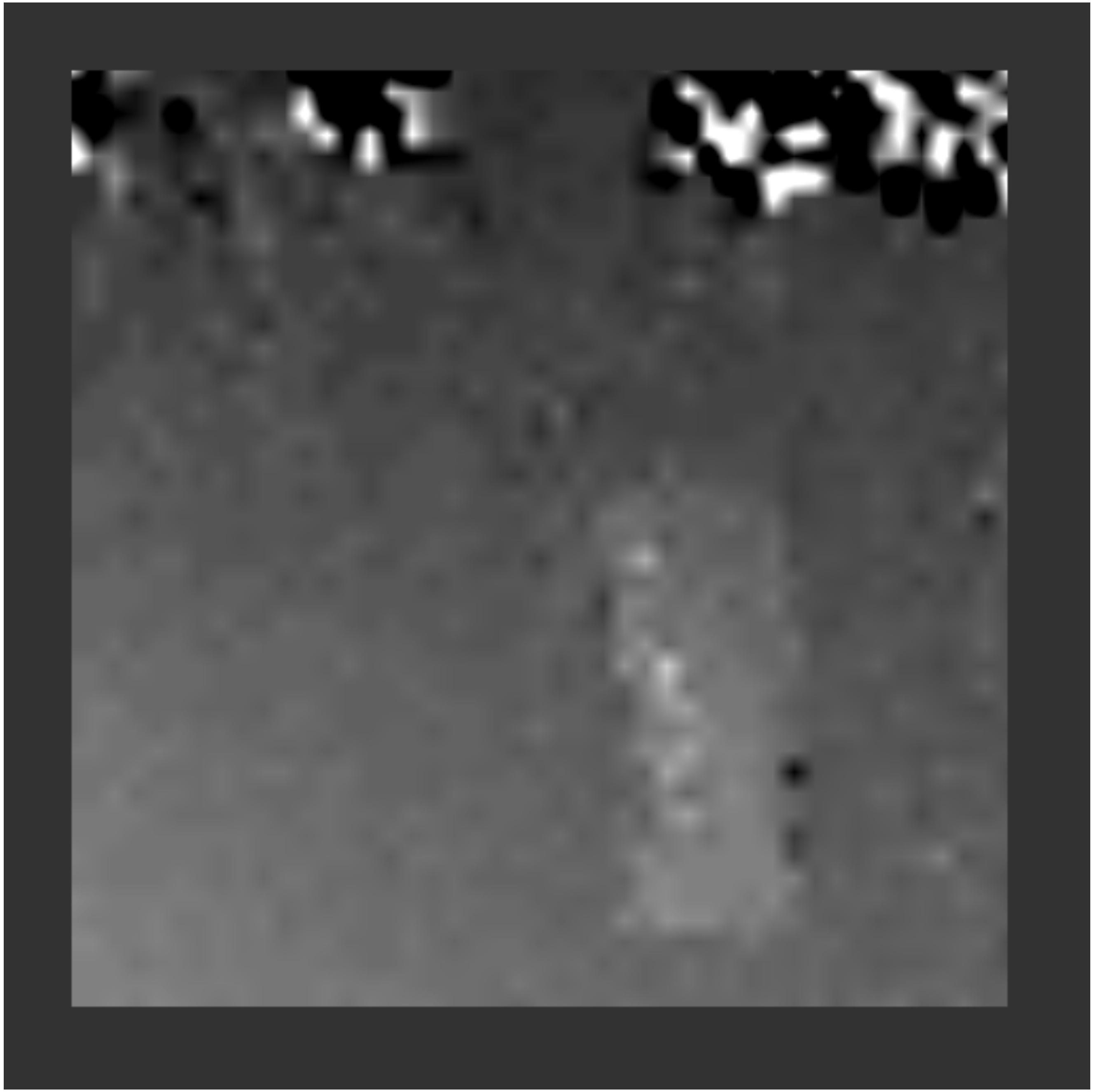
Computed
flow direction for Otte
and Nagel's Benchmark
Sequence

