Open CV Tutorial

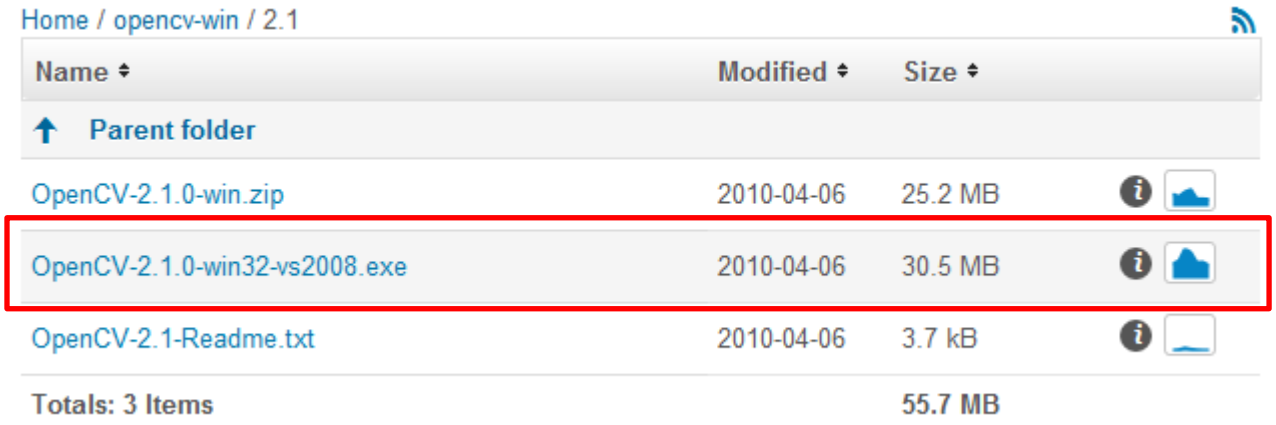
Ubiquitous Computing Lecture

February 15th, 2012

OpenCV installation

Setup – Windows (1)

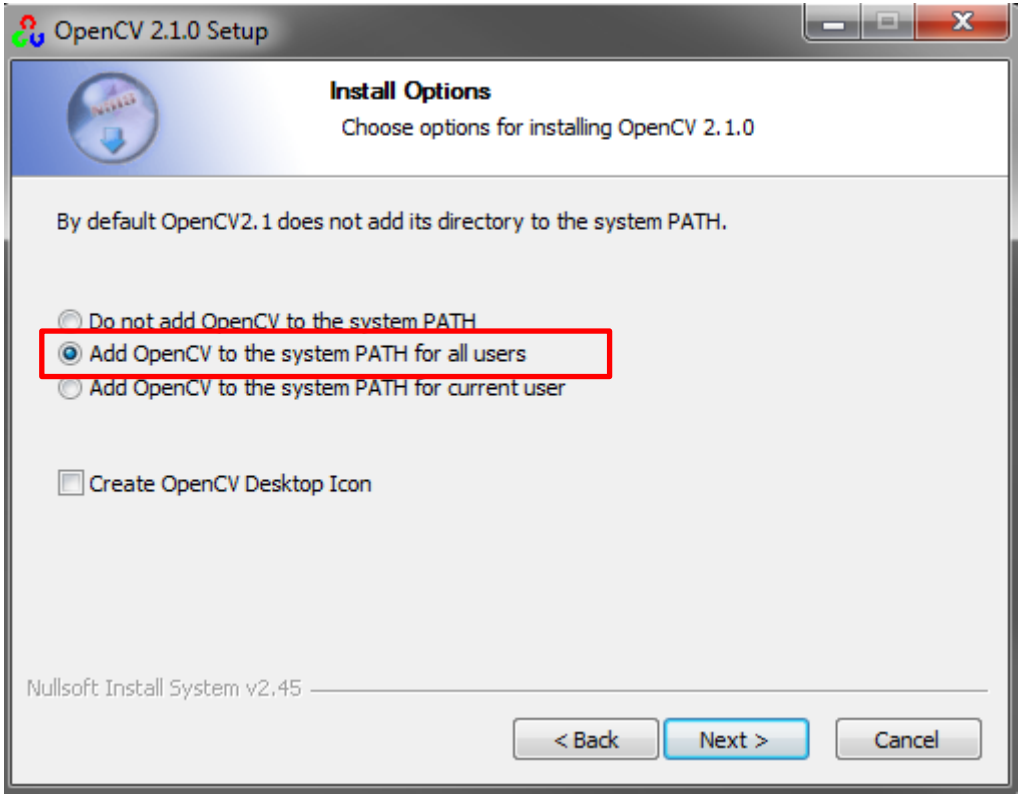
- Download OpenCV 2.1
  http://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.1/



Setup – Windows (2)

