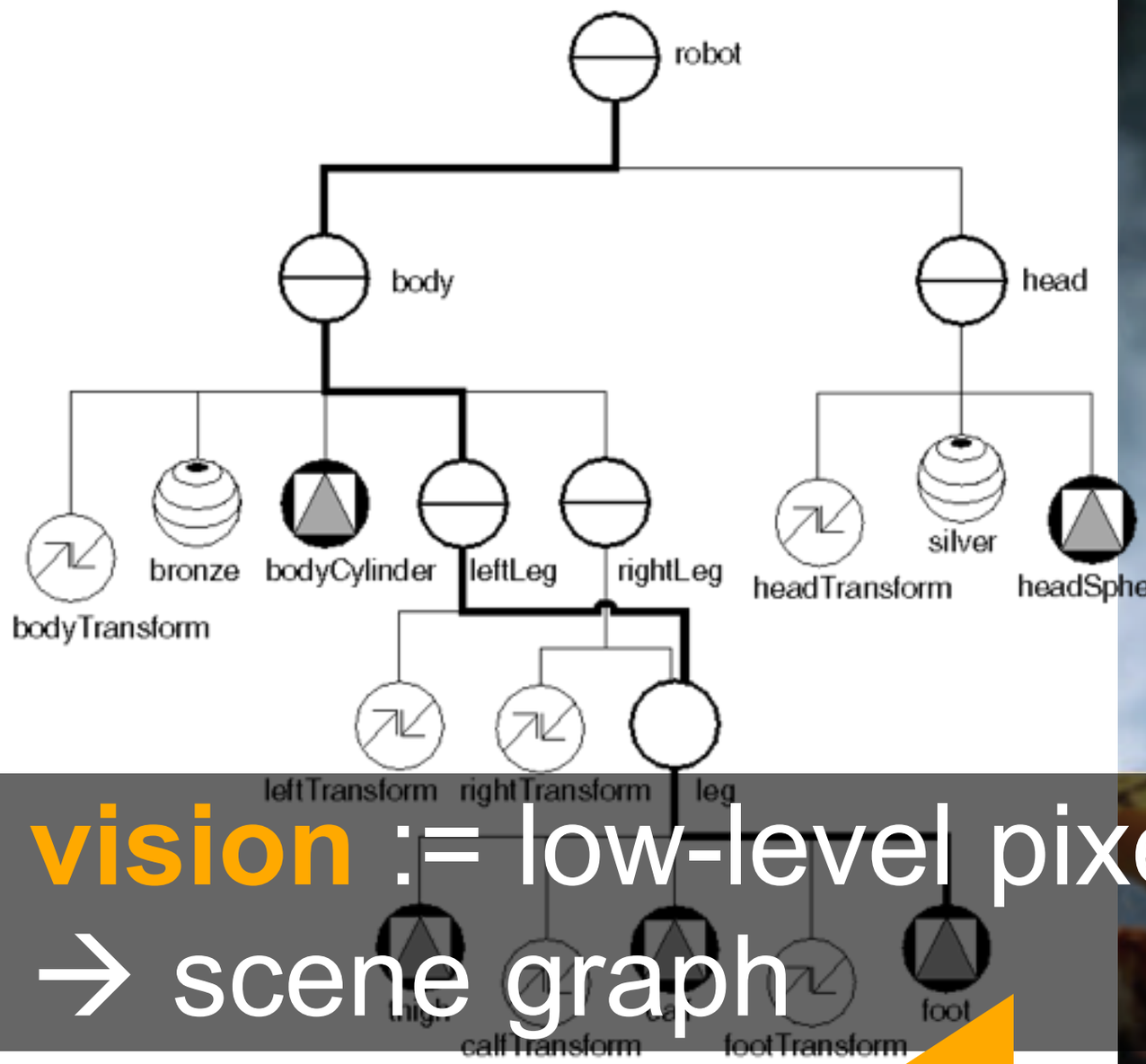
practical introduction to

(open) computer vision

hci 2 prototyping :: hasso plattner institute
christian holz



vision := low-level pixels

→ scene graph

today's about **image segmentation**

30sec brainstorming

30sec brainstorming

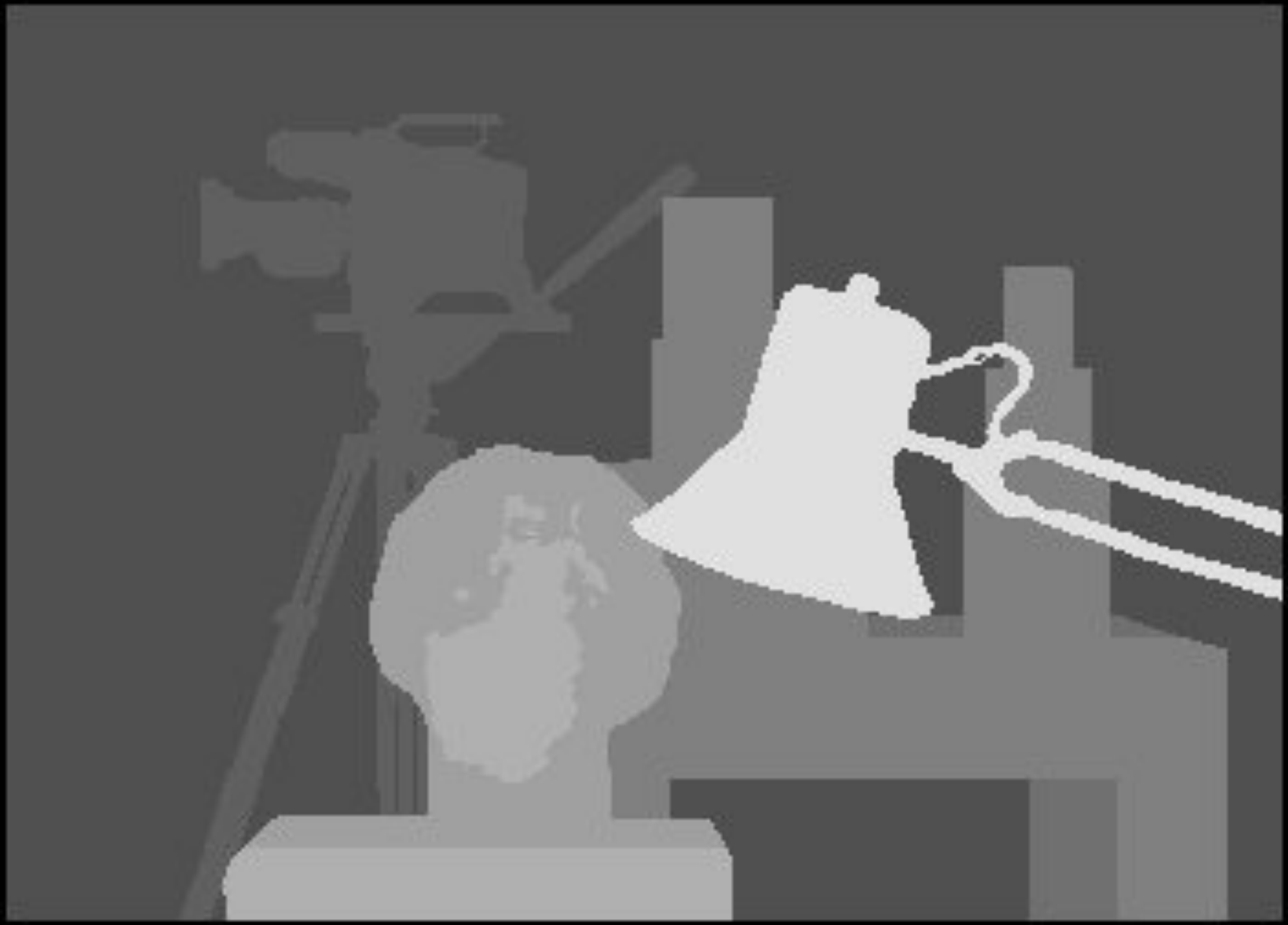
how to count all cars? how to ignore the rest?