Windowing function
50% weighting at
3w/8 pixels from
centre



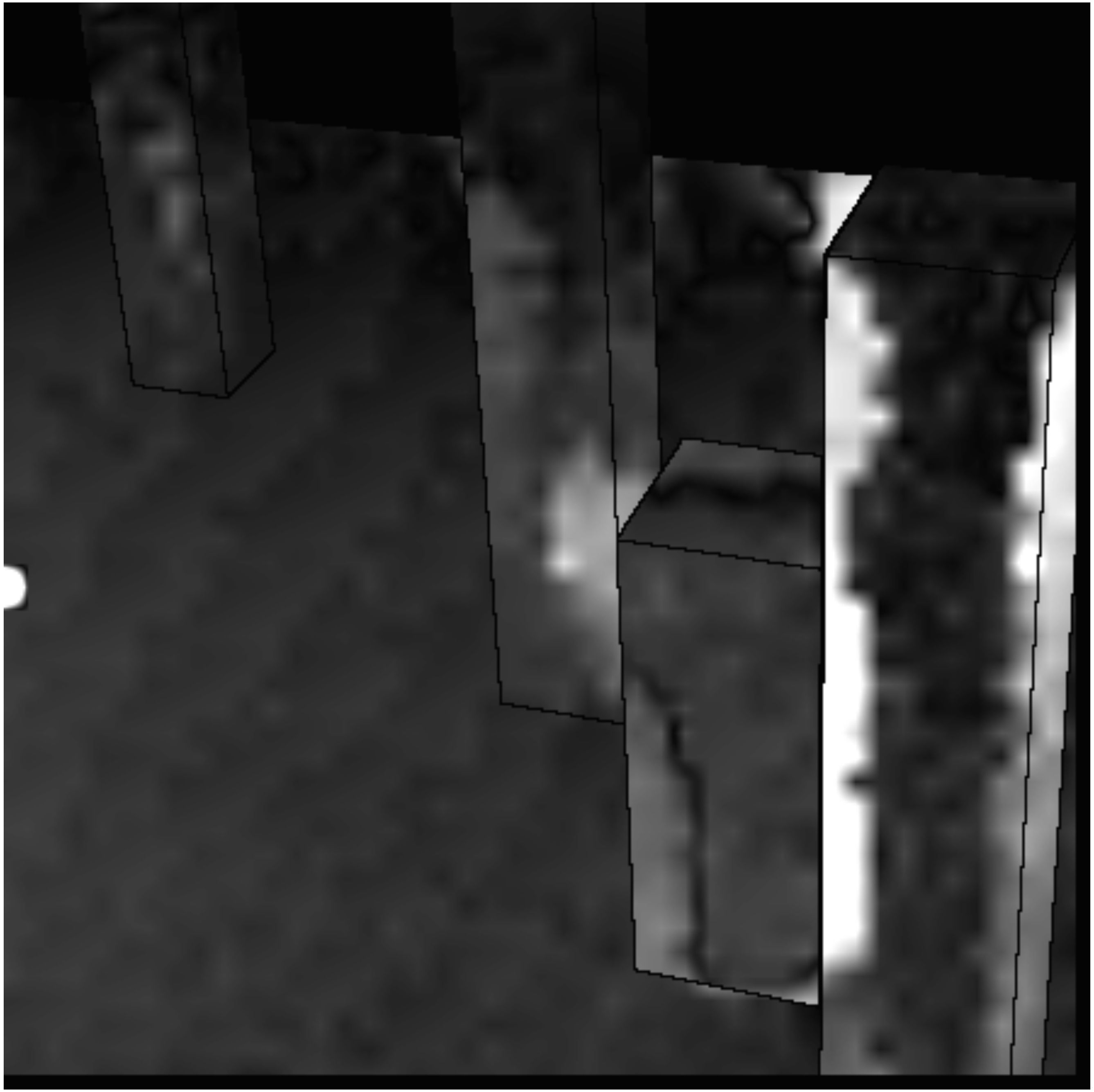
Computed
flow direction for Otte and Nagel's Benchmark Sequence

Windowing function
50% weighting at
2w/8 pixels from
centre



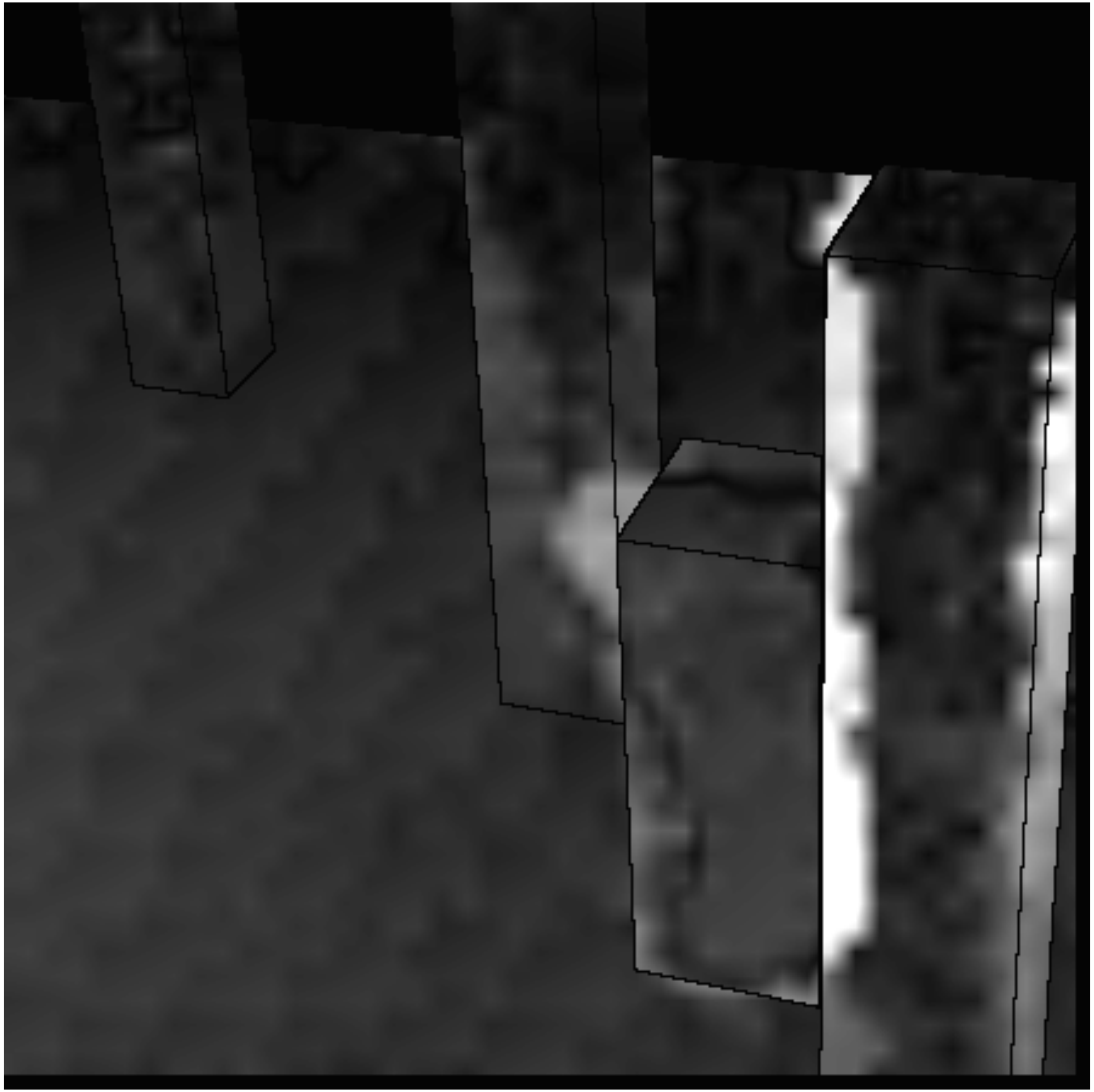
Computed
flow direction for Otte
and Nagel's Benchmark
Sequence