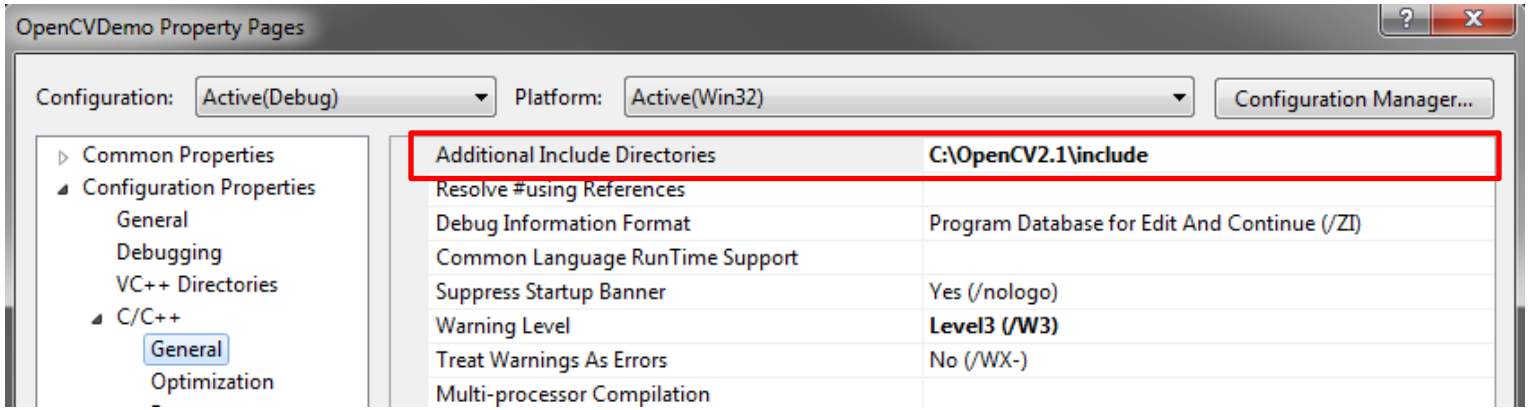
- Installation

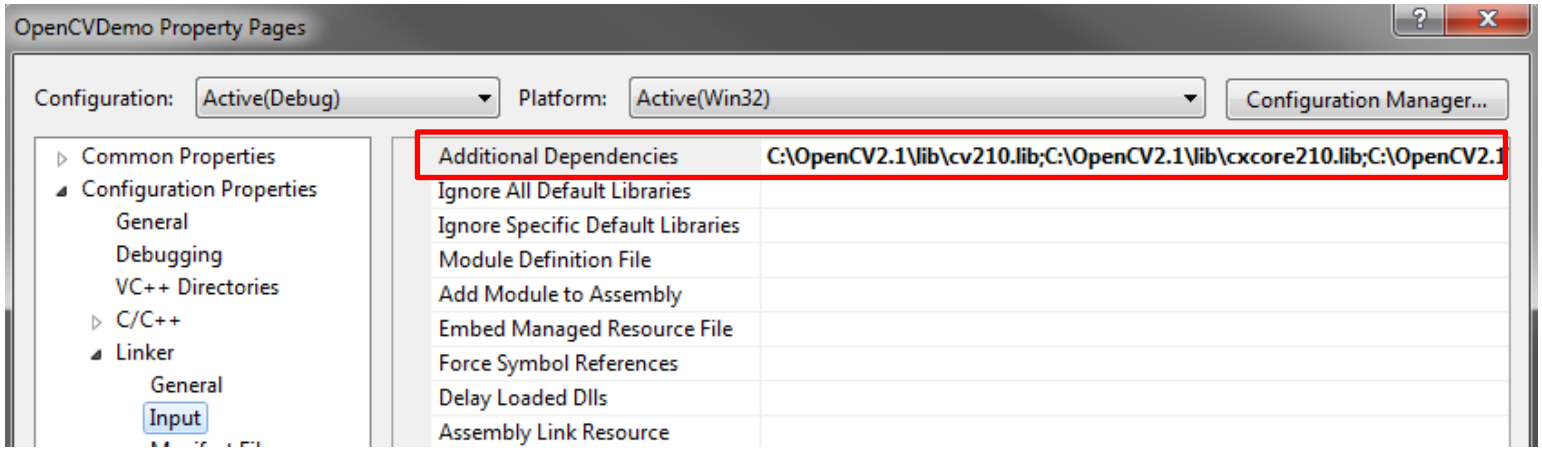
  Add OpenCV DLLs into system path



Setup – Windows (3)

- Include the header files and static libraries





Setup – Linux

- Install OpenCV (say yes to any dependencies):
  $> sudo apt-get install libcv4 libcv-dev libhighgui4 libhighgui-dev libcvaux4 libcvaux-dev

- To build:
  $> gcc -lcv -lhighgui –lcvaux YourFile.cpp

Links

- Main wiki
  http://opencv.willowgarage.com/wiki/

- On-line library
  http://opencv.willowgarage.com/documentation/c/

- Tutorial
  http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html

Getting Images from the Webcam

Two Methods:

- **Event Driven** (or interrupt driven)
  - New Frame == interrupt
  - Adv: constant FPS, multi-threaded processing
  - Dis: overhead, must be real-time

- **Polling** (what we will use, the norm)
  - Grab frame when ready
  - Adv: only process what we can in real time
  - Dis: might grab same frame, variable FPS

Getting Images from the Webcam

```
#include <stdio.h>
#include "opencv/cv.h"
#include "opencv/highgui.h"
#include "opencv/cxcore.h"

int main(int argc, char* argv[])
{
      CvCapture *capture = 0; //The camera
      IplImage* frame = 0; //The images you bring out of the camera

      capture = cvCaptureFromCAM( 0 ); //Open the camera
      if (!capture ){
          printf("Could not connect to camera\n" ); return 1;
      }

      cvNamedWindow( "demowin1", CV_WINDOW_AUTOSIZE ); //Create output window
       while(key != 27 /*escape key to quit*/)
      {
          //Query for the next frame
          frame = cvQueryFrame( capture );
          if( !frame ) break;
          //show the raw image in the first window
          cvShowImage( "demowin1", frame );
          //Get the last key that's been pressed for input
          key = cvWaitKey( 1 );

      }
      //Shutdown
      cvReleaseCapture( &capture ); // WARNING: DO NOT release the "frame" variable
      cvDestroyWindow( "demowin1" );
      return 0;
}
```

Saving a video file

```
 int main(int argc, char* argv[])
{
      CvCapture *capture = 0; //The camera
      IplImage* frame = 0; //The images you bring out of the camera

      capture = cvCaptureFromCAM( 0 ); //Open the camera
      if (!capture ){
          printf("Could not connect to camera\n" ); return 1;
      }

      CvVideoWriter *writer;
      int isColor = 1;
      int fps = 25;  // or 30
      writer = cvCreateVideoWriter("output.avi", CV_FOURCC_DEFAULT, fps, cvSize(640, 480));

      while(key != 27 /*escape key to quit*/)
      {
          //Query for the next frame
          frame = cvQueryFrame( capture );
          if( !frame ) break;

          cvWriteFrame(writer, frame); //Just like showing an image

          //Get the last key that's been pressed for input
          key = cvWaitKey( 1 );

      }
      cvReleaseVideoWriter(&writer); //Release the video file

      cvReleaseCapture( &capture ); // WARNING: DO NOT release the "frame" variable
      return 0;
}
```

Accessing a pixel

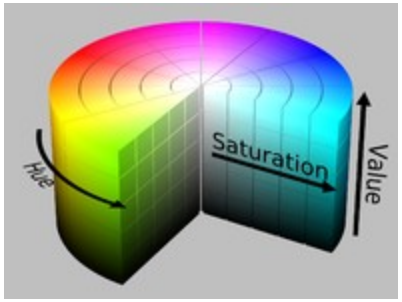
```
{
    int i = frame->height/2;
    int j = frame->width/2;

    //slow way
    CvScalar bgr = cvGet2D(frame, i, j);
    printf("B:%d, G:%d, R:%d\n", (int)bgr.val[0], (int)bgr.val[1], (int)bgr.val[2]);

    //fast way (access the array directly)
    printf("B:%d, G:%d, R:%d\n",
    ((uchar *)(frame->imageData + i*frame->widthStep))[j*frame->nChannels + 0], // B
    ((uchar *)(frame->imageData + i*frame->widthStep))[j*frame->nChannels + 1], // G
    ((uchar *)(frame->imageData + i*frame->widthStep))[j*frame->nChannels + 2]);// R

    //set the value
    cvSet2D(frame, i, j, CV_RGB(0,255,0));
    cvShowImage( "demowin2", frame );
}
```

Transforming color images



- **Grayscale**
  - 0.3*R+0.59*G+0.11*B, luminance

```
IplImage* grayscaleImg = cvCreateImage(cvSize(640,480), 8/*depth*/, 1/*channels*/);
{
     cvCvtColor(frame, grayscaleImg, CV_BGR2GRAY);
     cvShowImage( "demowin2", grayscaleImg );
}
```

- **HSV**
  - What we perceive as color, rather than “sense” as color (well… sort of)
    - Hue: the color value
    - Saturation: the richness of the color relative to brightness
    - Value: the gray level intensity

```
IplImage* singleChannelImg = cvCreateImage(cvSize(640,480), 8/*depth*/, 1/*channels*/);
IplImage* hsvImg = cvCreateImage(cvSize(640,480), 8/*depth*/, 3/*channels*/);
{
     cvCvtColor(frame, hsvImg, CV_BGR2HSV);
     cvSplit(hsvImg, singleChannelImg, NULL, NULL, NULL); // get the "Hue" component
     cvShowImage( "demowin3", singleChannelImg );
}
```