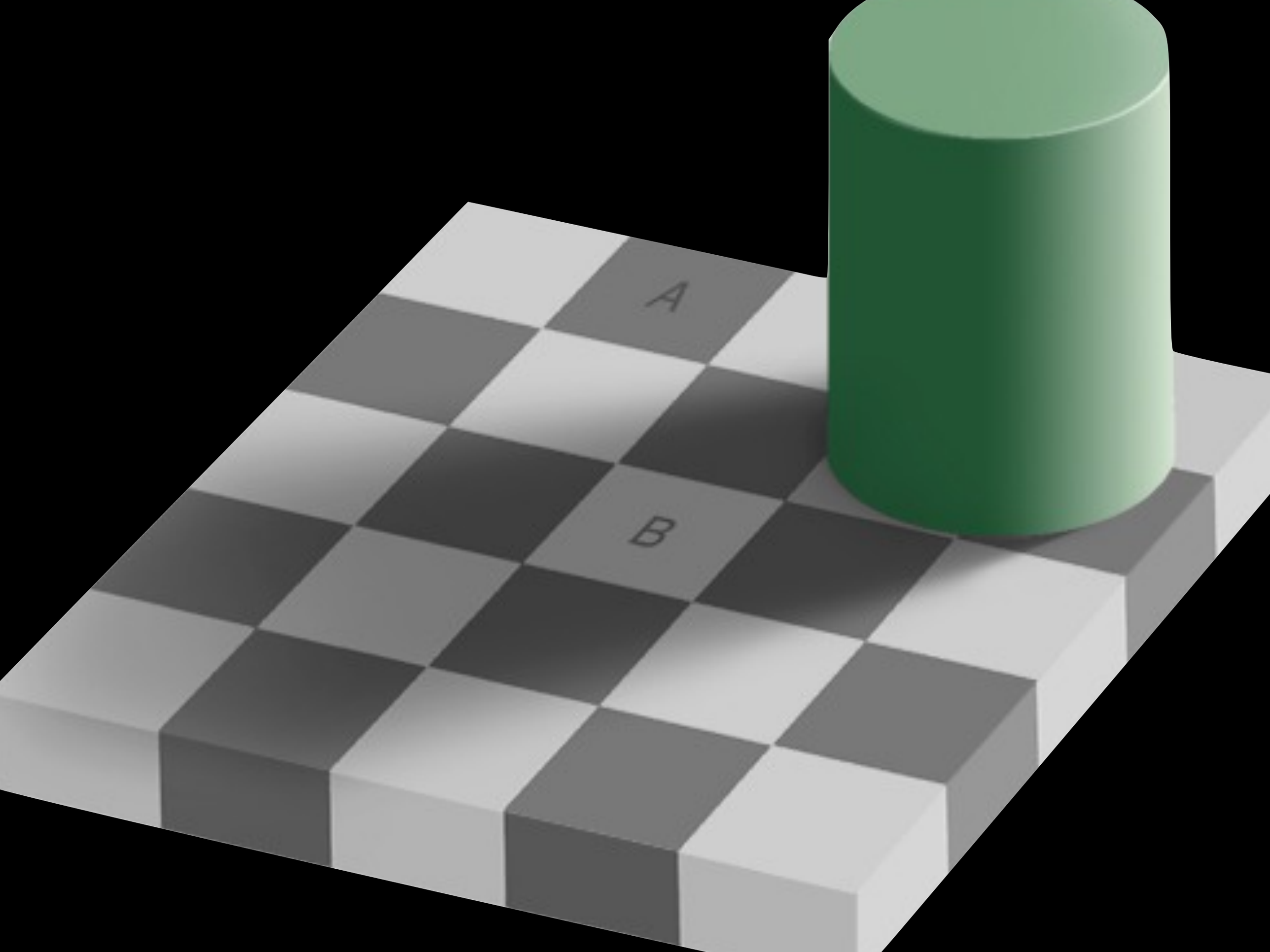

NYC
UBS
9M97

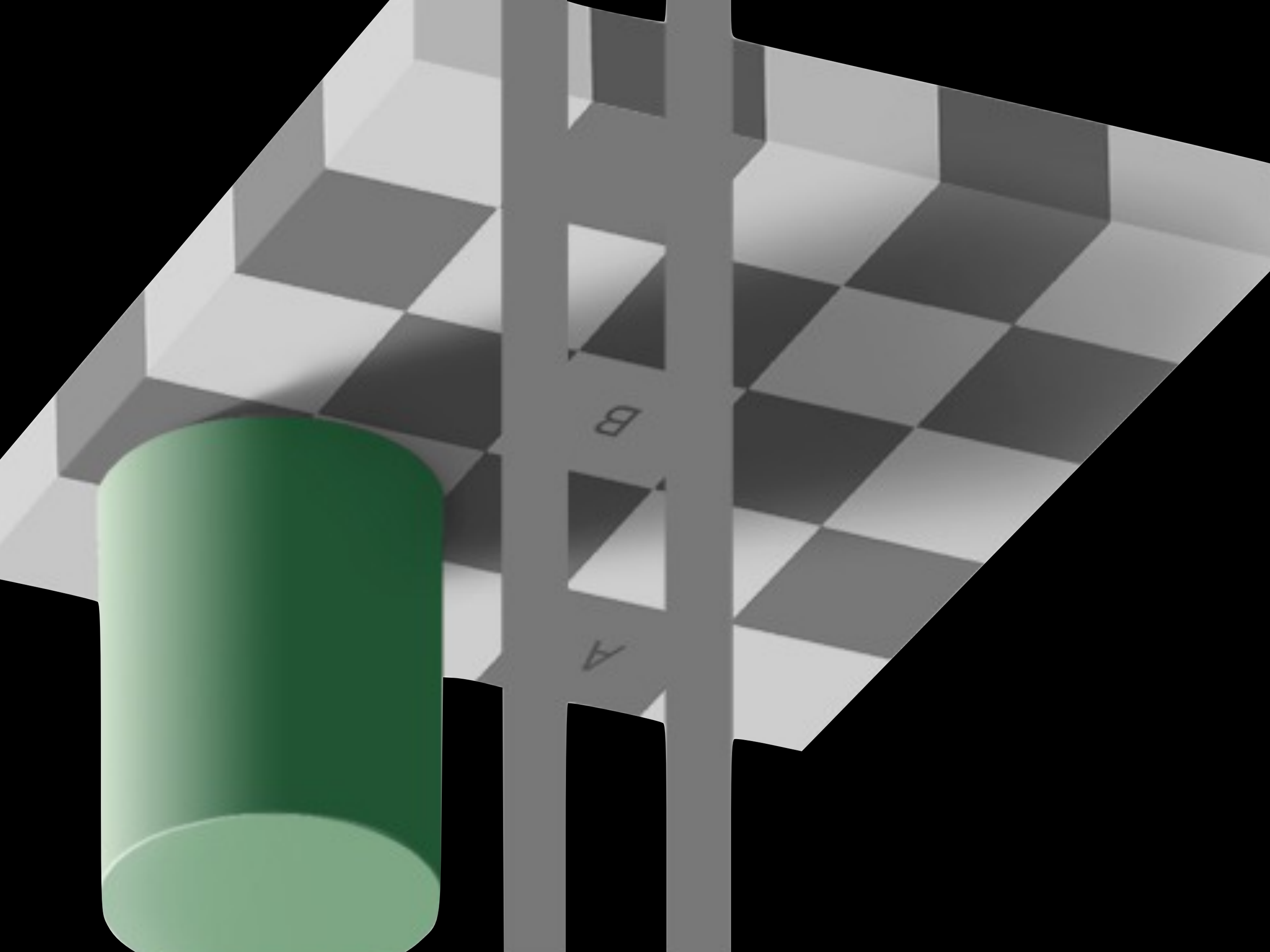
30sec brainstorming

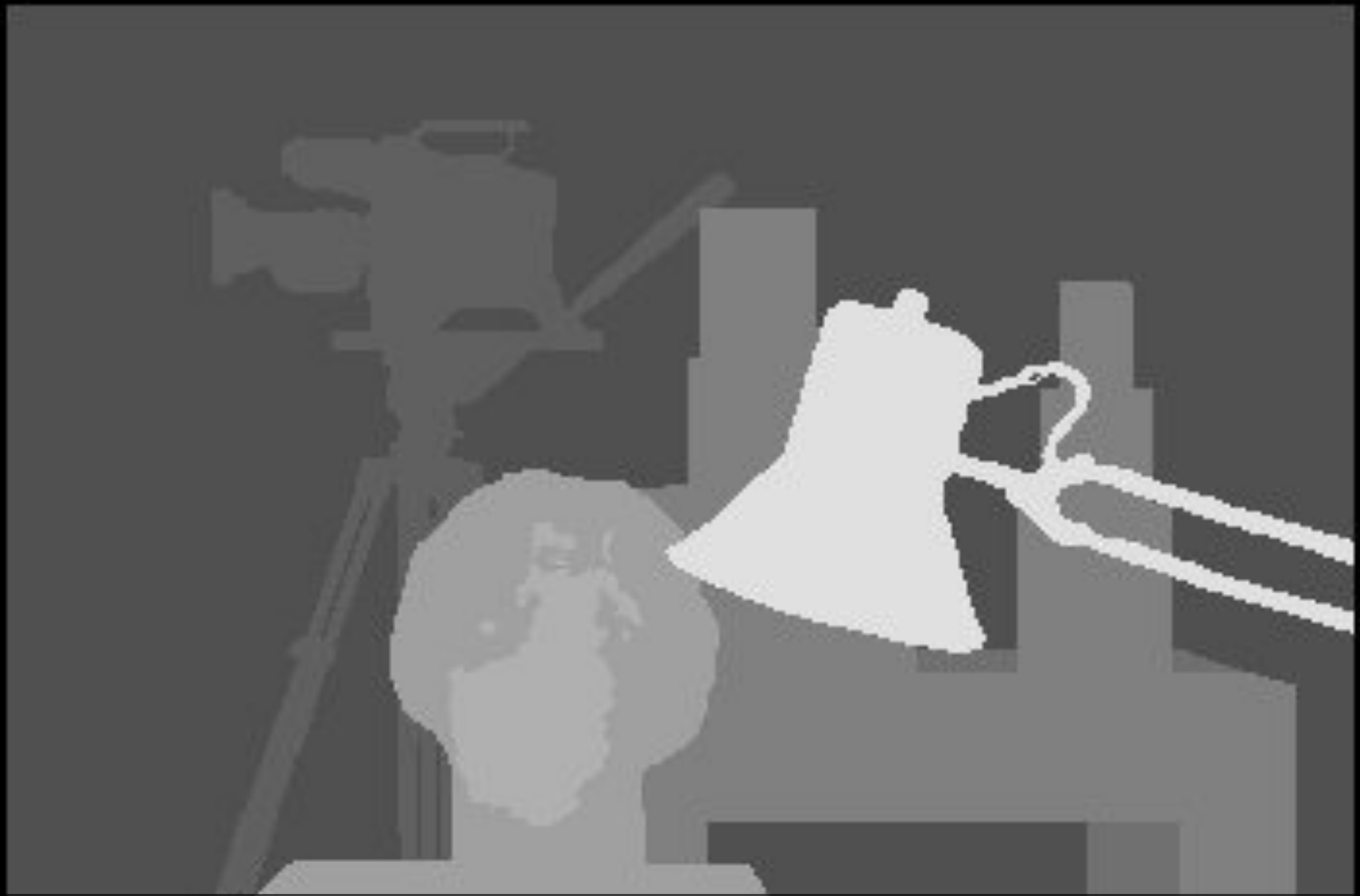
how to deal with occlusion? illumination?


Using Two - Level




A
B


B
A



detecting components = easy

assignment

assignment

goal = drawing

goal = drawing

shoes in contact with floor

shoes in contact with floor

30sec brainstorming

(top-down camera setup)

“thresholding”

turn all values < X
into black

–

=

remove everything
farther than X

“thresholding”

OpenCV

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

#include <ctype.h>
#include <iostream>

using namespace cv;
using namespace std;

int main( int argc, char** argv )
{
    VideoCapture cap;
    cap.open("video.avi");

    if( !cap.isOpened() ) {
        cout << "Could not initialize capturing...\n";
        return -1;
    }

    namedWindow( "output", 0 );

    for(;;) {
        Mat frame;
        cap >> frame;
        if( frame.empty() ) {
            break;
        }

        //
        // do something with frame
        //
        //...

        imshow( "output", frame );
```

scaffold provided

Quick search

Go

Enter search terms or a module, class or function name.

OpenCV 2.1 C Reference

The OpenCV Wiki is here: http://opencv.willowgarage.com/

Contents:

http://opencv.willowgarage.com/documentation/cpp/

OpenCV reference (C++)

Table Of Contents

Next topic

cxcore. The Core Functionality

This Page

Show Source

Quick search

Go

Enter search terms or a module, class or function name.

OpenCV 2.1 C Reference

The OpenCV Wiki is here: http://opencv.willowgarage.com/

Contents:

http://opencv.willowgarage.com/documentation/cpp/

OpenCV reference (C++)

or search for documentation online, e.g. book Learning OpenCV

```
cv::Mat img1 = cv::Mat( frame.size(), CV_8UC1  );
cv::Mat img2 = cv::Mat( frame.size(), CV_8UC1  );
cv::Mat img3 = cv::Mat( frame.size(), CV_32S   );
cv::Mat img4 = cv::Mat( frame.size(), CV_32FC2 );
```

creating images

```
cv::Mat img1 = cv::Mat( frame.size(), CV_8UC1  );
cv::Mat img2 = cv::Mat( frame.size(), CV_8UC1  );
cv::Mat img3 = cv::Mat( frame.size(), CV_32S   );
cv::Mat img4 = cv::Mat( frame.size(), CV_32FC2 );
```

creating images

```
cv::Mat img1 = cv::Mat( frame.size(), CV_8UC1  );
cv::Mat img2 = cv::Mat( frame.size(), CV_8UC1  );
cv::Mat img3 = cv::Mat( frame.size(), CV_32S   );
cv::Mat img4 = cv::Mat( frame.size(), CV_32FC2 );
```

CV_<bit depth><S/U/F>C<channels>

creating images