Windowing function
50% weighting at
w/8 pixels from centre



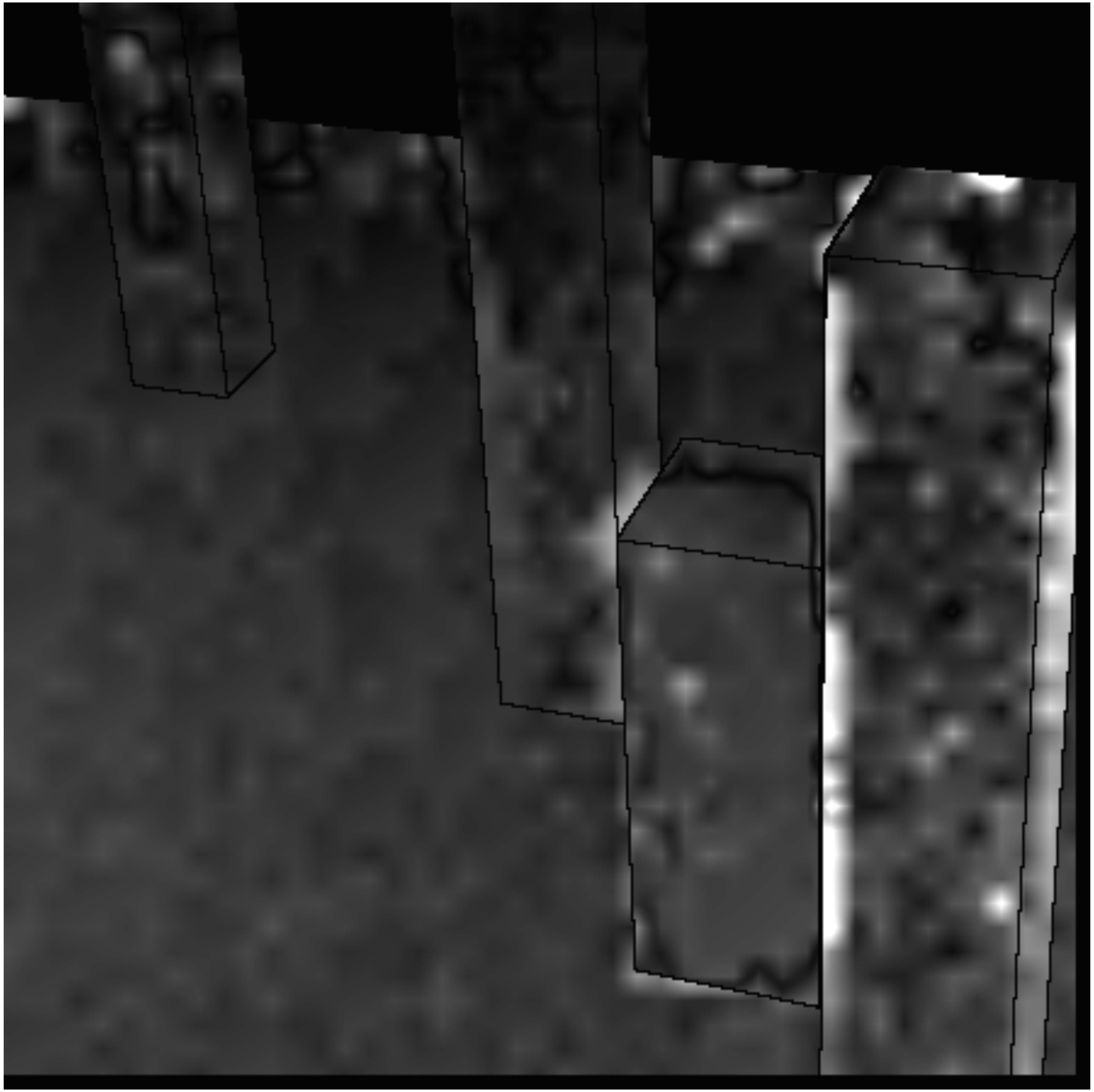
Nagel Error Measure

Windowing function 50% weighting at 3w/8 pixels from centre



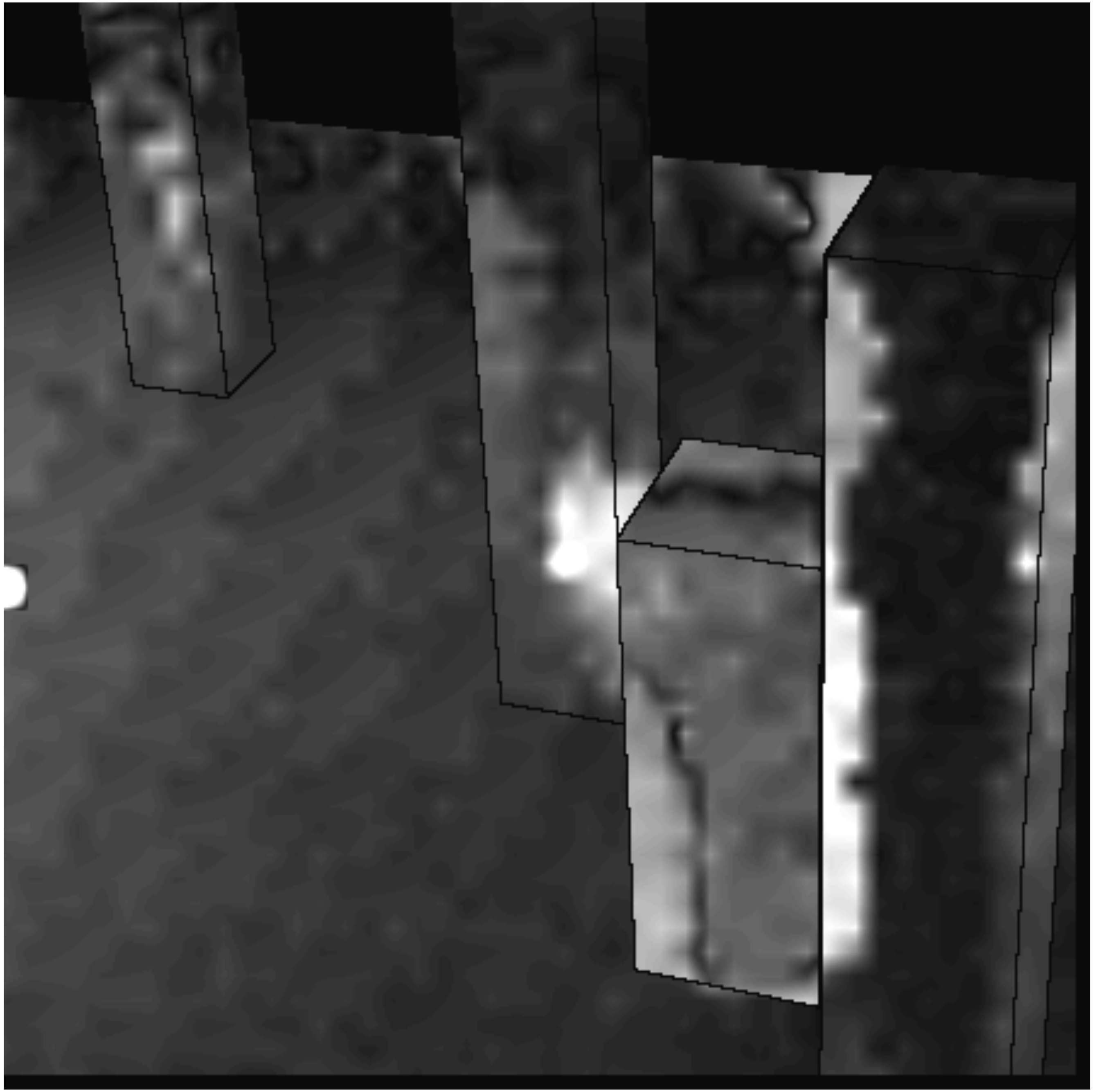
Nagel Error Measure

Windowing function
50% weighting at