Reducing image size

For reducing image size quickly (e.g., to increase frame rate)

- **Pyramid**
  - nearest neighbor (no interpolation)
  - integer multiple down-sampling (throws rows and columns)
  - Anti-aliasing built in

```
IplImage* smallerImg = cvCreateImage( cvSize(640/2,480/2), 8/*depth*/, 3/*channels*/);
cvPyrDown( frame, smallerImg );
```

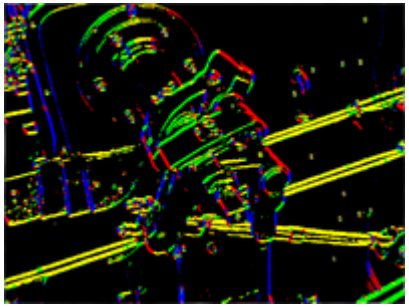
Image filtering

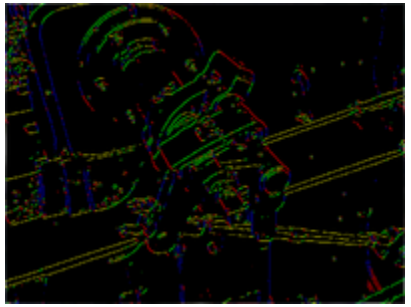
For example low pass filtering

```
cvSmooth(frame, frame, CV_GAUSSIAN, 15, 15); //size=15x15
```

CV_BLUR_NO_SCALE all ones (has ringing)
CV_BLUR all ones, sum to one (has ringing)
CV_GAUSSIAN a gaussian smoothing kernel (no ringing if large enough)
CV_MEDIAN choose the median of the block (non-linear)
CV_BILATERAL “perceptual closeness” (texture/color) of the pixels

Edge Detection

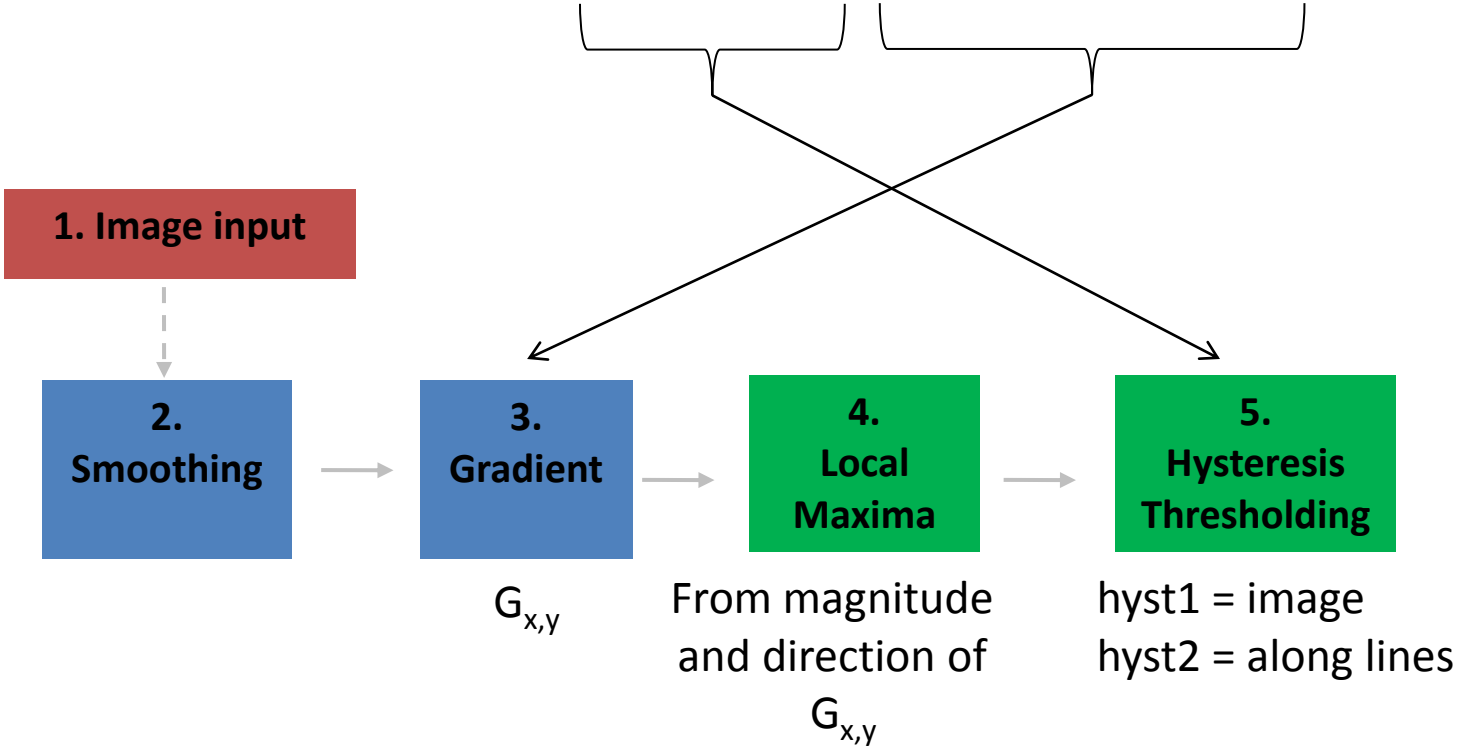




The Canny edge detector

```
//Requires a single-channel image:
cvCvtColor(frame, grayscaleImg, CV_BGR2GRAY);

//Reduce the image to the edges detected:
cvCanny(grayscaleImg, edgesImg, hyst1, hyst2, sobelSize, l2needed?);
```



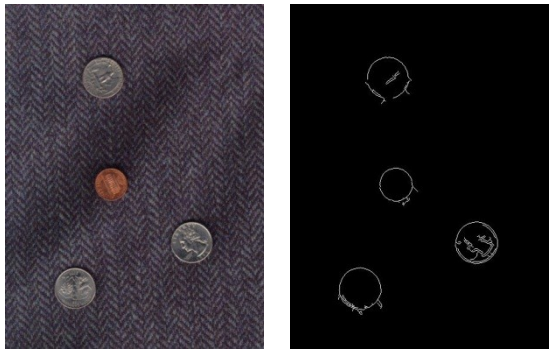
Hough Circles

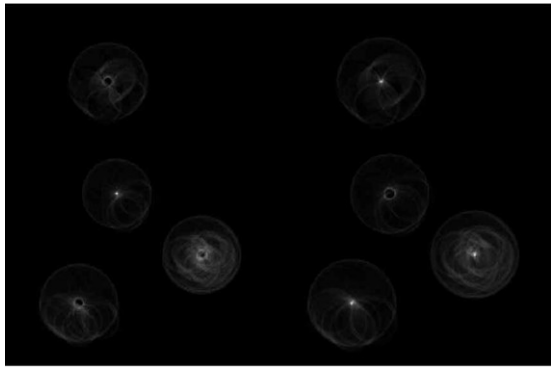
```
//Requires a single-channel image:
cvCvtColor(frame, grayscaleImg, CV_BGR2GRAY);
//Want to smooth it to get less noise
cvSmooth(grayscaleImg, grayscaleImg, CV_GAUSSIAN, 7, 9 );
//Detect the circles in the image
CvSeq* circles = cvHoughCircles(grayscaleImg, storage, CV_HOUGH_GRADIENT,
                  downsampling, minCenterDist, CannyThresh, AccumThresh
                  minRadius, maxRadius );
```