```
cv::Mat img1 = cv::Mat( frame.size(), CV_8UC1  );
cv::Mat img2 = cv::Mat( frame.size(), CV_8UC1  );
cv::Mat img3 = cv::Mat( frame.size(), CV_32S   );
cv::Mat img4 = cv::Mat( frame.size(), CV_32FC2 );
```

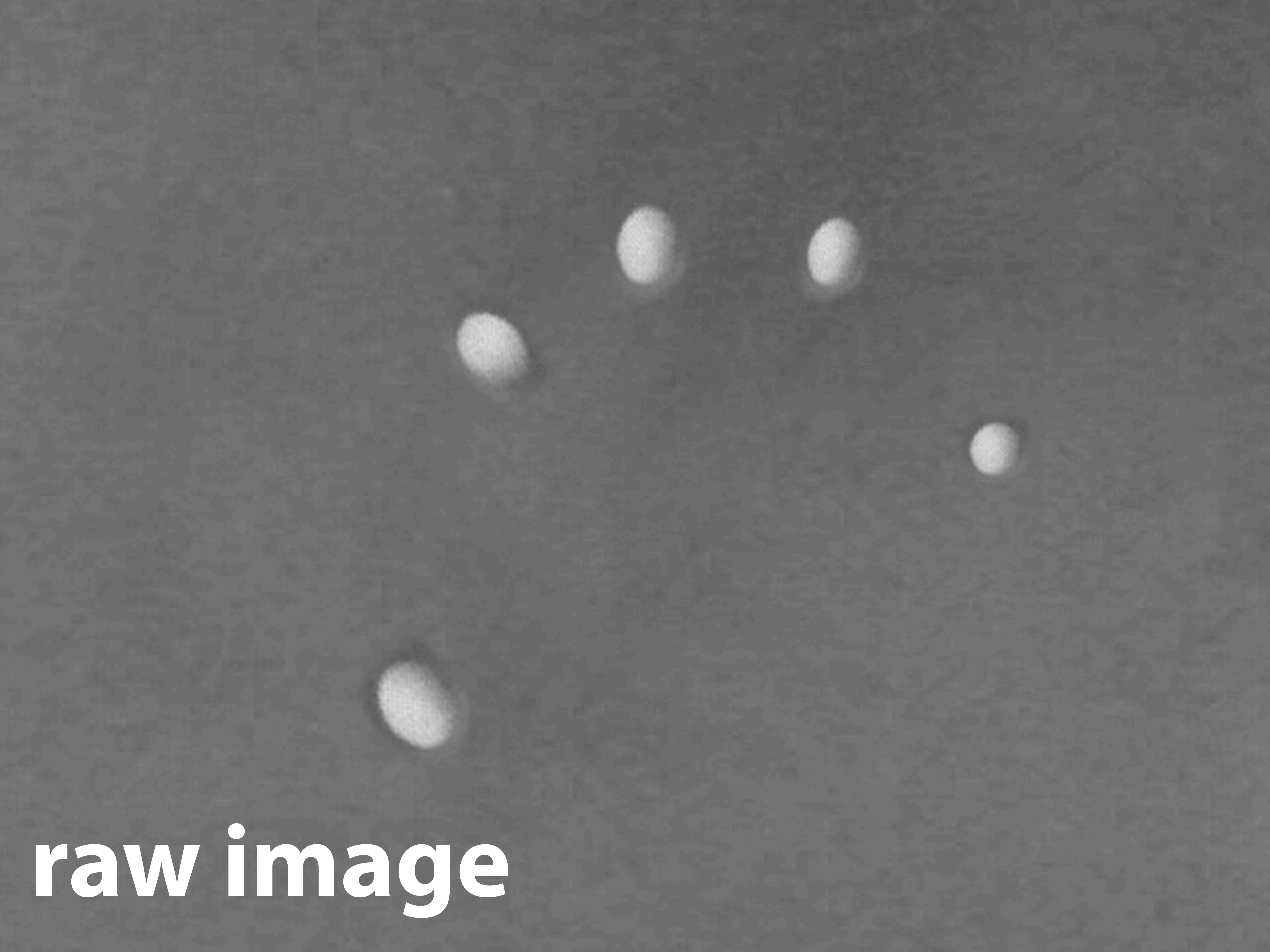
```
img1.copyTo( img2 );

cv::Mat img5 = img1.clone();
```

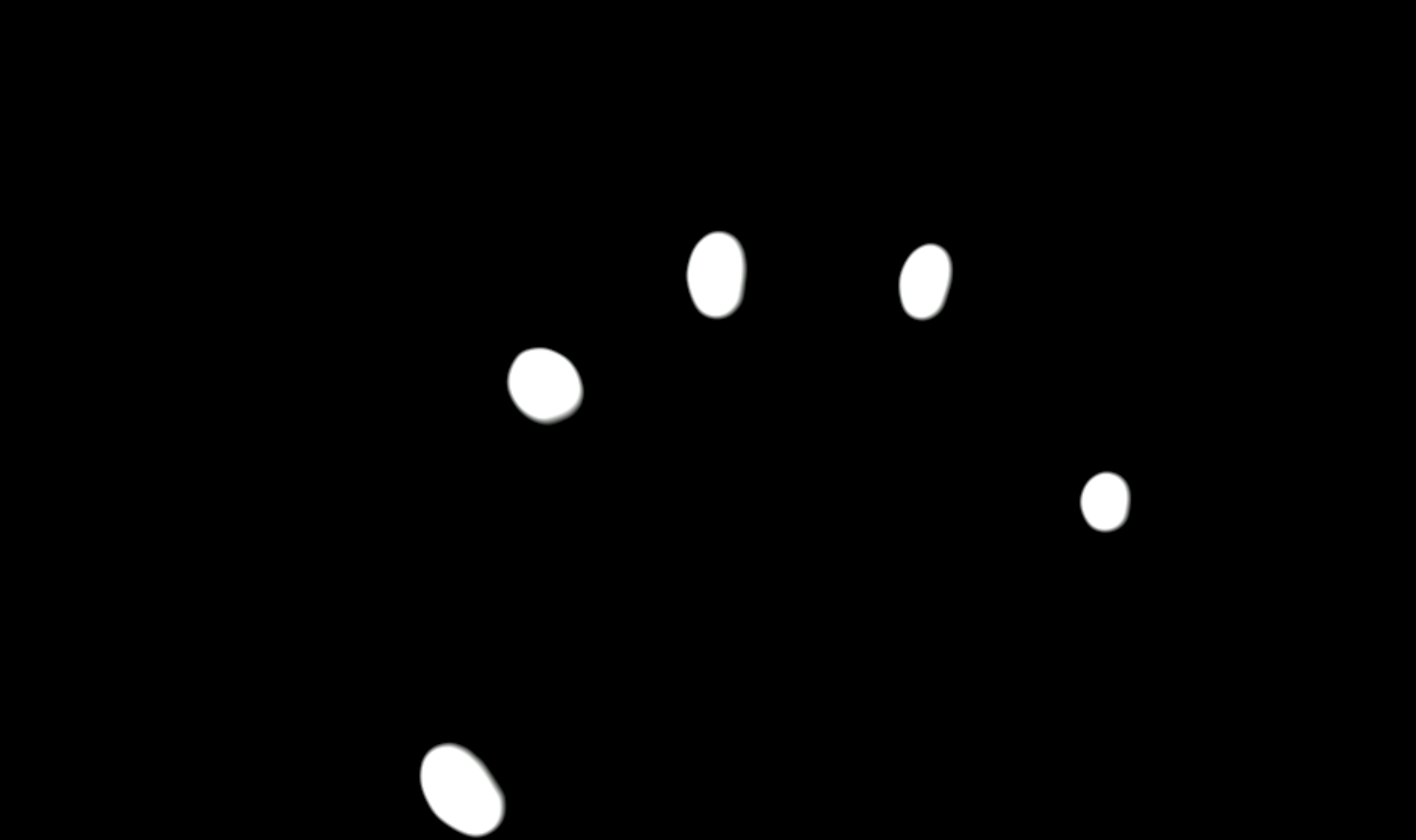
creating images

```
        cv::imshow( "output", frame );