2w/8 pixels from
centre



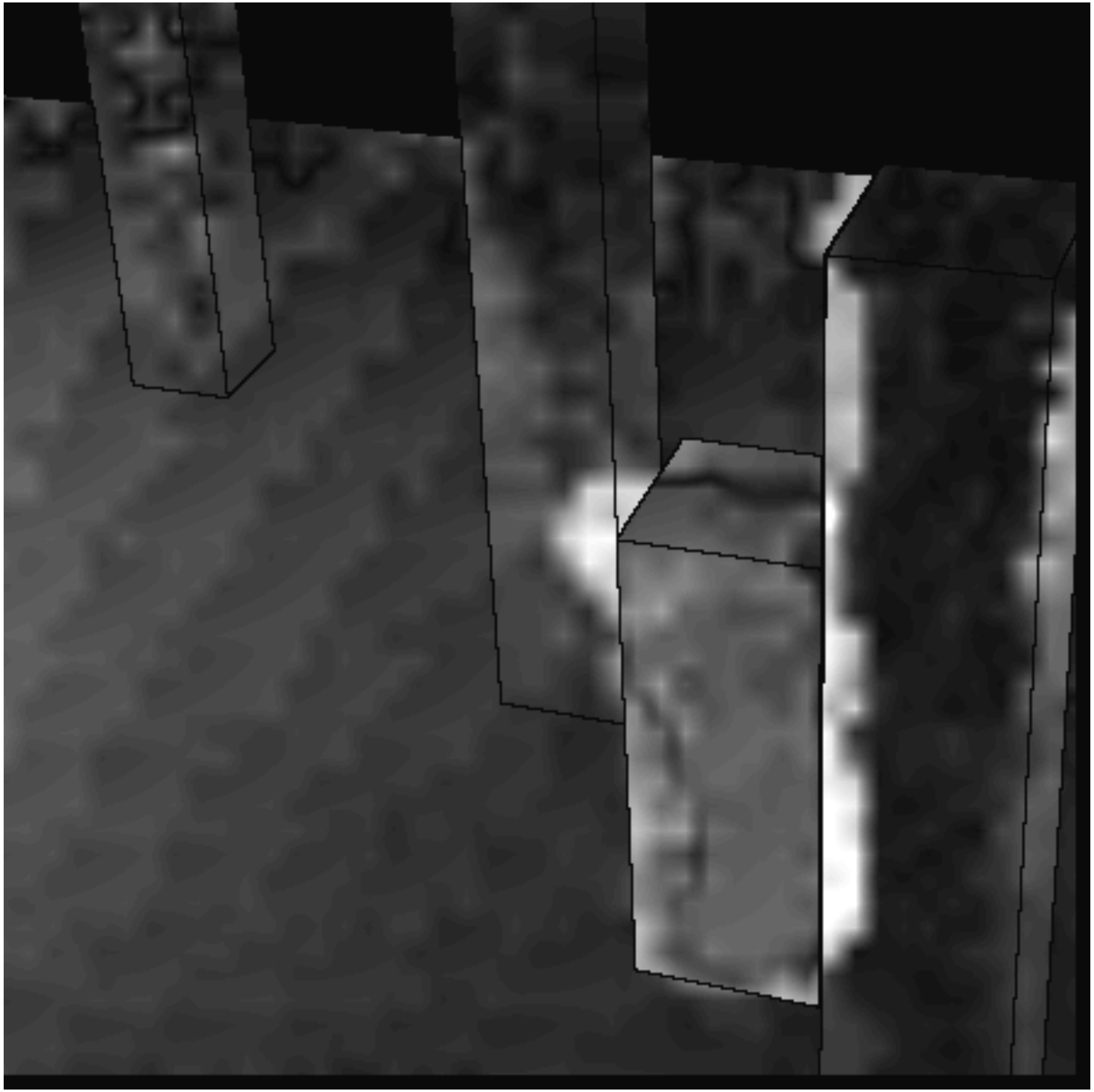
Nagel Error Measure

Windowing function
50% weighting at
w/8 pixels from centre



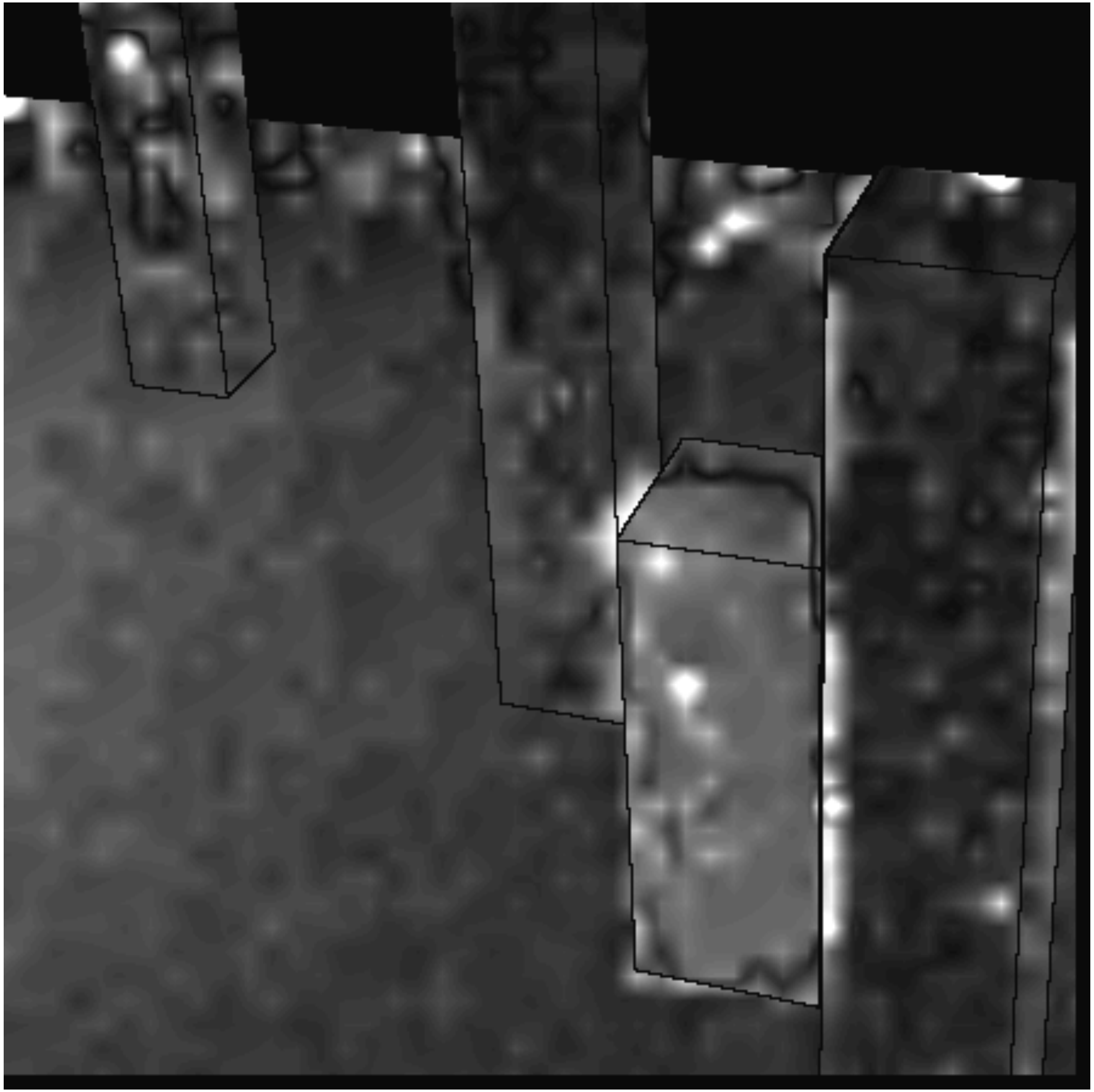
Fleet Error Measure

Windowing function
50% weighting at
3w/8 pixels from
centre



Fleet Error Measure

Windowing function
50% weighting at
2w/8 pixels from
centre



Fleet Error Measure

Windowing function
50% weighting at
w/8 pixels from centre

Demos

The following code is taken from the trackingMeanShift project in the lectures directory of the ACV repository