- **In General, Hough transform consists of**
  1. Edge detection
  2. Accumulation in parameter space

  For each detected point,
  draw shape with different parameter
  Look for local maxima

  For a circle space is (x,y,R)



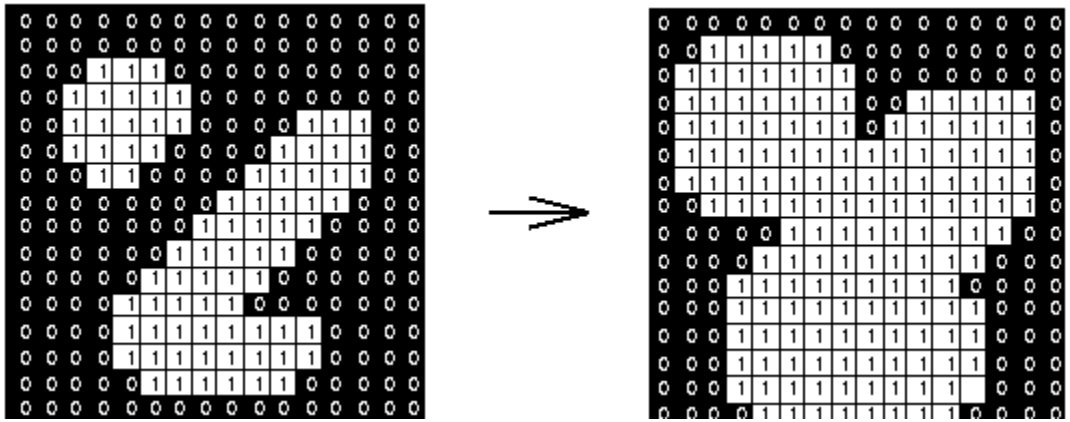


R=5 R=10

Morphology Primer

- Pick a structuring element
  - Cross
  - Disk
  - Rectangle

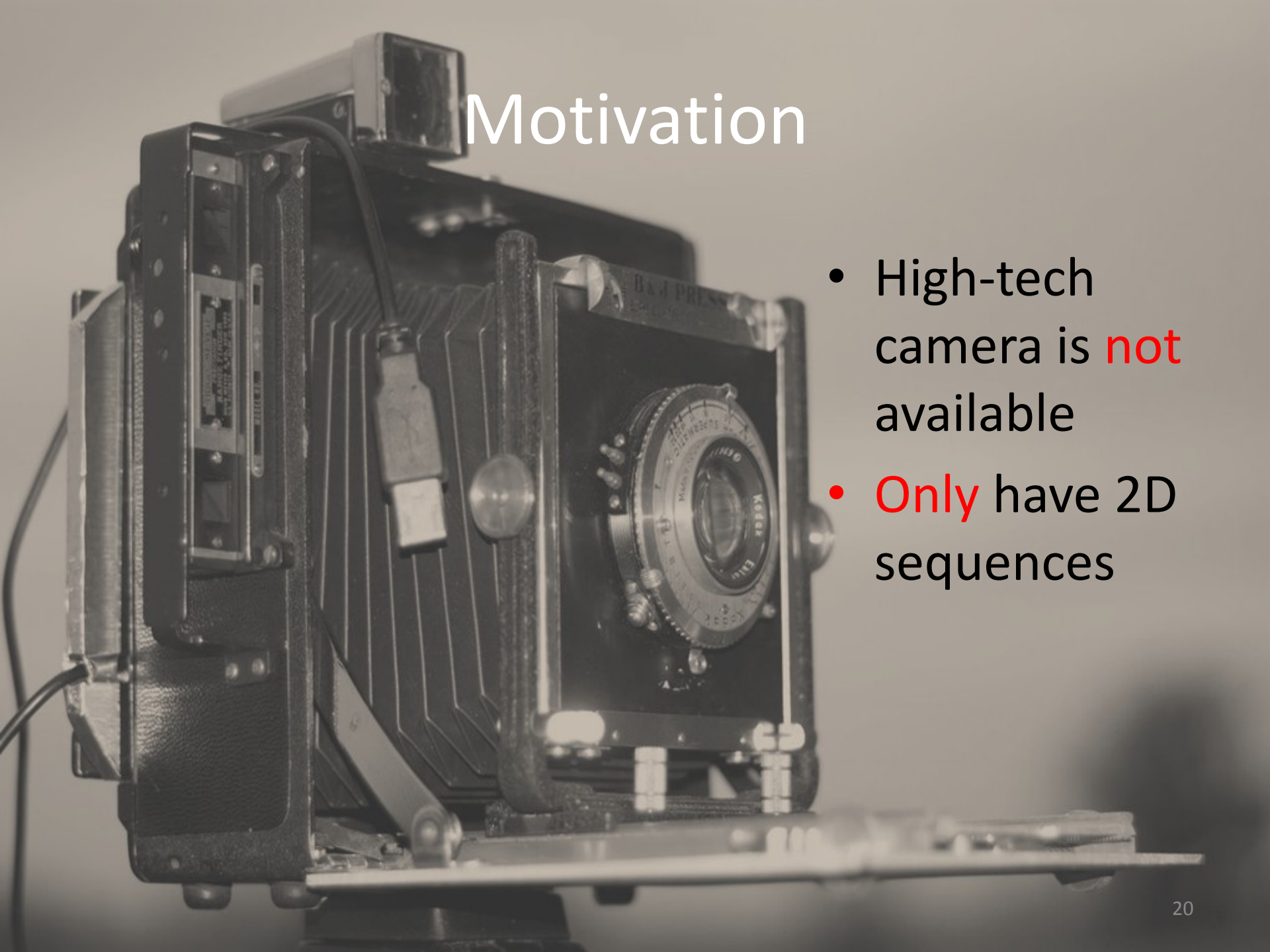
- Pick an operation
  - Dilation (Max, “OR”)
  - Erosion (Min, “AND”)
  - Opening (E+D)
  - Closing (D+E)




Torso Tracking
Ke-Yu Chen, 2012/02/15

Project goal

- Track the human **torso** in real time from a **single** general-purpose camera