        int c = cv::waitKey( 30 );
        if( c == 'q' ) {
            break;
        }
```

showing images

tabletop systems

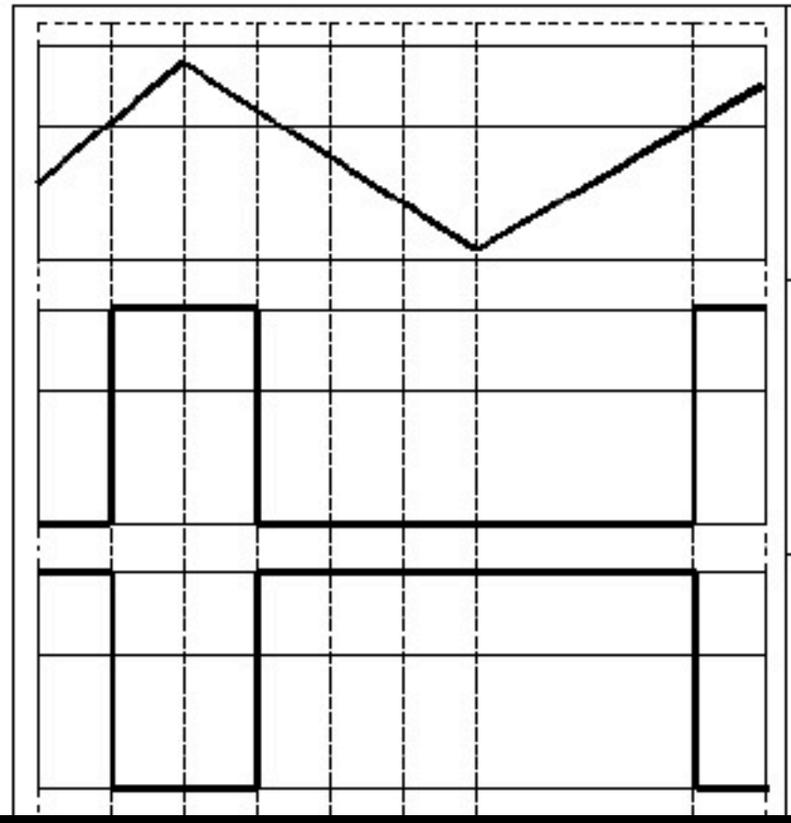

raw image

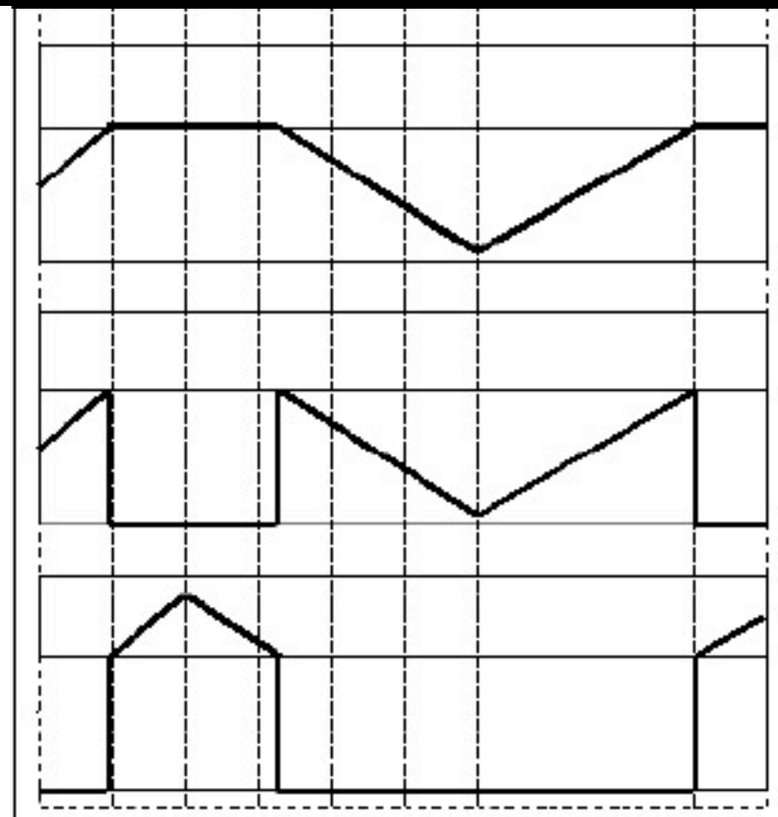


threshold

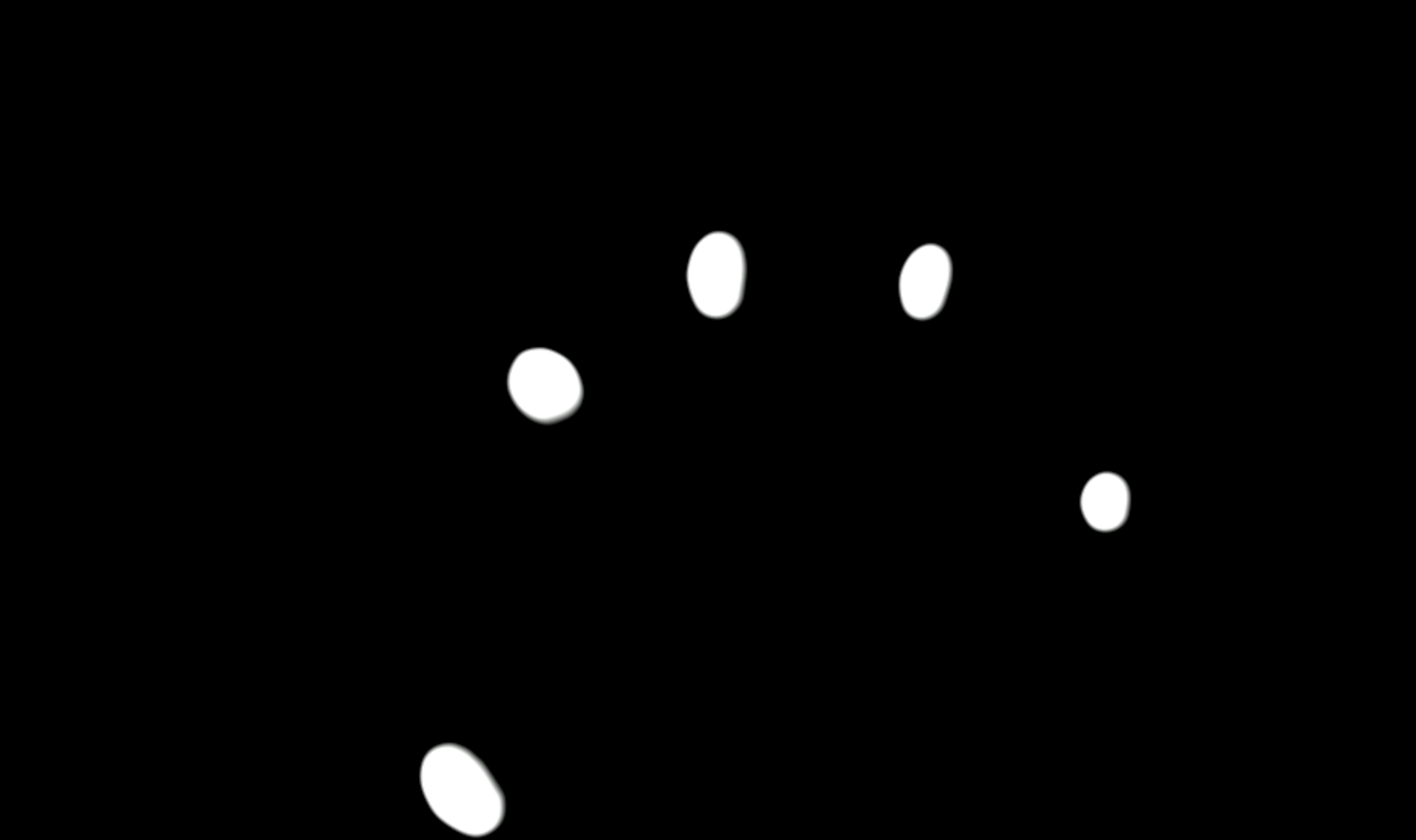
```
double cv::threshold(const cv::Mat& src,
                     cv::Mat& dst,
                     double thresh,
                     double maxVal,
                     int thresholdType);
```

threshold

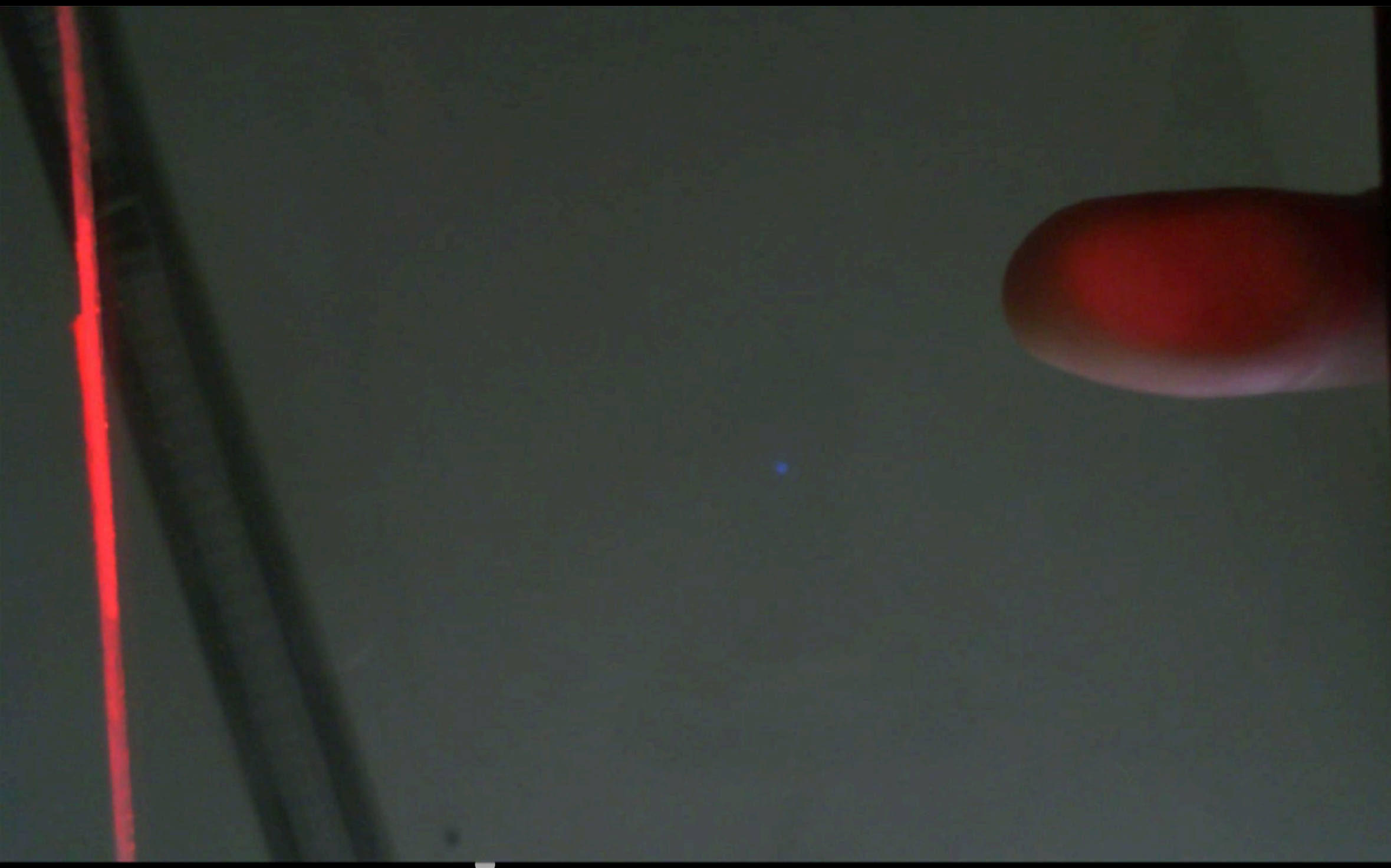

Value and Threshold Level
Threshold Binary
Threshold Binary, Inverted


Truncate
Threshold to Zero, Inverted
Threshold to Zero

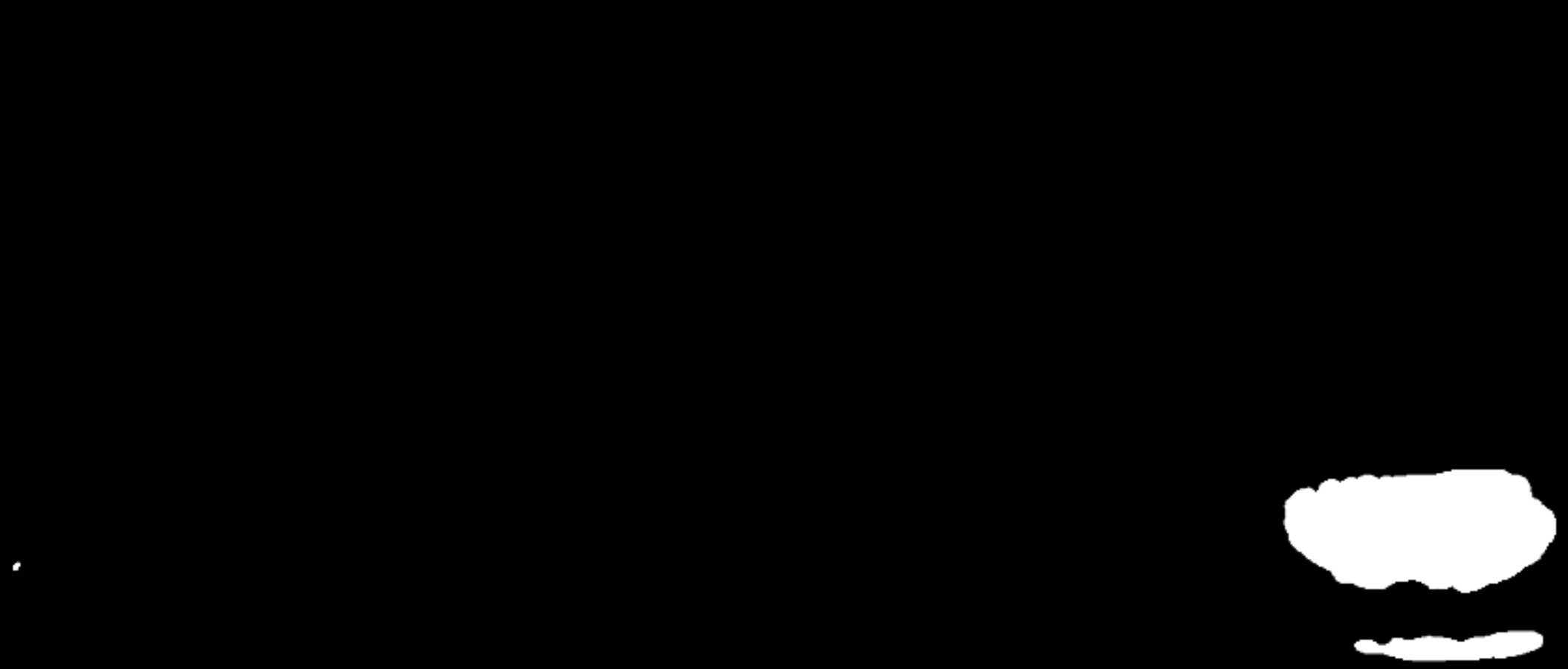
threshold



threshold



camera image



threshold

```
void cv::findContours(
            const cv::Mat& image,
            std::vector<std::vector<cv::Point> >& contours,
            int mode,
            int method,
            cv::Point offset = cv::Point());
```

find contours

```
void cv::findContours(
            const cv::Mat& image,
            std::vector<std::vector<cv::Point> >& contours,
            CV_RETR_EXTERNAL,
            CV_CHAIN_APPROX_NONE,
            cv::Point offset = cv::Point());
```

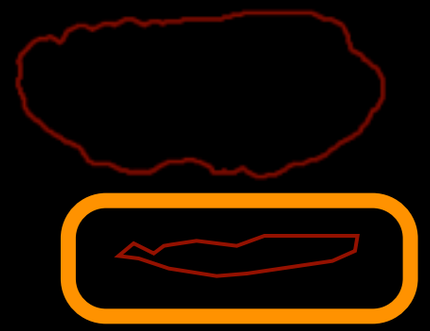
find contours

```
void cv::findContours(
            const cv::Mat& image,
            std::vector<std::vector<cv::Point> >& contours,
            CV_RETR_EXTERNAL,
            CV_CHAIN_APPROX_NONE,
            cv::Point offset = cv::Point());
```

// modifies image

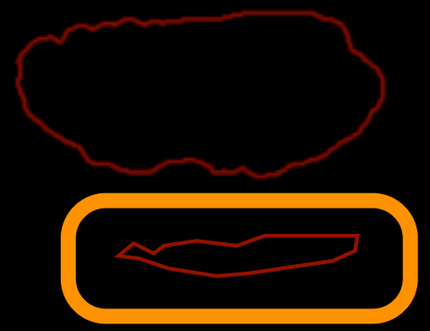
find contours

find contours



what about these?

find contours



what about these?

filter by **size, aspect, ...**

find contours

```
double cv::contourArea(const cv::Mat& contour);