See:

```
trackingMeanShift.h
trackingMeanShiftImplementation.cpp
trackingShiftApplication.cpp
```

```
/*
  Example use of openCV to track an object using histogram backprojection and mean-shift adjustment

  Read a sequence of video filenames, start frame number, and pairs of coordinates defining
  a rectangular region of interest surrouding the object to be tracked.

  For example, filename video.avi; frame 99; RoI (30,20), (64,50):

  ../data/media/video1.avi 99  34 20  64 50
  ---------------------------------------------------------------------------------------------------
  Application file

  David Vernon
  3 November 2017
*/

#include "trackingMeanShift.h"

int main() {

   int end_of_file;
   bool debug = true;
   char filename[MAX_FILENAME_LENGTH];
   VideoCapture video;                  //  the video device
   char* backprojected_window_name = "Backprojected Image";
   char* tracking_window_name      = "Mean Shift Tracking";
   Mat previousFrame;
   Mat currentFrame;
   Mat backProjectedImage;
   int starting_frame_number;
   int x1, y1, x2, y2;
   Rect currentPosition;
```

```
FILE *fp_in;

printf("Example use of openCV to track an object using mean shift\n\n");

if ((fp_in = fopen("../data/trackingMeanShiftInput.txt","r")) == 0) {
  printf("Error can't open input file trackingMeanShiftInput.txt\n");
  prompt_and_exit(1);
}

/*  Create a windows for image of tracked object */
namedWindow(tracking_window_name, CV_WINDOW_AUTOSIZE );

do {
   end_of_file = fscanf(fp_in, "%s %d %d %d %d %d", filename, &starting_frame_number, &x1, &y1, &x2, &y2);

   if (end_of_file != EOF) {

      printf("Press any key to continue ...\n");

      video.open(filename);                     // open the video input
      video.set(CV_CAP_PROP_POS_FRAMES,starting_frame_number);

      if (video.isOpened()) {

         printf("Press any key to stop image display\n");

         Rect position(x1, y1, x2, y2);  // region of interest in which object appears in the first frame
```

```
            do {
               video >> currentFrame;

               if (!currentFrame.empty()) {

                  trackingMeanShift(currentFrame, backProjectedImage, position); // position is updated on each call
                  rectangle(currentFrame, position,Scalar(0,255,0),2);           // draw current position of object

                  imshow(tracking_window_name, currentFrame);
                  imshow(backprojected_window_name, backProjectedImage);
                  waitKey(30);

                  currentFrame.copyTo(previousFrame);
               }
            } while ((!_kbhit()) && (!currentFrame.empty()));

            if (_kbhit())
               getchar(); // flush the buffer from the keyboard hit

            waitKey(30); // allow user to move the windows after the sequence has finished
         }
         else {
            cout << "can not open " << filename << endl;
         }
      }
   } while (end_of_file != EOF);

   destroyWindow(tracking_window_name);
   fclose(fp_in);
   return 0;
}
```

```
/*
  Example use of openCV to track an object using histogram backprojection and mean-shift adjustment
  -------------------------------------------------------------------------------------------------
  Implementation file

  David Vernon
  3 November 2017
*/

#include "trackingMeanShift.h"

/* In the first call to this function, extract the histogram in the region of interest given by the position rectangle         */
/* In this call and all subsequent calls, backproject this histogram and use mean-shift to adjust the location of the rectangle */
/*                                                                                                                              */
/* For documentation, see  https://docs.opencv.org/2.4/modules/video/doc/motion_analysis_and_object_tracking.html#meanshift     */
/* See in particular the advice about pre-filtering the backprojected image to remove small 'noisy' regions                     */

void trackingMeanShift(Mat& currentFrame, Mat& backProjectedImage, Rect& position) {

   /* -------------------------------------------------------------------------------------------------
    * Adapted from code provided as part of "A Practical Introduction to Computer Vision with OpenCV"
    * by Kenneth Dawson-Howe © Wiley & Sons Inc. 2014.  All rights reserved.
    */

   static bool firstCall = true;
   float channel_range[2] = { 0.0, 255.0 };
   int channel_numbers[1] = { 0 };
   int number_bins[1] = { 32 };
   static MatND histogram[1];
   int chosen_channel = 0;    // Hue channel
   const float* channel_ranges = channel_range;
   Mat back_projection_probabilities;

   Mat saturation_mask;
   Mat hls_image;
   std::vector<cv::Mat> hls_planes(3);
```

```
    /* extract the hue plane */
    cvtColor(currentFrame, hls_image, CV_BGR2HLS);
    split(hls_image,hls_planes);

    if (firstCall) { // DV
       /* first frame so extract the histogram of the image in the region of interest containing the object to be tracked */
       Mat image1ROI = hls_planes[chosen_channel](position);
       calcHist(&(image1ROI), 1, channel_numbers, Mat(), histogram[0], 1 , number_bins, &channel_ranges);
       normalize(histogram[0],histogram[0],1.0);
       firstCall = false;
    }

    /* Calculate back projection */
    calcBackProject(&(hls_planes[chosen_channel]),1,channel_numbers,*histogram,back_projection_probabilities,&channel_ranges,255.0);

    /* Mean shift */
    TermCriteria criteria(cv::TermCriteria::MAX_ITER,5,0.01);
    meanShift(back_projection_probabilities,position,criteria);

    cvtColor(back_projection_probabilities, backProjectedImage, CV_GRAY2BGR);

    /* ------------------------------------------------------------------------------------------------ */
}
```

Demos

The following code is taken from the farnebackOpticalFlow project in the lectures directory of the ACV repository