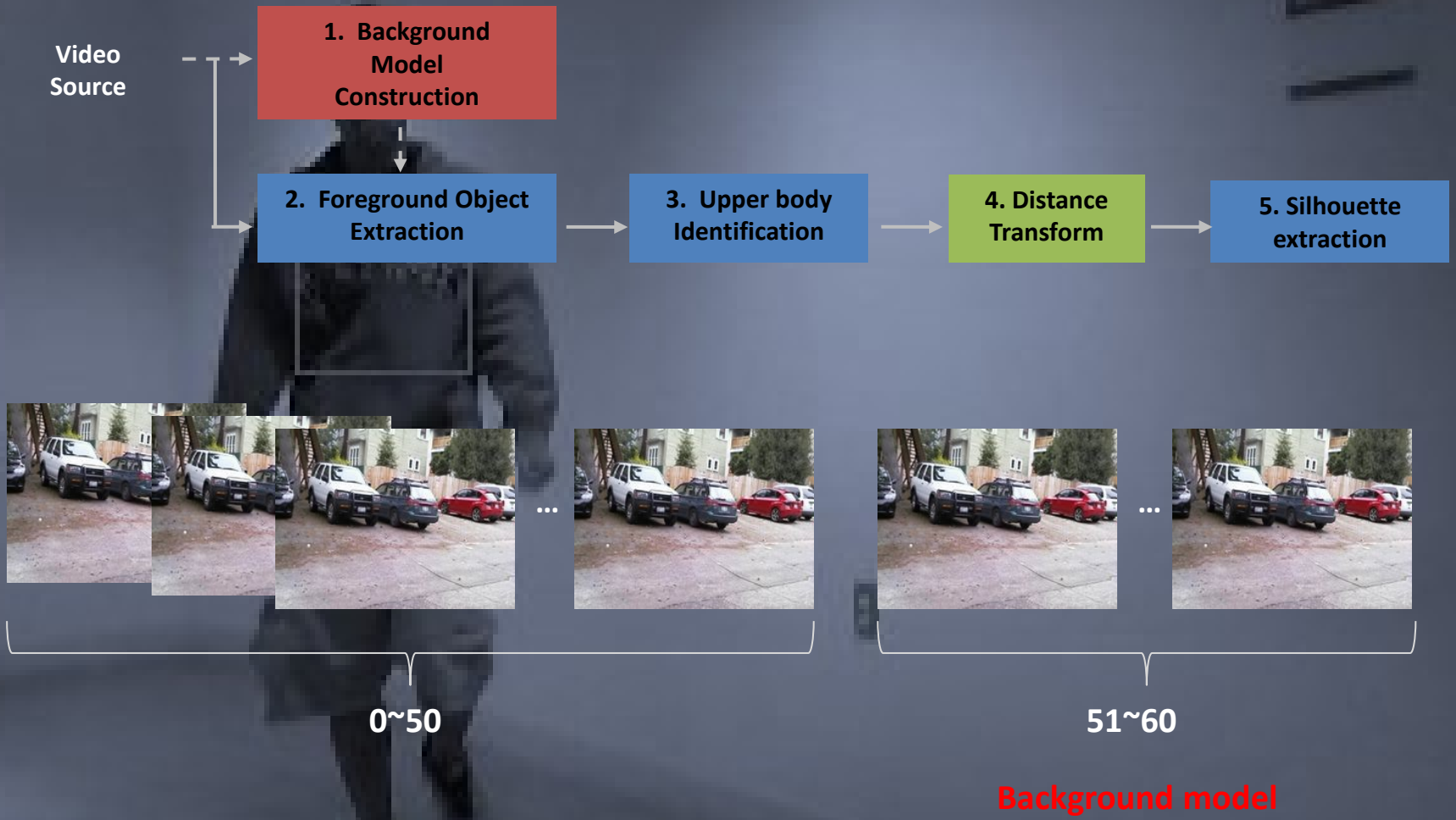
Motivation

- High-tech camera is not available
- Only have 2D sequences

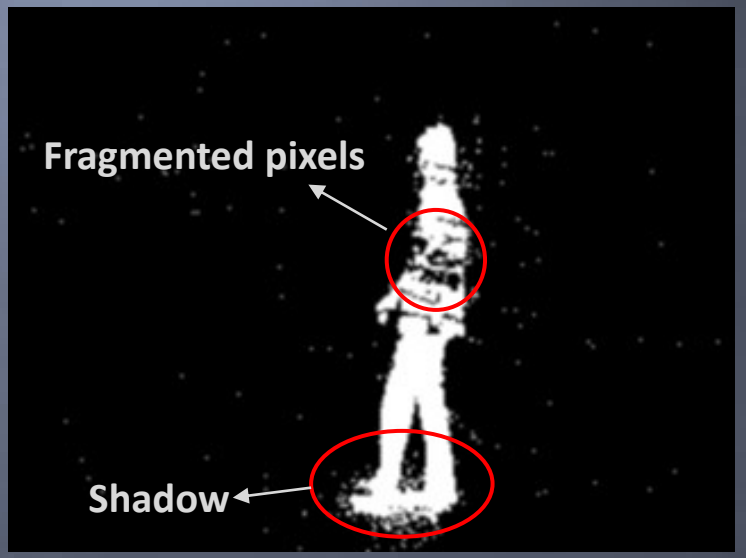
Algorithm


Video Source
1. Background Model Construction
2. Foreground Object Extraction
3. Upper body Identification
4. Distance Transform
5. Silhouette extraction