// cv::Mat offers a CTOR for std::vector<cv::Point>
```

filtered contours

turn all values < X
into black

–

=

remove everything
farther than X

“thresholding”

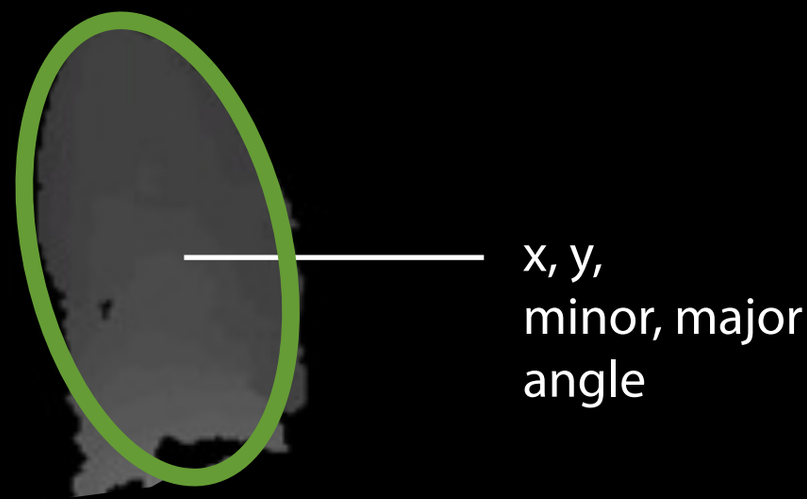
turn all values < X
into black

–

=

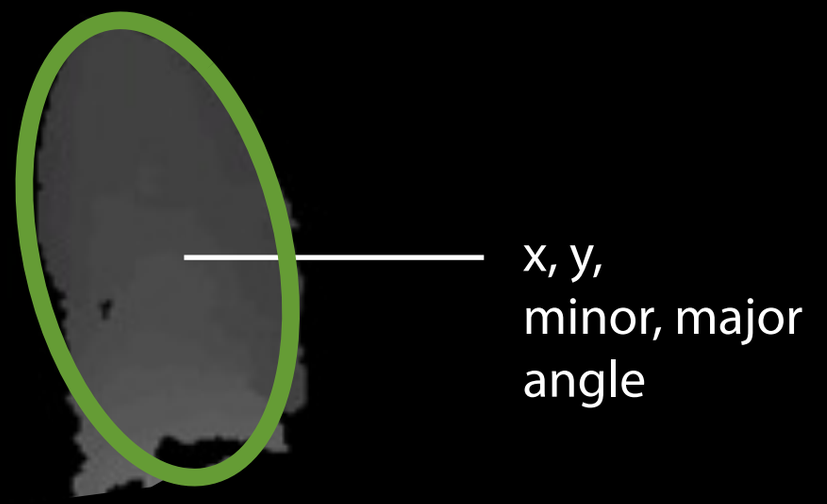
remove everything
farther than X

“thresholding”



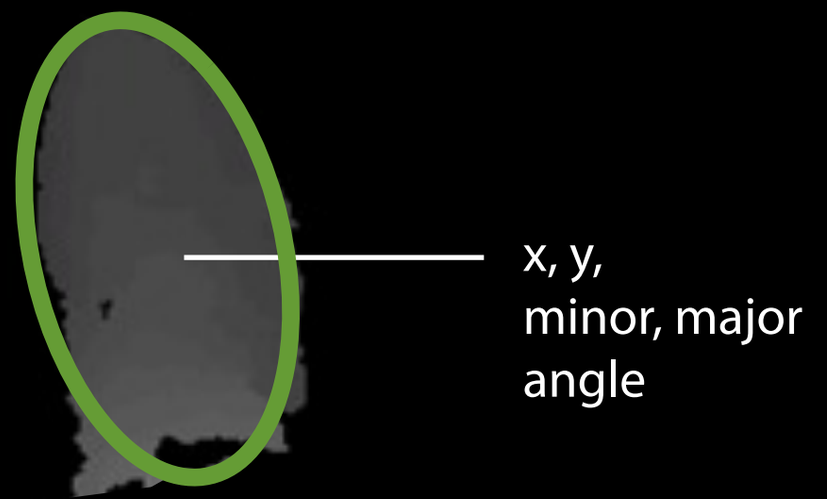
fit ellipse

```
cv::RotatedRect cv::fitEllipse(const cv::Mat& points);
```



fit ellipse

```
cv::RotatedRect cv::fitEllipse(const cv::Mat& points);
```




```
void cv::ellipse(cv::Mat& img,
                 const cv::RotatedRect& box,
                 const cv::Scalar& color,
                 int thickness = 1, int lineType = 8);
```

fit ellipse

we’re done!

oops, we made one assumption, remember?

what now?

30sec brainstorming

what now?

“empty” scene (= bg)

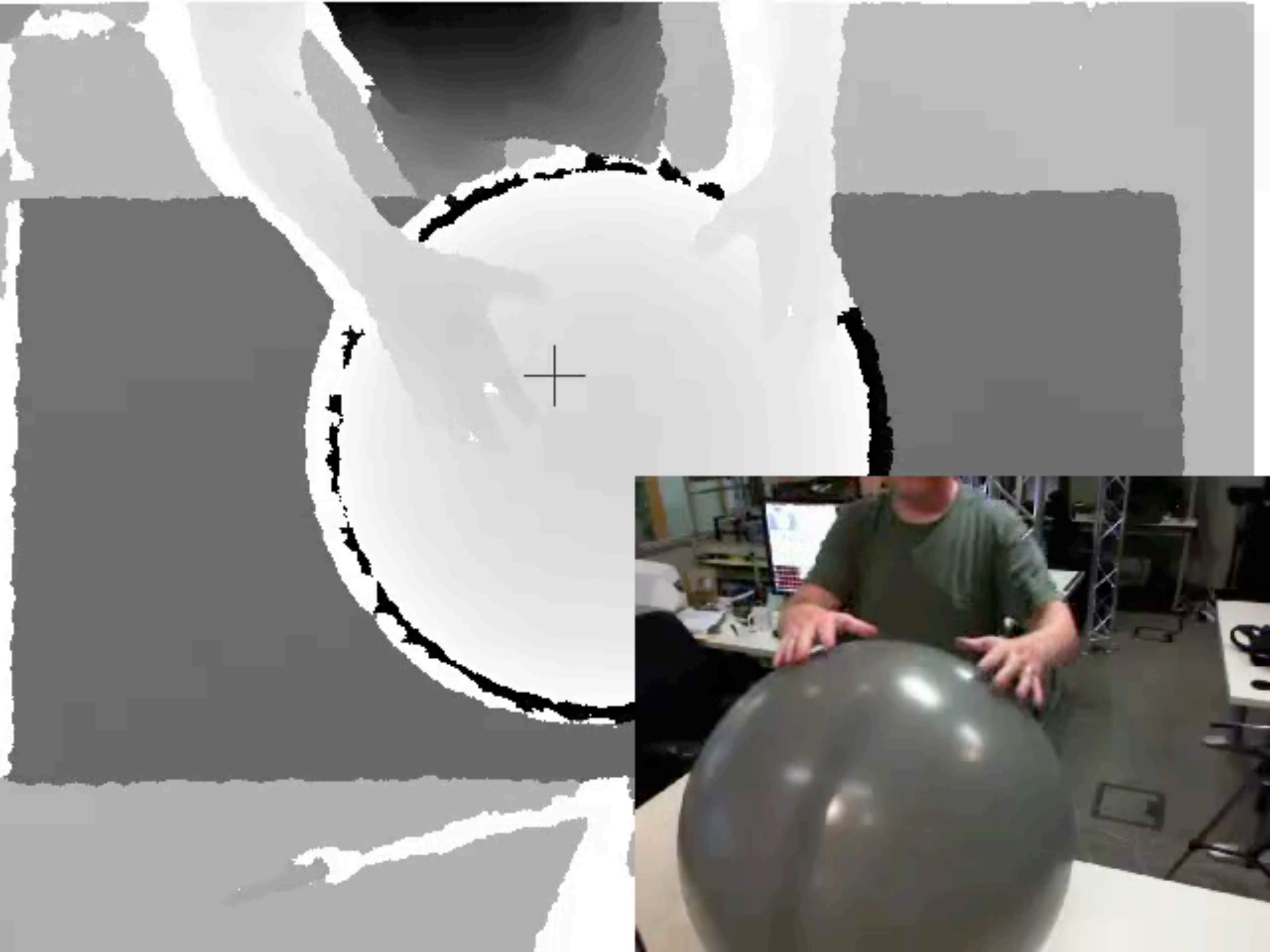
background subtraction

```
// several ways

void cv::absdiff(const cv::Mat& src1,
                 const cv::Mat& src2,
                 cv::Mat& dst);

void cv::subtract(const cv::Mat& src1,
                  const cv::Mat& src2,
                  cv::Mat& dst);
```

```
void cv::addWeighted(const cv::Mat& src1, double alpha,
                     const cv::Mat& src2, double beta,
                     double gamma, cv::Mat& dst);