See:

```
farnebackOpticalFlow.h
farnebackOpticalFlowImplementation.cpp
farnebackOpticalFlowApplication.cpp
```

```
/*
  Example use of openCV to compute dense optical flow: Farneback algorithm [Farneback 2003]

  [Farneback 2003] Farnebäck G., "Two-Frame Motion Estimation Based on Polynomial Expansion", in Image Analysis,
  Proc. Scandinavian Conference on Image Analysis SCIA, Bigun J., Gustavsson T. (eds) I
  Lecture Notes in Computer Science, vol 2749,  Springer, pp. 363-370, 2003.
  ---------------------------------------------------------------------------------------------------------------
  Application file

  David Vernon
  1 November 2017
*/

#include "opticalFlowFarneback.h"

/* Global variables to allow access by the display window callback functions */
int windowSize                  = 15; // default window size

int main() {

   int end_of_file;
   bool debug = true;
   char filename[MAX_FILENAME_LENGTH];
   int const max_window_size = 31;
   VideoCapture video;                  //  the video device
   char* input_window_name          = "Input Image";
   char* optical_flow_window_name  = "Optical flow: Farneback algorithm";
   Mat previousFrameGreyscale;
   Mat currentFrameGreyscale;
   Mat previousFrame;
   Mat currentFrame;
   Mat opticalFlow;
```

```
FILE *fp_in;

printf("Example use of openCV to compute dense optical flow: Farneback algorithm.\n\n");

if ((fp_in = fopen("../data/opticalFlowFarnebackInput.txt","r")) == 0) {
  printf("Error can't open input file opticalFlowFarnebackStaticInput.txt\n");
  prompt_and_exit(1);
}

/*  Create a windows for input, background, and foreground images */
namedWindow(input_window_name, CV_WINDOW_AUTOSIZE );
namedWindow(optical_flow_window_name, CV_WINDOW_AUTOSIZE );
resizeWindow(optical_flow_window_name,0,0); // this forces the trackbar to be as small as possible (and to fit in the window)
createTrackbar( "Window Size", optical_flow_window_name, &windowSize, max_window_size, windowSizeCallback);

do {
   end_of_file = fscanf(fp_in, "%s", filename);

   if (end_of_file != EOF) {

      printf("Press any key to continue ...\n");

      video.open(filename);                     // open the video input

      if (video.isOpened()) {

         printf("Press any key to stop image display\n");

         video >> previousFrame;
         cvtColor(previousFrame, previousFrameGreyscale, CV_BGR2GRAY);
```

```
            do {
               video >> currentFrame;
               cvtColor(currentFrame, currentFrameGreyscale, CV_BGR2GRAY);

               if (!currentFrame.empty()) {

                  opticalFlowFarneback(previousFrameGreyscale, currentFrameGreyscale, opticalFlow, windowSize);

                  drawOpticalFlow(opticalFlow, previousFrame, 8, Scalar(0, 255, 0), Scalar(0, 0, 255));

                  imshow(input_window_name, currentFrame);
                  imshow(optical_flow_window_name, previousFrame);

                  waitKey(30);

                  currentFrame.copyTo(previousFrame);
                  currentFrameGreyscale.copyTo(previousFrameGreyscale);
               }

            } while ((!_kbhit()) && (!currentFrame.empty()));

            if (_kbhit())
               getchar(); // flush the buffer from the keyboard hit

            waitKey(30); // allow user to move the windows after the sequence has finished
         }
         else {
            cout << "can not open " << filename << endl;
         }
      }
   } while (end_of_file != EOF);

   destroyWindow(input_window_name);
   destroyWindow(optical_flow_window_name);

   fclose(fp_in);

   return 0;
}
```

```
/*
  Example use of openCV to compute dense optical flow: Farneback algorithm [Farneback 2003]

  [Farneback 2003] Farnebäck G., "Two-Frame Motion Estimation Based on Polynomial Expansion", in Image Analysis,
  Proc. Scandinavian Conference on Image Analysis SCIA, Bigun J., Gustavsson T. (eds) I
  Lecture Notes in Computer Science, vol 2749,  Springer, pp. 363-370, 2003.
  -------------------------------------------------------------------------------------------------------------
  Implementation file

  David Vernon
  1 November 2017
*/

#include "opticalFlowFarneback.h"

/*
 * function opticalFlow
 * Trackbar callback - windowSize user input
*/

void opticalFlowFarneback(Mat& previousFrame, Mat& currentFrame, Mat& opticalFlow, int windowSize) {

   /* For documentation, see https://docs.opencv.org/2.4/modules/video/doc/motion_analysis_and_object_tracking.html#calcopticalflowfa

   calcOpticalFlowFarneback(previousFrame, // greyscale image
                            currentFrame,  // greyscale image
                            opticalFlow,   // optical flow image
                            0.5,           // scale factor for each level of the pyramid
                            3,             // number of levels in pyramid
                            windowSize,    // size of region to use in computing flow
                            3,             // number of iterations
                            5,             // degree of polynomial
                            1.2,           // standard deviation for polynomial  ... openCV suggests 1.1 for degree 5
                            OPTFLOW_FARNEBACK_GAUSSIAN // use Gaussian window
                           );
}
```

Demos

The following code is taken from the lucasKanadeOpticalFlow project in the lectures directory of the ACV repository