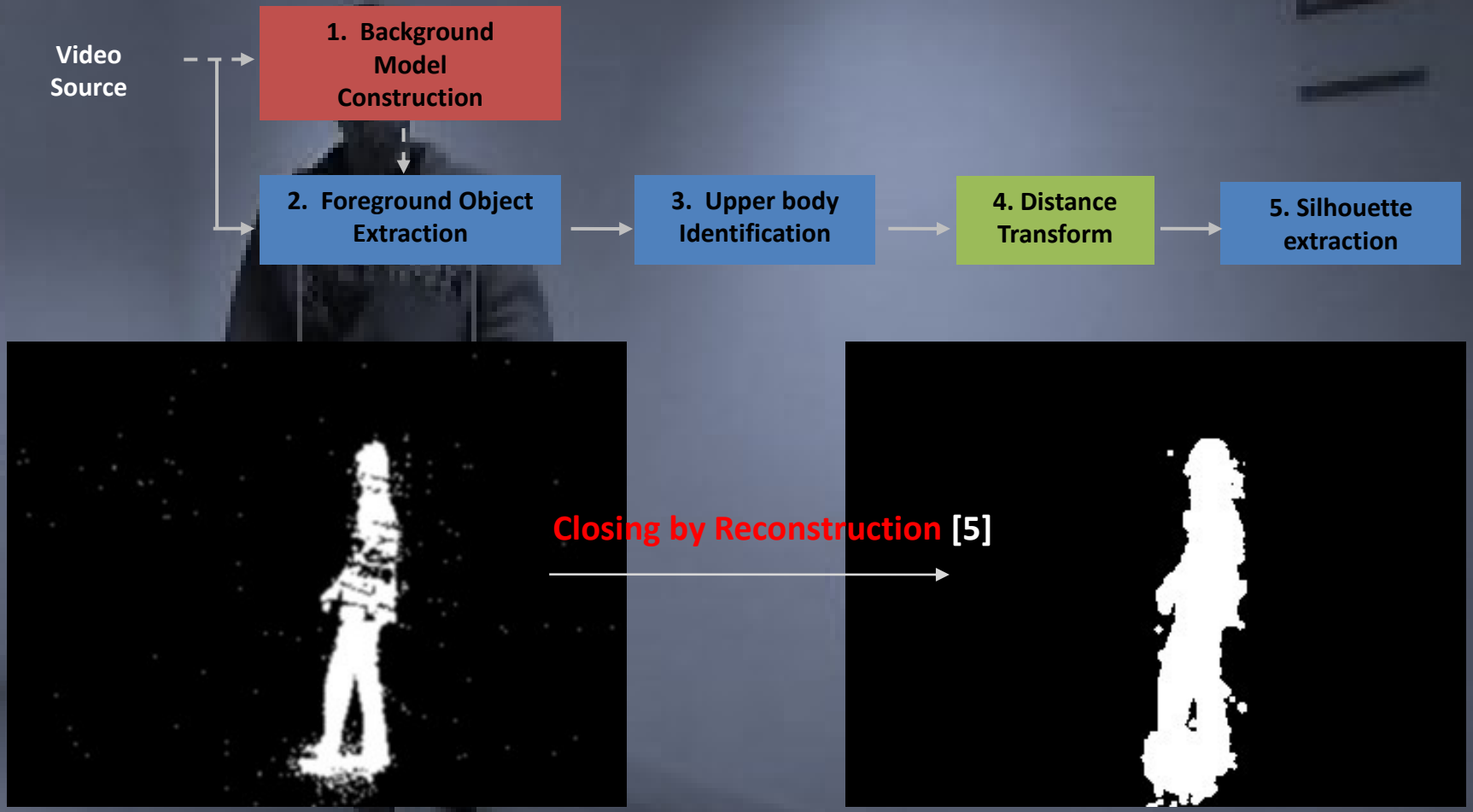
1. Background Model Construction



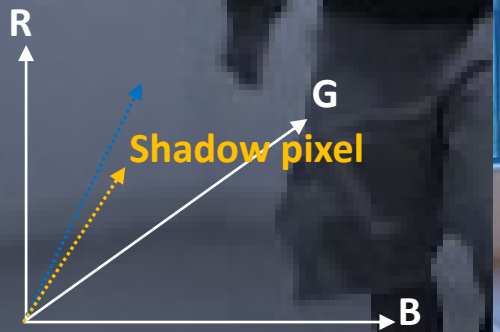
2. Foreground Object Extraction

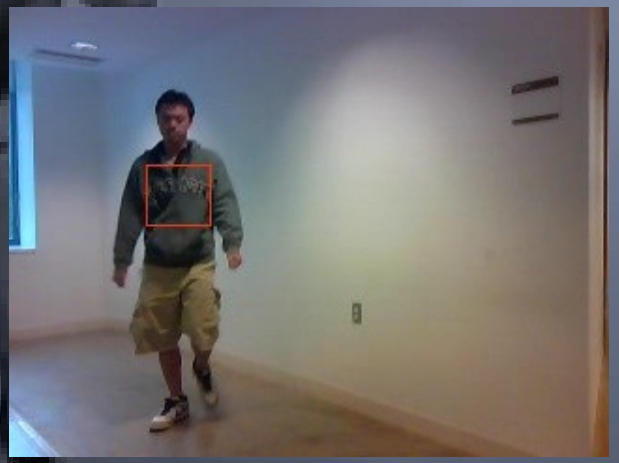


2-1. Fill the fragmented pixles

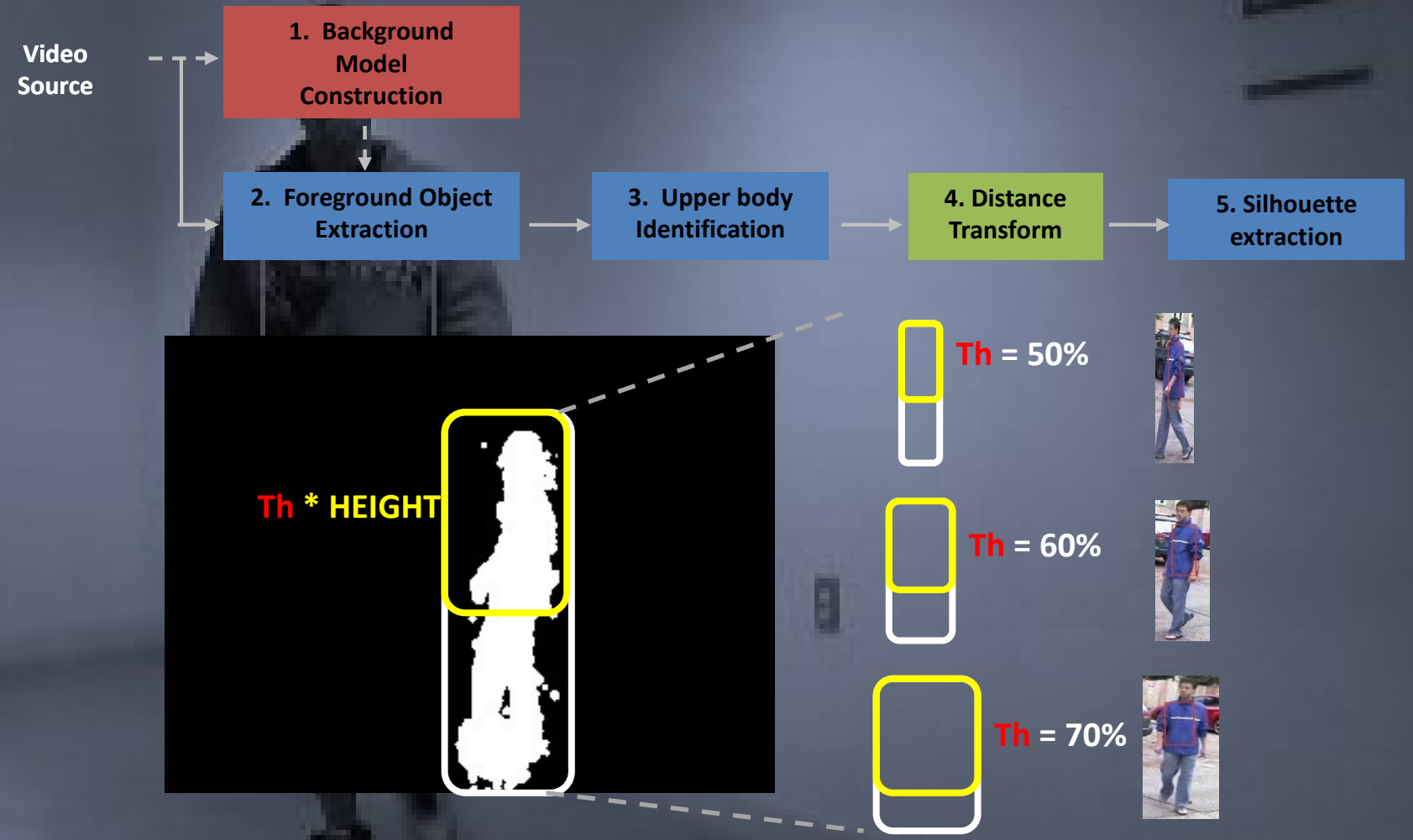


2-2. Shadow detection [4]