// dst = src1 * alpha + src2 * beta + gamma;
```

background subtraction

Depth Image



Using a Depth Camera as a Touch Sensor

Andrew D. Wilson
Microsoft Research
Redmond, WA 98052 USA
awilson@microsoft.com

ABSTRACT
We explore the application of depth-sensing cameras to detect touch on a tabletop. Limits of depth estimate resolution and line of sight requirements dictate that the determination of the moment of touch will not be as precise as that of more direct sensing techniques such as capacitive touch screens. However, using a depth-sensing camera to detect touch has significant advantages: first, the interactive surface need not be instrumented. Secondly, this approach allows touch sensing on non-flat surfaces. Finally, information about the shape of the users and their arms and hands above the surface may be exploited in useful ways, such as determining hover state, or that multiple touches are from same hand or from the same user. We present techniques and findings using Microsoft Kinect.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Design, Human Factors

Keywords: Depth-sensing cameras, touchscreen interfaces.

INTRODUCTION

Depth-sensing cameras report distance to the nearest surface at each pixel. The depth camera included with Microsoft Kinect [1], based on a PrimeSense design, projects a fixed pattern of infrared light [2]. An offset infrared camera is used to calculate the precise manner in which this

In comparison with more traditional techniques, such as capacitive sensors, the use of depth cameras to sense touch has the following advantages:

- The interactive surface need not be instrumented.
- The interactive surface need not be flat.
- Information about the shape of the users and users' arms and hands above the surface may be exploited in useful ways, such as determining hover state, or that multiple touches are from the same hand or user.

However, given the depth estimate resolution of today's depth-sensing cameras, and the various limitations imposed by viewing the user and table from above, we expect that relying exclusively on the depth camera will not give as precise determination of the moment of touch as more traditional touchscreen technologies.

After reviewing related work, we discuss some considerations in sensing touch from depth cameras and then present a simple technique to achieve a useable touch signal.

RELATED WORK

Among the earliest applications of overhead video cameras to track fingers and detect touch on a tabletop are Wellner's Digital Desk [15] and Kruger's VIDEODESK [10]. Wellner discusses the difficulty of accurately determining touch, and proposes using a microphone to detect when the user's

Using a Depth Camera as a Touch Sensor

Andrew D. Wilson
Microsoft Research
Redmond, WA 98052 USA
awilson@microsoft.com

ABSTRACT
We explore the application of depth-sensing cameras to detect touch on a tabletop. Limits of depth estimate resolution and line of sight requirements dictate that the determination of the moment of touch will not be as precise as that of more direct sensing techniques such as capacitive touch screens. However, using a depth-sensing camera to detect touch has significant advantages: first, the interactive surface need not be instrumented. Secondly, this approach allows touch sensing on non-flat surfaces. Finally, information about the shape of the users and their arms and hands above the surface may be exploited in useful ways, such as determining lower state, or that multiple touches are from same hand or from the same user. We present techniques and findings using Microsoft Kinect.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Design, Human Factors

Keywords: Depth-sensing cameras, touch screen interfaces.

INTRODUCTION
Depth-sensing cameras report distance to the nearest surface at each pixel. The depth camera included with Microsoft Kinect [1], based on a PrimeSense design, projects a fixed pattern of infrared light [2]. An offset infrared camera is used to calculate the precise manner in which this pattern is distorted as a function of the depth of the nearest physical surface. Because the depth calculations are based on triangulating features in the image, depth precision decreases as the distance from the camera to subject increases. Pablo Prime/Soreac documents claim per-pixel depth resolution of 1cm when the camera is 2m away.

The Kinect software development kit exploits depth information to provide game developers with skeletal models of Xbox players. Such models are useful for animating a player's avatar in a game, for example. Here instead we consider the use of the same depth-sensing camera to emulate touch-screen interfaces. In particular, our goal is to deduce a useful touch signal when the camera is mounted well above a surface such as a desk or table.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ITS'10, November 7–10, 2010, Saarbrücken, Germany.
Copyright 2010 ACM 978-1-4503-0399-6/10/11...$10.00.

In comparison with more traditional techniques, such as capacitive sensors, the use of depth cameras to sense touch has the following advantages:

- The interactive surface need not be instrumented.
- The interactive surface need not be flat.
- Information about the shape of the users and users' arms and hands above the surface may be exploited in useful ways, such as determining lower state, or that multiple touches are from same hand or user.

However, given the depth estimate resolution of today's depth-sensing cameras, and the various limitations imposed by viewing the user and table from above, we expect that relying exclusively on the depth camera will not give as precise determination of the moment of touch as more traditional touchscreen technologies.

After reviewing related work, we discuss some considerations in sensing touch from depth cameras and then present a simple technique to achieve a useable touch signal.

RELATED WORK
Among the earliest applications of overhead video cameras to track fingers and detect touch on a tabletop are Wellner's Digital Desk [3,4] and Krueger's VIDEODESK [5,6]. Wellner discusses the difficulty of accurately determining touch, and proposes using a microphone to detect when the user's finger taps the surface. More recent attempts using a single camera exploit more refined models of the shape of fingertips [9,11], and possibly use dwell-time to signal clicking. Marshall et al [10] detect touch from the change in color of the fingernail when the finger is pressed against a surface.

The use of range information to determine when the user touches an uninstrumented surface has been explored in a variety of ways. Wren et al [28] used two cameras to detect touch using a "dual disparity" technique which finds objects at a particular depth rather than computing depth generally over the scene. Wilson's TouchLight [16] similarly uses two cameras and looks only for objects at the depth of a transparent display. Malik's Visual Touchpad [12] uses two cameras and binocular disparity calculations to determine when a fingertip is touching the surface. Agarwal et al [13] use two cameras and machine learning techniques to detect fingertips touching a display. Wilson's PlayAnywhere [17] single camera system exploits the shadow cast by the finger to determine range of the finger to the surface.

Depth-sensing cameras have been used in various interactive surface applications. Wilson's Micronurbanism game [18] explored the use of depth cameras to inform a physics simulation about objects on the surface. Benko et al's

DepthTouch [8] applied a depth camera to TouchLight's display, to more easily detect touch.

One motivation for using depth cameras to detect touch is the various other capabilities that they afford beyond touch, such as enabling "above the surface interaction" [14,7]. Another useful capability is the ability to attribute each touch to specific users. While such ability was first built into DiamondTouch [6], there have been a number of vision-based approaches to exploit more sophisticated models of contact orientation and shape with similar goals [5,16].

SENSING TOUCH FROM DEPTH
Assuming a clear line of sight, a natural approach to detect touch using a depth camera is to compare the current input depth image against some model of the touch surface. Pixels corresponding to the finger or hand will appear to be closer to the camera than the corresponding part of the known touch surface.

Taking all pixels closer than some threshold representing the depth of the surface will also include pixels belonging to the user's arms and potentially other objects that are not in contact with the tabletop. A second threshold may be used to eliminate pixels that are too far from the surface to be considered part of object in contact (see Figure 1):

$$d_{touch} > d_{off} > d_{surface} \quad (1)$$

This relation establishes a "shell" around surface area of interest (similar to [20] (see Figure 1). In the following we discuss how to set d_{touch} and d_{off}.

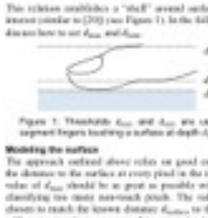


Figure 1. Thresholds d_{touch} and d_{off} are used to segment fingers touching a surface at depth $d_{surface}$.

Modeling the surface
The approach outlined above relies on good estimates of the distance to the surface at every pixel in the image. The value of d_{touch} should be as great as possible without misclassifying too many non-touch pixels. The value can be chosen to match the known distance $d_{surface}$ to the surface, with some margin to accommodate any noise in the depth image values. Setting this value too loosely risks cutting the tips of fingers off, which will cause an undesirable drift in contact position in later stages of processing.

For flat surfaces, such as a table, it may suffice to model the 3D position and orientation of the surface and compute $d_{surface}$ at given image coordinates from this model (this is the approach used in [20]).

Unfortunately, this idealized model does not account for the deviations due to noise in the depth image, slight variations in surface flatness or uncorrected lens distortions effects. Thus it will be necessary to place d_{touch} some distance above $d_{surface}$ to account for deviations from the model. In order to provide the best touch signal, we would like to minimize $d_{touch} - d_{true}$.

A better approach may be to find $d_{surface}$ for every pixel location by taking a "snapshot" of the depth image when the surface is empty. This non-parametric approach can model surfaces that are not flat, with the usual caveat the sensed surface must have a line of sight to the camera.

In our experience, depth image noise at a given pixel location is not normal nor is it the same at every pixel location. Depth is reported in millimeters as 16-bit integer values; these real world values are calculated from raw disk values, also 16-bit integers. A per-pixel histogram of raw disk values over several hundred frames of a motionless scene reveals that depth estimates are remarkably stable at many pixel locations, taking on only one value, but at other locations can oscillate between two adjacent values. In our experiments, d_{true} is determined at each pixel location by inspecting this histogram and, considering depth values from least depth to greatest depth, finding the first depth value for which histogram exceeds some small threshold value. We note that rather than build a full 16-bit histogram over the image, it is convenient to first take a "snapshot" of the scene and compute a histogram over a small range of deviation from the snapshot at each pixel location.

Setting the near threshold
Setting d_{touch} is less straightforward: too low of a value (too near) will cause contacts to be generated well before there is an actual touch. Too great of a value (too far) may make the resulting image of classified pixels difficult to group into distinct contacts. Setting d_{touch} too low or too high will cause a drift in contact position.

More generally, it is impossible to say whether an observed object is in contact from the depth image alone. We might be observing an object resting on the table, or it may be thinner than we expect and is instead hovering over the table. In particular, camera line of sight and the posture of the hand may make it impossible to detect touch. Consider, for example, an outstretched index finger lying along the ray from the camera to the hand, or the hand completely obscuring the touching finger.

In practice, we must make some assumption about the anthropometry of fingers and hands, and their posture during touch. We choose d_{touch} to match the typical thickness of the finger resting on the surface, and assume that the finger lies flat on the surface at least along the area of contact: $d_{touch} = d_{true} - \tau$.

Forming contacts
After computing the surface histogram, the values d_{true} may be stored as an image of thresholds, used in a single pass to classify all pixels in the input depth image according to Equation 1.

The resulting binary image shows significant edge effects around the contour of the hand, even when the hand is well above d_{touch} (see Figure 2a). These artifacts may be removed by low-pass filtering the image; we use a separable binomial

filter (9×9 pixels) followed by thresholding to obtain regions where there is good support for full contacts (Figure 2d). Discrete points of contact may be found in this final image by techniques common to imaging interactive touch screens. For example, connected components analysis may be used to discover groups of pixels corresponding to contacts. These may be tracked over time to implement familiar multi-touch interactions.

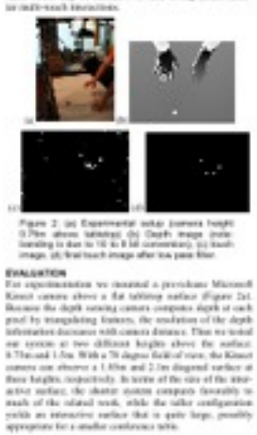


Figure 2: (a) Experimental setup (camera height 0.79m above tabletop) (b) Depth image (note banding is due to 10- to 8-bit conversion). (c) touch image. (d) final touch image after low pass filter.

EVALUATION
For experimentation we mounted a prerelease Microsoft Kinect camera above a flat tabletop surface (Figure 2a). Because the depth sensing camera computes depth at each pixel by triangulating features, the resolution of the depth information decreases with camera distance. Thus we tested our system at two different heights above the surface: 0.75m and 1.3m. With a 70 degree field of view, the Kinect camera can observe a 1.05m and 2.1m diagonal surface at these heights, respectively. In terms of the size of the interactive surface, the shorter system supports (roughly) as much of the related work, while the taller configuration yields an interactive surface that is quite large, possibly appropriate for a smaller conference table.

We configured the camera to report depth shift data in a 640×480 16 bit image at 30fps. The threshold d_{near} was set automatically as described earlier by collecting a histogram of depth values of the empty surface over a few hundred frames. After some experimentation, it was determined that $\tau = 4$ and $\tau = 7$ (depth shift values, not mm) gives a good value for $d_{touch} = d_{true} - \tau$, for the 0.75m height and 1.3m height configurations, respectively. These values result in good contact formation, as well as the ability to pick up much of the hand when it is flat on the surface.

In evaluating the performance of our technique, we are interested in the reliability of touch determination, and in the accuracy of touch position. Because of the limitations of depth resolution and choosing thresholds d_{touch} and d_{true}, we expect touch to be less reliable than that of more direct means of sensing touch (e.g., capacitive touch screens).

To verify d_{true} and d_{touch} in the actual setup and also give an idea of the limits of touch reliability, we observed where the thresholds lie in terms of actual height above the surface by stacking physical objects of varying height and observing changes in the output image. We used Post-it® note pads, because they can easily be stacked and split to achieve a particular height (Figure 3). For the 0.75m height, d_{true} and d_{touch} are observed to lie at 5mm and 14mm above $d_{surface}$, respectively. For the 1.3m height, d_{true} and d_{touch} lie 6mm and 16mm above $d_{surface}$, respectively.

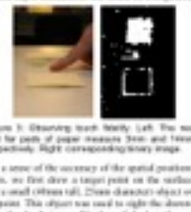


Figure 3: Observing touch fidelity. Left: The near and far pads of paper measure 5mm and 14mm, respectively. Right: corresponding binary image.

To get a sense of the accuracy of the spatial position of the contacts, we first drew a target point on the surface, then placed a small (40mm tall, 25mm diameter) object over the target point. This object was used to right the drawn target point in the depth image. We then clicked on the center of the object in the depth image to provide ground truth target position. After removing the object, the target was reached a number of times, from a variety of angles. It is important to note that calculating the position of the target was very simplistic, and did not incorporate adjustments to more closely match users' expectations (see [8]). In each case the contact position (centroid) was compared to the ground truth target position. Informal observations indicate that the mean case error was about 6 pixels (about 10mm at the surface) (1.5cm height) and 3 pixels (7mm @ 75cm height).

To demonstrate that the system can work with non-flat surfaces, we placed a book on the table, and re-ran the surface calibration. Figure 4 illustrates contacts recovered when touching the book.

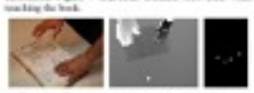


Figure 4. Detecting touch on a non-flat surface. Left: touching a book. Middle: depth image. Right: detected contacts.

BEYOND TOUCH
The focus of the present work is on enabling touch on an uninstrumented surface. While the touch performance using a depth camera alone may never approach that of touch sensors such as capacitive touch screens, depth cameras enable a wide variety of interactions that go beyond any conventional touch screen sensor. In particular to interactive surface applications, it is easy to see how depth cameras can provide more information about the user during the sensing. Figure 5 illustrates a simple segmentation of the user above the calibrated surface. For example, depth cameras are well suited to enable "above the surface" interactions [18,8,7,19], such as picking up a virtual object, "holding" it in the air above the surface, and dropping it elsewhere.

One particularly basic calculation that seems especially useful in considering touch interfaces is the ability to determine that multiple touch contacts are from the same hand, or that multiple contacts are from the same user. Such connectivity information is easily calculated by noting that two contacts made by the same user will reduce into the same "above the surface" component (as in Figure 5).

Figure 5: Simple depth threshold shows user's body as a single connected component (camera at 1.5m height above tabletop, for 2.1m diagonal area of interaction).

EXTENSIONS
The limitations of the very simple approach proposed in this paper suggest a number of items for future work. First, we note that further processing would be required to segregate physical objects placed and possibly moved on the surface, as distinct from touch contacts. To then detect touching those objects (such as the book in Figure 4), the surface calibration must be updated appropriately. Dynamic calibration may also be useful when the surface itself is moved.

Secondly, the calculation of contact position could be made more accurate by exploiting shape or posture information available in the depth camera. This could even include corrections based on the user's eye-point (as in [8]), which may be approximated directly from the depth image by finding the user's head position. Note that, as discussed above, it is easy to match a particular contact to that user's body.

Finally, some changes may be required to work with other depth-sensing camera technologies. Time-of-flight based depth cameras, for example, have different noise characteristics than the PrimeSense camera and may require more than a simple histogram of depth values at each pixel location.

CONCLUSION
We demonstrate how a depth-sensing camera may be used to detect touch on an un-instrumented surface. While the performance of this approach is less than that of more conventional touch screen technologies, we believe the performance is good enough to be useful in a variety of applications. Additionally, the approach offers certain interesting advantages, such as working on non-flat surfaces and to convert with "above the surface" interaction techniques.

REFERENCES
1. http://www.xbox.com/kinect last accessed Sep. 20, 2010.
2. http://www.primesense.com, last accessed Sep. 20, 2010.
3. Agarwal, A., Izadi, S., Chandraker M., Blake, A. High precision multi-touch sensing on surfaces using overhead cameras. *Proc. IEEE Tabletop 2007*, 197–200.
4. Benko, H. and Wilson, A.D. DepthTouch: using depth-sensing camera to enable freehand interactions on and above the interactive surface. *Microsoft Research Technical Report MSR-TR-2009-23* March, 2009.
5. Dang, C. T., Straub, M., and André, E. Hand distinction for multi-touch tabletop interaction. *Proc. ACM ITS 2009*, 101–108.
6. Dietz, P. and Leigh, D. DiamondTouch: a multi-user touch technology. *Proc. ACM UIST 2001*, 219-226.
7. Hilliges, O., Izadi, S., Wilson, A. D., Hodges, S., Garcia-Mendoza, A., and Butz, A. 2009. Interactions in the air: adding further depth to interactive tabletops. *Proc. ACM UIST 2009*, 139-148.
8. Holz, C. and Baudisch, P. The generalized perceived input point model and how to double touch accuracy by extracting fingerprints. *Proc. ACM CHI 2010*, 581-590.
9. Kjeldsen, R., Pinhanez, C., Pingali, G., Jacob Hartman, J., Tang Levas, T., Podlaseck, M. Interacting with steerable projected displays. *Proc. International Conference on Automatic Face and Gesture Recognition*, 2002.