See:

```
lucasKanadeOpticalFlow.h
lucasKanadeOpticalFlowImplementation.cpp
lucasKanadeOpticalFlowApplication.cpp
```

```
/*
  Example use of openCV to compute dense optical flow: Lucas Kanade Feature Tracker algorithm
  -------------------------------------------------------------------------------------------------------------
  Application file

  David Vernon
  2 November 2017
*/

#include "opticalFlowLucasKanade.h"

/* Global variables to allow access by the display window callback functions */
int windowSize                = 10; // default window size (the eventual window size is four times this number + 1)

int main() {

   int end_of_file;
   bool debug = true;
   char filename[MAX_FILENAME_LENGTH];
   int const max_window_size = 15;
   VideoCapture video;               //  the video device
   char* input_window_name         = "Input Image";
   char* optical_flow_window_name  = "Optical flow: LucasKanade algorithm";
   Mat previousFrameGreyscale;
   Mat currentFrameGreyscale;
   Mat previousFrame;
   Mat currentFrame;
   Mat opticalFlow;
   vector<Point2f> previousFeatures;
   vector<Point2f> currentFeatures;
   vector<uchar>   featuresFound;
   FILE *fp_in;
```

```
printf("Example use of openCV to compute dense optical flow: Lucas Kanade algorithm.\n\n");

if ((fp_in = fopen("../data/opticalFlowLucasKanadeInput.txt","r")) == 0) {
  printf("Error can't open input file opticalFlowLucasKanadeStaticInput.txt\n");
  prompt_and_exit(1);
}

/*  Create a windows for input, background, and foreground images */
namedWindow(input_window_name, CV_WINDOW_AUTOSIZE );
namedWindow(optical_flow_window_name, CV_WINDOW_AUTOSIZE );
resizeWindow(optical_flow_window_name,0,0); // this forces the trackbar to be as small as possible (and to fit in the window)
createTrackbar( "Window/4", optical_flow_window_name, &windowSize, max_window_size, windowSizeCallback);

do {
   end_of_file = fscanf(fp_in, "%s", filename);

   if (end_of_file != EOF) {

      printf("Press any key to continue ...\n");

      video.open(filename);                     // open the video input

      if (video.isOpened()) {

         printf("Press any key to stop image display\n");

         video >> previousFrame;
         cvtColor(previousFrame, previousFrameGreyscale, CV_BGR2GRAY);
```

```
do {
   video >> currentFrame;
   cvtColor(currentFrame, currentFrameGreyscale, CV_BGR2GRAY);

   if (!currentFrame.empty()) {

      opticalFlowLucasKanade(previousFrameGreyscale, currentFrameGreyscale,
                             previousFeatures, currentFeatures, featuresFound,
                             windowSize);

      /* draw flow field */
      for (int i=0; i<(int)previousFeatures.size(); i++) {
         if (featuresFound[i]) {
             circle(previousFrame, previousFeatures[i], 1, Scalar(0,0,255));
             line(previousFrame,   previousFeatures[i], currentFeatures[i], Scalar(0,255,0));
         }
      }

      imshow(input_window_name, currentFrame);
      imshow(optical_flow_window_name, previousFrame);

      waitKey(30);

      currentFrame.copyTo(previousFrame);
      currentFrameGreyscale.copyTo(previousFrameGreyscale);
   }

} while ((!_kbhit()) && (!currentFrame.empty()));

if (_kbhit())
   getchar(); // flush the buffer from the keyboard hit
```

```
            if (_kbhit())
               getchar(); // flush the buffer from the keyboard hit

            waitKey(30); // allow user to move the windows after the sequence has finished
         }
         else {
            cout << "can not open " << filename << endl;
         }
      }
   } while (end_of_file != EOF);

   destroyWindow(input_window_name);
   destroyWindow(optical_flow_window_name);

   fclose(fp_in);

   return 0;
}
```

```
/*
  Example use of openCV to compute dense optical flow: Lucas Kanade Feature Tracker algorithm
  ---------------------------------------------------------------------------------------------------------------
  Implementation file

  David Vernon
  2 November 2017
*/

#include "opticalFlowLucasKanade.h"

/*
 * function opticalFlowLucasKanade
 * Trackbar callback - windowSize user input
*/

void opticalFlowLucasKanade(Mat& previousFrame, Mat& currentFrame,
                            vector<Point2f>& previousFeatures, vector<Point2f>& currentFeatures, vector<uchar>& featuresFound,
                            int windowSize) {

   vector<uchar>   errorFlags;
   windowSize = windowSize*4 + 1; // enlarge window and ensure window size is odd
```

```
   /* see https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html#goodfeaturestotrack */

   goodFeaturesToTrack(previousFrame,                    // greyscale image
                       previousFeatures,                 // identified corners
                       MAX_CORNERS,                      // maximum number of corners
                       0.05,                             // quality level
                       5,                                // minimum distance between features
                       noArray(),                        // region of interest mask
                       3,                                // block size for covariance matrix
                       false,                            // true for Harris corner detector
                       0.04                              // Harris trace multiplier
                      );

   /* see https://docs.opencv.org/2.4/modules/video/doc/motion_analysis_and_object_tracking.html#calcopticalflowpyrlk */

   calcOpticalFlowPyrLK(previousFrame,                    // greyscale image
                        currentFrame,                     // greyscale image
                        previousFeatures,                 // features in previous image
                        currentFeatures,                  // features in current image
                        featuresFound,                    // flag if features found in current frame
                        noArray(),                        // error flags
                        Size(windowSize,windowSize),      // window size; default 21
                        1,                                // max level; default 3
                        TermCriteria(CV_TERMCRIT_ITER |   // number of iterations
                                     CV_TERMCRIT_EPS,     // threshold on accuracy
                                     20,                  // maximum interations; default 30
                                     .3)                  // desired accuracy; default 0.01
                       );

}
```

Reading

R. Szeliski, *Computer Vision: Algorithms and Applications*, Springer, 2010.
Section 8.4 Optical flow