3. Upper body identification



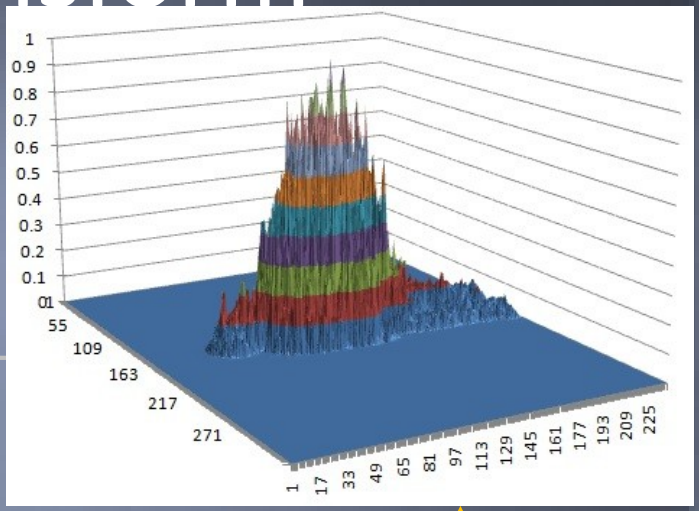
4. Distance Transform

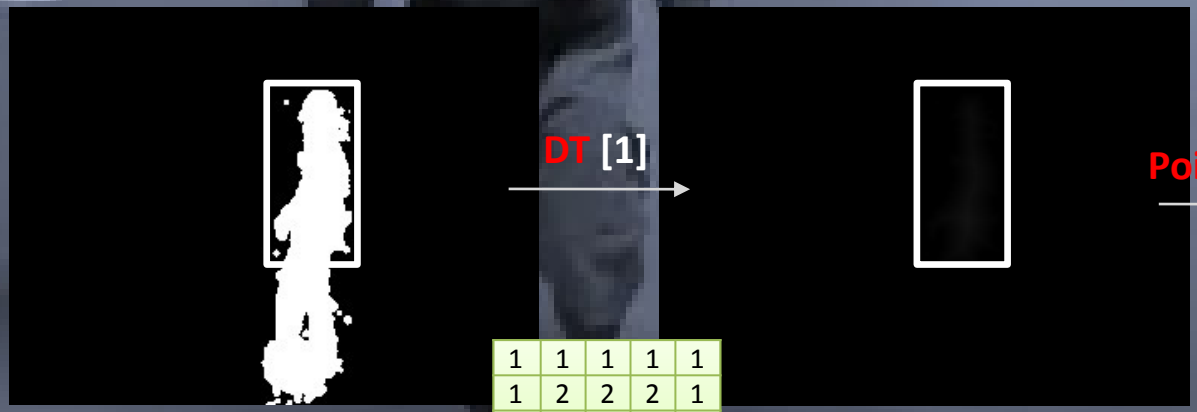
Video Source

1. Background Model Construction

2. Foreground Object Extraction

3. Upper body Identification





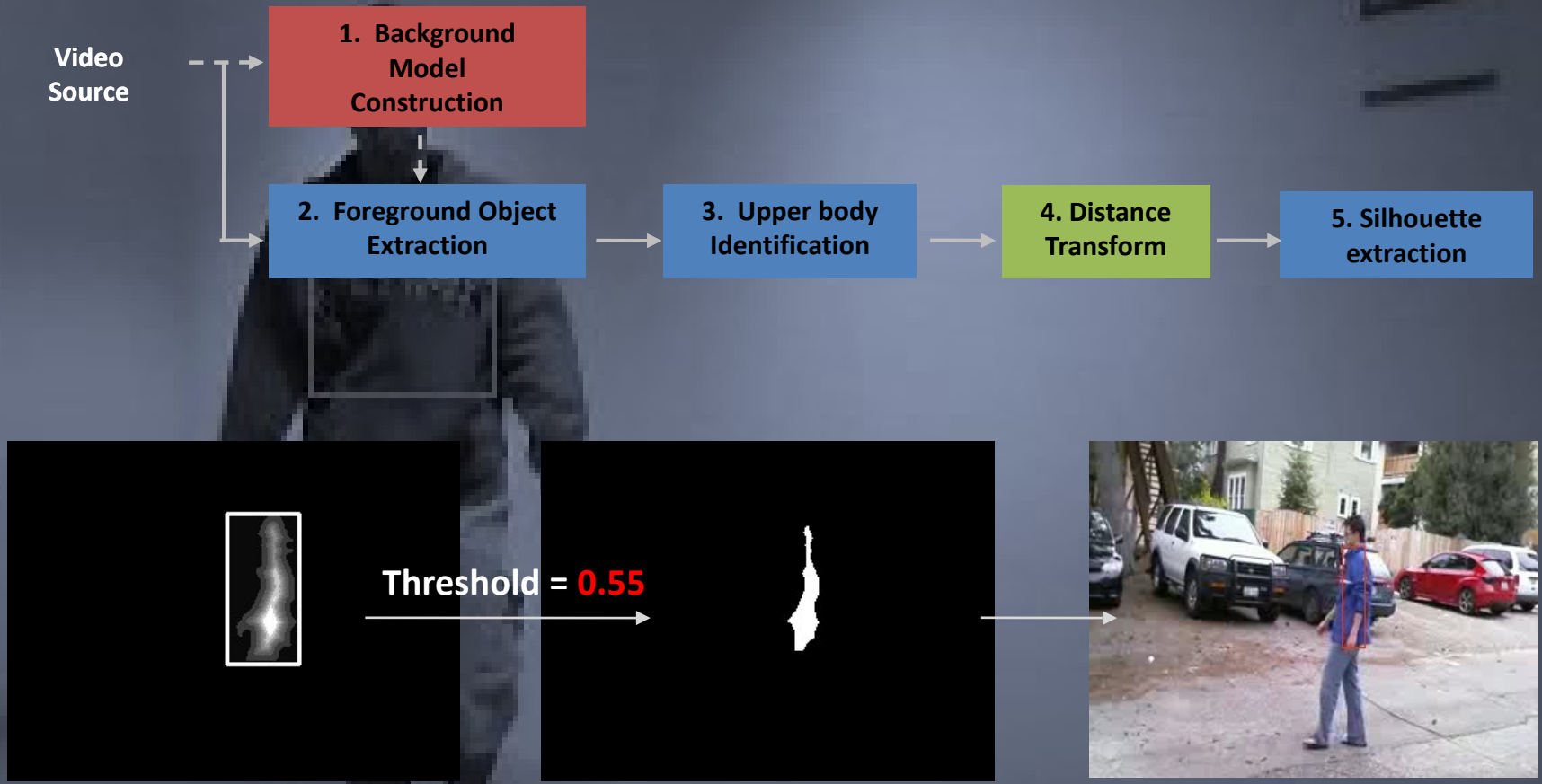
DT [1]

1	1	1	1	1
1	2	2	2	1
1	2	3	2	1
1	2	2	2	1
1	1	1	1	1

Poisson [3]