10. Krueger, M. *Artificial Reality 2*. Addison-Wesley, 1991.
11. Leibe, J. and Benard, P. Visual tracking of bare fingers for interactive surfaces. *Proc. ACM UIST 2004*, 119-122.
12. Malik, S. and Laszlo, J. Visual touchpad: a two-handed gestural input device. *Proc. ICMI 2004*, 289-296.
13. Marshall, J., Pridmore, T., Pound, M., Benford, S., and Koleva, B. Proving the Point: Sensing Multiple Touch and Finger Pressure on Arbitrary Surfaces. *Proc. Pervasive 2008*, 80-97.
14. Wang, F., Cao, X., Ren, X., and Irani, P. Detecting and leveraging finger orientation for interaction with direct-touch surfaces. *Proc. ACM UIST 2009*, 23-32.
15. Wellner, P. The DigitalDesk calculator: tangible manipulation on a desk top display. *Proc. ACM UIST 1991*, 27-33.
16. Wilson, A. D. TouchLight: an imaging touch screen and display for gesture-based interaction. *Proc. ICMI 2004*, 69-76.
17. Wilson, A. D. PlayAnywhere: a compact interactive tabletop projection-vision system. *Proc. ACM UIST 2005*, 83-92.
18. Wilson, A.D. Depth-sensing video cameras for 3D tangible tabletop interaction. *Proc. IEEE Tabletop 2007*, 201-204.
19. Wilson, A.D., and Benko, H. Combining multiple depth cameras and projectors for interactions on, above and between surfaces. *Proc. ACM UIST 2010*.
20. Wren, C., and Ivanov, Y. Volumetric operations with surface margins. In *CVPR: Technical Sketches*, 2001.

Wilson ITS '10

Using a Depth Camera as a Touch Sensor

Andrew D. Wilson
Microsoft Research
Redmond, WA 98052 USA
awilson@microsoft.com

ABSTRACT
We explore the application of depth-sensing cameras to detect touch on a tabletop. Limits of depth estimate resolution and line of sight requirements dictate that the determination of the moment of touch will not be as precise as that of more direct sensing techniques such as capacitive touch screens. However, using a depth-sensing camera to detect touch has significant advantages: first, the interactive surface need not be instrumented. Secondly, this approach allows touch sensing on non-flat surfaces. Finally, information about the shape of the users and their arms and hands above the surface may be exploited in useful ways, such as determining lower state, or that multiple touches are from same hand or from the same user. We present techniques and findings using Microsoft Kinect.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Design, Human Factors

Keywords: Depth-sensing cameras, touch screen interfaces.

INTRODUCTION
Depth-sensing cameras report distance to the nearest surface at each pixel. The depth camera included with Microsoft Kinect [1], based on a PrimeSense design, projects a fixed pattern of infrared light [2]. An offset infrared camera is used to calculate the precise manner in which this pattern is distorted as a function of the depth of the nearest physical surface. Because the depth calculations are based on triangulating features in the image, depth precision decreases as the distance from the camera to subject increases. Pablo PrimeSense documents claim per-pixel depth resolution of 1cm when the camera is 2m away.

The Kinect software development kit exploits depth information to provide game developers with skeletal models of Xbox players. Such models are useful for animating a player's avatar in a game, for example. Here instead we consider the use of the same depth-sensing camera to emulate touch-screen technology. In particular, our goal is to deduce a useful touch signal when the camera is mounted well above a surface such as a desk or table.

In comparison with more traditional techniques, such as capacitive sensors, the use of depth cameras to sense touch has the following advantages:

- The interactive surface need not be instrumented.
- The interactive surface need not be flat.
- Information about the shape of the users and users' arms and hands above the surface may be exploited in useful ways, such as determining lower state, or that multiple touches are from the same hand or user.

However, given the depth estimate resolution of today's depth-sensing cameras, and the various limitations imposed by viewing the user and table from above, we expect that relying exclusively on the depth camera will not give as precise determination of the moment of touch as more traditional touchscreen technologies.

After reviewing related work, we discuss some considerations in sensing touch from depth cameras and then present a simple technique to achieve a useable touch signal.

RELATED WORK
Among the earliest applications of overhead video cameras to track fingers and detect touch on a tabletop are Wellner's Digital Desk [3,5] and Krueger's VIDEODESK [6]. Wellner discusses the difficulty of accurately determining touch, and proposes using a microphone to detect when the user's finger taps the surface. More recent attempts using a single camera exploit more refined models of the shape of fingertips [9,11], and possibly use dwell-time to signal clicking. Marshall et al [10] detect touch from the change in color of the fingernail when the finger is pressed against a surface.

The use of range information to determine when the user touches an uninstrumented surface has been explored in a variety of ways. Wren et al [28] used two cameras to detect touch using a "dual disparity" technique which finds objects at a particular depth rather than computing depth generally over the scene. Wilson's TouchLight [16] similarly uses two cameras and looks only for objects at the depth of a transparent display. Malik's Visual Touchpad [12] uses two cameras and binocular disparity calculations to determine when a fingertip is touching the surface. Agarwal et al [13] use two cameras and machine learning techniques to detect fingertips touching a display. Wilson's PlayAnywhere [17] single camera system exploits the shadow cast by the finger to determine range of the finger to the surface.

Depth-sensing cameras have been used in various interactive surface applications. Wilson's Micromotorcross game [16] explored the use of depth cameras to inform a physics simulation about objects on the surface. Benko et al's DepthTouch [4] applied a depth camera to TouchLight's display, to more easily detect touch.

One motivation for using depth cameras to detect touch is the various other capabilities that they afford beyond touch, such as enabling "above the surface interaction" [4,17]. Another useful capability is the ability to attribute each touch to specific users. While such ability was first built into DiamondTouch [8], there have been a number of vision-based approaches to exploit more sophisticated models of contact

SENSING TOUCH FROM DEPTH
Assuming a clear line of sight, a natural approach to detect touch using a depth camera is to compare the current input depth image against some model of the touch surface. Pixels corresponding to the finger or hand will appear to be closer to the camera than the corresponding part of the known touch surface.

Taking all pixels closer than some threshold representing the depth of the surface will also include pixels belonging to the user's arm and potentially other objects that are not in contact with the tabletop. A second threshold may be used to eliminate pixels that are too far from the surface to be considered part of object in contact (see Figure 1):

$$d_{touch} > d_{pixel} > d_{hover} \quad (1)$$

This relation establishes a "shell" around surface area of interest (similar to [20] (see Figure 1). In the following we discuss how to set d_{hover} and d_{touch}.

Figure 1. Thresholds d_{hover} and d_{touch} are used to segment fingers touching a surface at depth $d_{surface}$.

Modeling the surface
The approach outlined above relies on good estimates of the distance to the surface at every pixel in the image. The value of d_{touch} should be as great as possible without misclassifying too many non-touch pixels. The value can be chosen to match the known distance $d_{surface}$ to the surface, with some margin to accommodate any noise in the depth image values. Setting this value too loosely risks cutting the tips of fingers off, which will cause an undesirable shift in contact position in later stages of processing.

For flat surfaces, such as a table, it may suffice to model the 3D position and orientation of the surface and compute $d_{surface}$ at given image coordinates from this model (this is the approach used in [20]).

Unfortunately, this idealized model does not account for the deviations due to noise in the depth image, slight variations in surface flatness or uncorrected lens distortions effects. Thus it will be necessary to place d_{hover} some distance above $d_{surface}$ to account for deviations from the model. In order to provide the best touch signal, we would like to minimize $d_{touch} - d_{hover}$.

A better approach may be to find $d_{surface}$ for every pixel location by taking a "snapshot" of the depth image when the surface is empty. This non-parametric approach can model surfaces that are not flat, with the usual caveat the sensed surface must have a line of sight to the camera.

In our experience, depth image noise at a given pixel location is not normal nor is it the same at every pixel location. Depth is reported in millimeters as 16-bit integer values; these real world values are calculated from raw disk values, also 16-bit integers. A per-pixel histogram of raw disk values over several hundred frames of a motionless scene reveals that depth estimates are remarkably stable at many pixel locations, taking on only one value, but at other locations can oscillate between two adjacent values. In our experiments, $d_{surface}$ is determined at each pixel location by inspecting this histogram and, considering depth values from least depth to greatest depth, finding the first depth value for which histogram exceeds some small threshold value. We note that rather than build a full 16-bit histogram over the image, it is convenient to first take a "snapshot" of the scene and compute a histogram over a small range of deviation from the snapshot at each pixel location.

Setting the near threshold
Setting d_{hover} is less straightforward: too low of a value (too near) will cause contacts to be generated well before there is an actual touch. Too great of a value (too far) may make the resulting image of classified pixels difficult to group into distinct contacts. Setting d_{hover} too low or too high will cause a drift in contact position.

More generally, it is impossible to say whether an observed object is in contact from the depth image alone. We might be observing an object resting on the table, or it may be thinner than we expect and is instead hovering over the table. In particular, camera line of sight and the posture of the hand may make it impossible to detect touch. Consider, for example, an outstretched index finger lying along the ray from the camera to the hand, or the hand completely obscuring the touching finger.

In practice, we must make some assumption about the anthropometry of fingers and hands, and their posture during touch. We choose d_{hover} to match the typical thickness of the finger resting on the surface, and assume that the finger lies flat on the surface at least along the area of contact: $d_{hover} \approx d_{touch}$.

After computing the surface histogram, the values d_{hover} may be stored as an image of thresholds, used in a single pass to classify all pixels in the input depth image according to Equation 1.

The resulting binary image shows significant edge effects around the contour of the hand, even when the hand is well above d_{hover} (see Figure 2c). These artifacts may be removed by low-pass filtering the image; we use a separable binomial

filter (9×9 pixels) followed by thresholding to obtain regions where there is good support for full contacts (Figure 2d). Discrete points of contact may be found in this final image by techniques common to imaging interactive touch screens. For example, connected components analysis may be used to discover groups of pixels corresponding to contacts. These may be tracked over time to implement familiar multi-touch interactions.

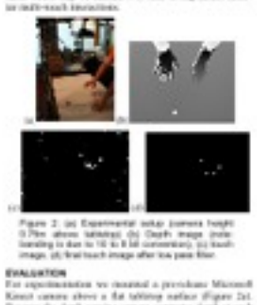


Figure 2: (a) Experimental setup (camera height 0.79m above tabletop) (b) Depth image (note banding is due to 10- to 8-bit conversion). (c) touch image. (d) final touch image after low pass filter.

EVALUATION
For experimentation we mounted a prosilumo Microsoft Kinect camera above a flat tabletop surface (Figure 2a). Because the depth sensing camera computes depth at each pixel by triangulating features, the resolution of the depth information decreases with camera distance. Thus we tested our system at two different heights above the surface: 0.75m and 1.3m. With a 70 degree field of view, the Kinect camera can observe a 1.05m and 2.1m diagonal surface at these heights, respectively. In terms of the size of the interactive surface, the shorter setting supports (roughly) as much of the related work, while the taller configuration yields an interactive surface that is quite large, possibly appropriate for a smaller conference table.

We configured the camera to report depth shift data in a 640×480 16 bit image at 30fps. The threshold d_{hover} was set automatically as described earlier by collecting a histogram of depth values of the empty surface over a few hundred frames. After some experimentation, it was determined that $r = 4$ and $r = 7$ (depth shift values, not mm) gives a good value for $d_{hover} = d_{touch} - r$, for the 0.75m height and 1.3m height configurations, respectively. These values result in good contact formation, as well as the ability to pick up much of the hand when it is flat on the surface.

In evaluating the performance of our technique, we are interested in the reliability of touch determination, and in the accuracy of touch position. Because of the limitations of depth resolution and choosing thresholds d_{hover} and d_{touch}, we expect touch to be less reliable than that of more direct means of sensing touch (e.g., capacitive touch screens).

To verify d_{hover} and d_{touch} in the actual setup and also give an idea of the limits of touch reliability, we observed where the thresholds lie in terms of actual height above the surface by stacking physical objects of varying height and observing changes in the output image. We used Post-it® note pads, because they can easily be stacked and split to achieve a particular height (Figure 3). For the 0.75m height, d_{hover} and d_{touch} are observed to lie at 5mm and 14mm above $d_{surface}$, respectively. For the 1.3m height, d_{hover} and d_{touch} lie 6mm and 16mm above $d_{surface}$, respectively.

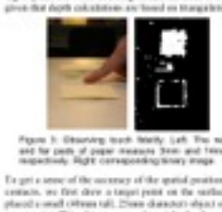


Figure 3: Observing touch fidelity. Left: Two near and far pads of paper measure 5mm and 14mm, respectively. Right: corresponding binary image.

To get a sense of the accuracy of the spatial position of the contacts, we first drew a target point on the surface, then placed a small (40mm tall, 25mm diameter) object over the target point. This object was used to right the drawn target point in the depth image. We then clicked on the center of the object in the depth image to provide ground truth target position. After removing the object, the target was reached a number of times, from a variety of angles. It is important to note that calculating the position of the target was very simplistic, and did not incorporate adjustments to more closely match users' expectations (see [8]). In each case the contact position (centroid) was compared to the ground truth target position. Informal observations indicate that the worst case error was about 6 pixels (about 10mm at the surface) (1.5cm height and 3 pixels (7mm) @ 75cm height).

To demonstrate that the system can work with non-flat surfaces, we placed a book on the table, and re-ran the surface calibration. Figure 4 illustrates contacts recovered when touching the book.

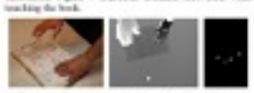


Figure 4. Detecting touch on a non-flat surface. Left: touching a book. Middle: depth image. Right: detected contacts.

BEYOND TOUCH
The focus of the present work is on enabling touch on an uninstrumented surface. While the touch performance using a depth camera alone may never approach that of touch sensors such as capacitive touch screens, depth cameras enable a wide variety of interactions that go beyond any conventional touch screen sensor. In particular to interactive surface applications, it is easy to see how depth cameras can provide more information about the user doing the touching. Figure 5 illustrates a simple segmentation of the user above the calibrated surface. For example, depth cameras are well suited to enable "above the surface" interactions [18,8,7,19], such as picking up a virtual object, "holding" it in the air above the surface, and dropping it elsewhere.

One particularly basic calculation that seems especially useful in considering touch interfaces is the ability to determine that multiple touch contacts are from the same hand, or that multiple contacts are from the same user. Such connectivity information is easily calculated by noting that two contacts made by the same user will adhere into the same "above the surface" component (as in Figure 5).

Figure 5: Simple depth threshold shows user's body as a single connected component (camera at 1.5m height above tabletop, for 2.1m diagonal area of interaction).

EXTENSIONS
The limitations of the very simple approach proposed in this paper suggest a number of items for future work. First, we note that further processing would be required to recognize physical objects placed and possibly moved on the surface, as distinct from touch contacts. To then detect touching those objects (such as the book in Figure 4), the surface calibration must be updated appropriately. Dynamic calibration may also be useful when the surface itself is moved.

Secondly, the calculation of contact position could be made more accurate by exploiting shape or posture information available in the depth camera. This could even include extensions based on the user's eye-point (as in [8]), which may be approximated directly from the depth image by finding the user's head position. Note that, as discussed above, it is easy to match a particular contact to that user's body.

Finally, some changes may be required to work with other depth-sensing camera technologies. Time-of-flight based depth cameras, for example, have different noise characteristics than the PrimeSense camera and may require more than a simple histogram of depth values at each pixel location.

CONCLUSION
We demonstrate how a depth-sensing camera may be used to detect touch on an uninstrumented surface. While the performance of this approach is less than that of more conventional touch screen technologies, we believe the performance is good enough to be useful in a variety of applications. Additionally, the approach offers certain interesting advantages, such as working on non-flat surfaces and in concert with "above the surface" interaction techniques.

REFERENCES
1. http://www.xbox.com/kinect last accessed Sep 20, 2010.
2. http://www.primesense.com/ last accessed Sep 20, 2010.
3. Agarwal, A., Izadi, S., Chandraker M., Blake, A. High precision multi-touch sensing on surfaces using overhead cameras. *Proc. IEEE Tabletop 2007*, 197-200.
4. Benko, H. and Wilson, A.D. DepthTouch: using depth-sensing camera to enable freehand interactions on and above the interactive surface. *Microsoft Research Technical Report MSR-TR-2009-23* March, 2009.
5. Dang, C. T., Straub, M., and André, E. Hand distinction for multi-touch tabletop interaction. *Proc. ACM ITS 2009*, 101-108.
6. Davis, P. and Leigh, D. DiamondTouch: a multi-user touch technology. *Proc. ACM UIST 2001*, 219-226.
7. Hilliges, O., Izadi, S., Wilson, A. D., Hodges, S., Garcia-Mendoza, A., and Butz, A. 2009. Interactions in the air: adding further depth to interactive tabletops. *Proc. ACM UIST 2009*, 139-148.
8. Holz, C. and Baudisch, P. The generalized perceived input point model and how to double touch accuracy by extracting fingerprints. *Proc. ACM CHI 2010*, 581-590.
9. Kjeldsen, R., Pinhanez, C., Pingali, G., Hartman, J., Levas, T., Podlaseck, M. Interacting with steerable projected displays. *Proc. International Conference on Automatic Face and Gesture Recognition*, 2002.
10. Krueger, M. *Artificial Reality 2*. Addison-Wesley, 1991.
11. Leibe, J. and Starner, P. Visual tracking of bare fingers for interactive surfaces. *Proc. ACM UIST 2004*, 119-122.
12. Malik, S. and Laszlo, J. Visual touchpad: a two-handed gestural input device. *Proc. ICMI 2004*, 289-296.
13. Marshall, J., Pridmore, T., Pound, M., Benford, S., and Koleva, B. Proving the Point: Sensing Multiple Touch and Finger Pressure on Arbitrary Surfaces. *Proc. Pervasive 2008*, 80-97.
14. Wang, F., Cao, X., Ren, X., and Irani, P. Detecting and leveraging finger orientation for interaction with direct-touch surfaces. *Proc. ACM UIST 2009*, 23-32.
15. Wellner, P. The DigitalDesk calculator: tangible manipulation on a desk top display. *Proc. ACM UIST 1991*, 27-33.
16. Wilson, A. D. TouchLight: an imaging touch screen and display for gesture-based interaction. *Proc. ICMI 2004*, 69-76.
17. Wilson, A. D. PlayAnywhere: a compact interactive tabletop projection-vision system. *Proc. ACM UIST 2005*, 83-92.
18. Wilson, A.D. Depth-sensing video cameras for 3D tangible tabletop interaction. *Proc. IEEE Tabletop 2007*, 201-204.
19. Wilson, A.D., and Benko, H. Combining multiple depth cameras and projectors for interactions on, above and between surfaces. *Proc. ACM UIST 2010*.
20. Wren, C., and Ivanov, Y. Volumetric operations with surface margins. In *CVPR Technical Sketches*, 2001.

Wilson ITS '10

```
// dst = src * alpha + beta

img1.convertTo( img2, CV_8U, 10.0, -20.0 );
```

Kinect image

```
// dst = src * alpha + beta

img1.convertTo( img2, CV_8U, 10.0, -20.0 );
```

depth

Kinect image

```
// dst = src * alpha + beta

img1.convertTo( img2, CV_8U, 10.0, -20.0 );
                             ↑
                             α
```

Kinect image

```
// dst = src * alpha + beta

img1.convertTo( img2, CV_8U, 10.0, -20.0 );
                                   ↑
                                   β
```

Kinect image

1) find shoes in the depth image

2) draw a circle **iff** a shoe is making **contact with the ground**

for those who have done this already:

3) track the shoes

4) draw lines between corresponding circles

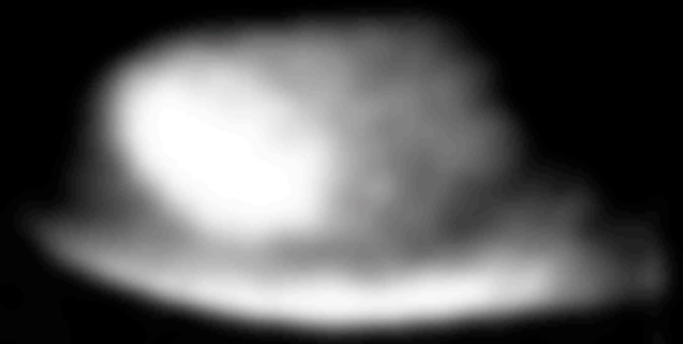
assignment

practical introduction to

(open) computer vision

hci research :: hasso plattner institute
christian holz

blur



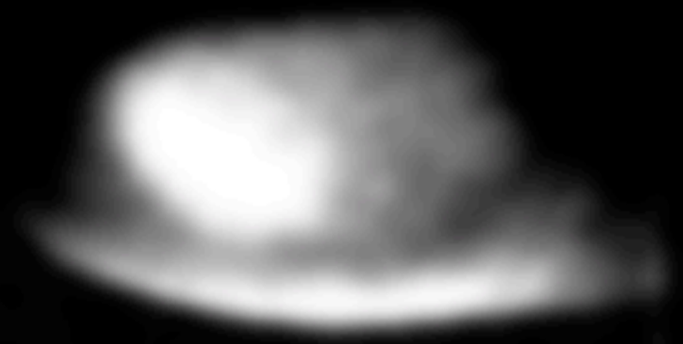
```
void cv::blur(const cv::Mat& src,
              cv::Mat& dst,
              cv::Size ksize);
```

blur

blur

```
void cv::blur(const cv::Mat& src,
              cv::Mat& dst,
              cv::Size ksize);
```

odd size



```
void cv::blur(const cv::Mat& src,
              cv::Mat& dst,
              cv::Size ksize);
```

blur