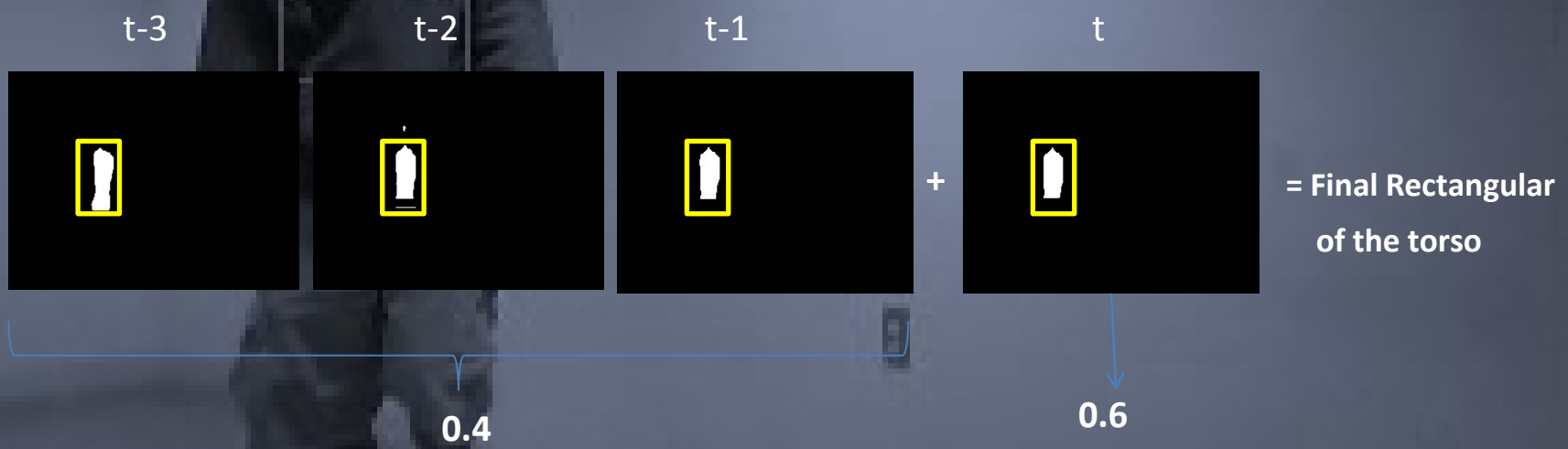
$$w(x, y) = 1 + \frac{1}{4} * \left(w(x+1, y) + w(x, y+1) + w(x-1, y) + w(x, y-1) \right)$$

5. Silhouette Extraction



Smooth the result

- Improve shaking results



Demo

http://youtu.be/DWy3ac9HoKY

Reference

1. Distance transform: http://en.wikipedia.org/wiki/Distance_transform

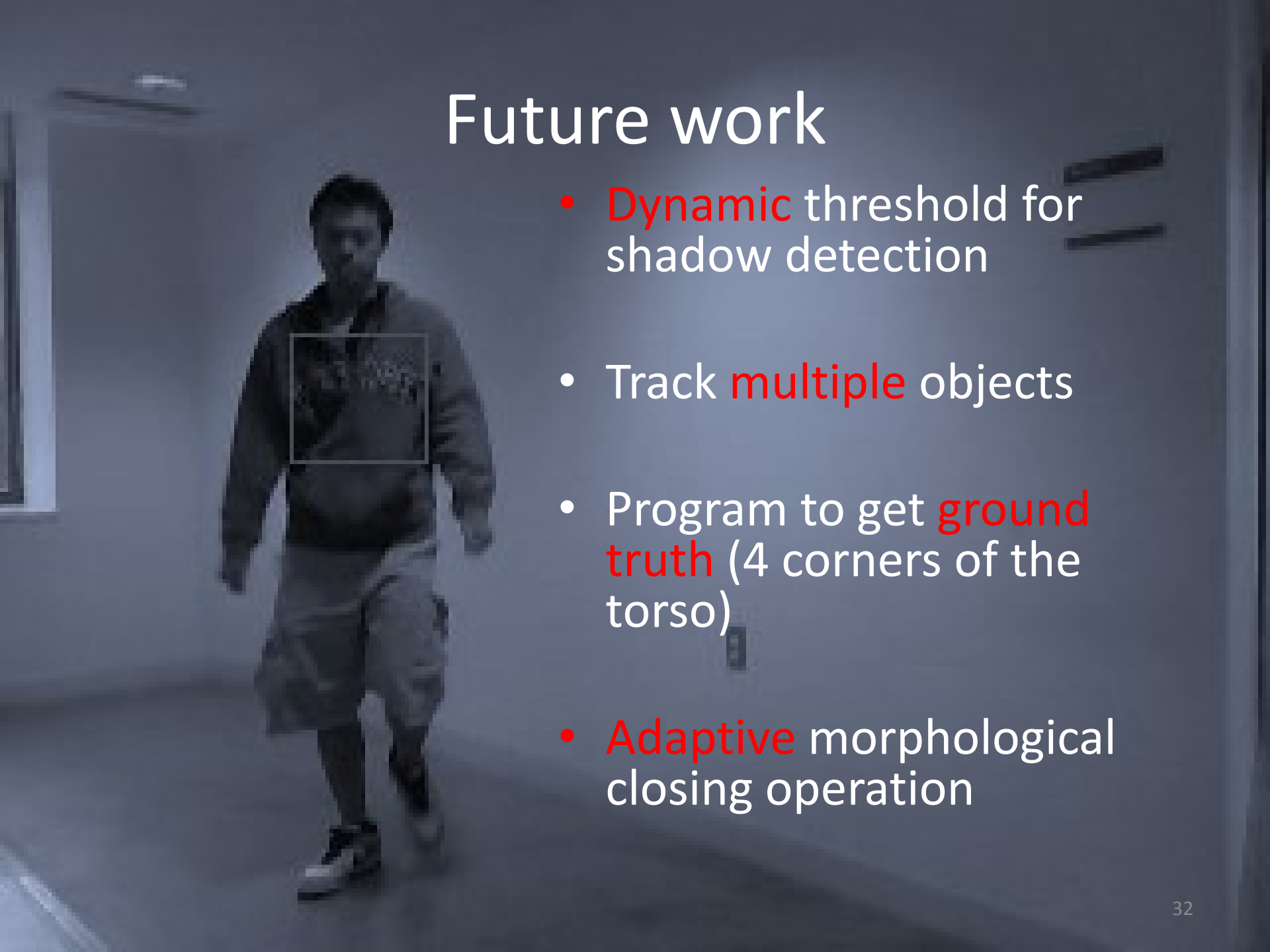
2. Siddiqi, K.; Member, S. & Kimia, B. B. Parts of Visual Form: Computational Aspects IEEE Transactions on Pattern Analysis and Machine Intelligence, 1995, 17, 239-251

3. Gorelick, L.; Galun, M.; Sharon, E.; Basri, R.; Society, I. C.; Society, I. C. & Br, A. Shape representation and classification using the poisson equation In In Proc. of CVPR'04, 2004, 61-67

4. Horprasert, T.; Harwood, D. & Davis, L. S. A statistical approach for real-time robust background subtraction and shadow detection 1999, 1-1

5. Closing by Reconstruction: http://artis.imag.fr/Members/Adrien.Bousseau/morphology/morphomath.pdf

Future work



- Dynamic threshold for shadow detection
- Track multiple objects
- Program to get ground truth (4 corners of the torso)
- Adaptive morphological closing operation

Questions?