# Segmentation

# Segmentation

**This section will cover :**

- Binary Images
- Thresholding
- Masking
- Morphology

These topics will be **implemented** and **tested** in Python with OpenCV.

# Binary Images

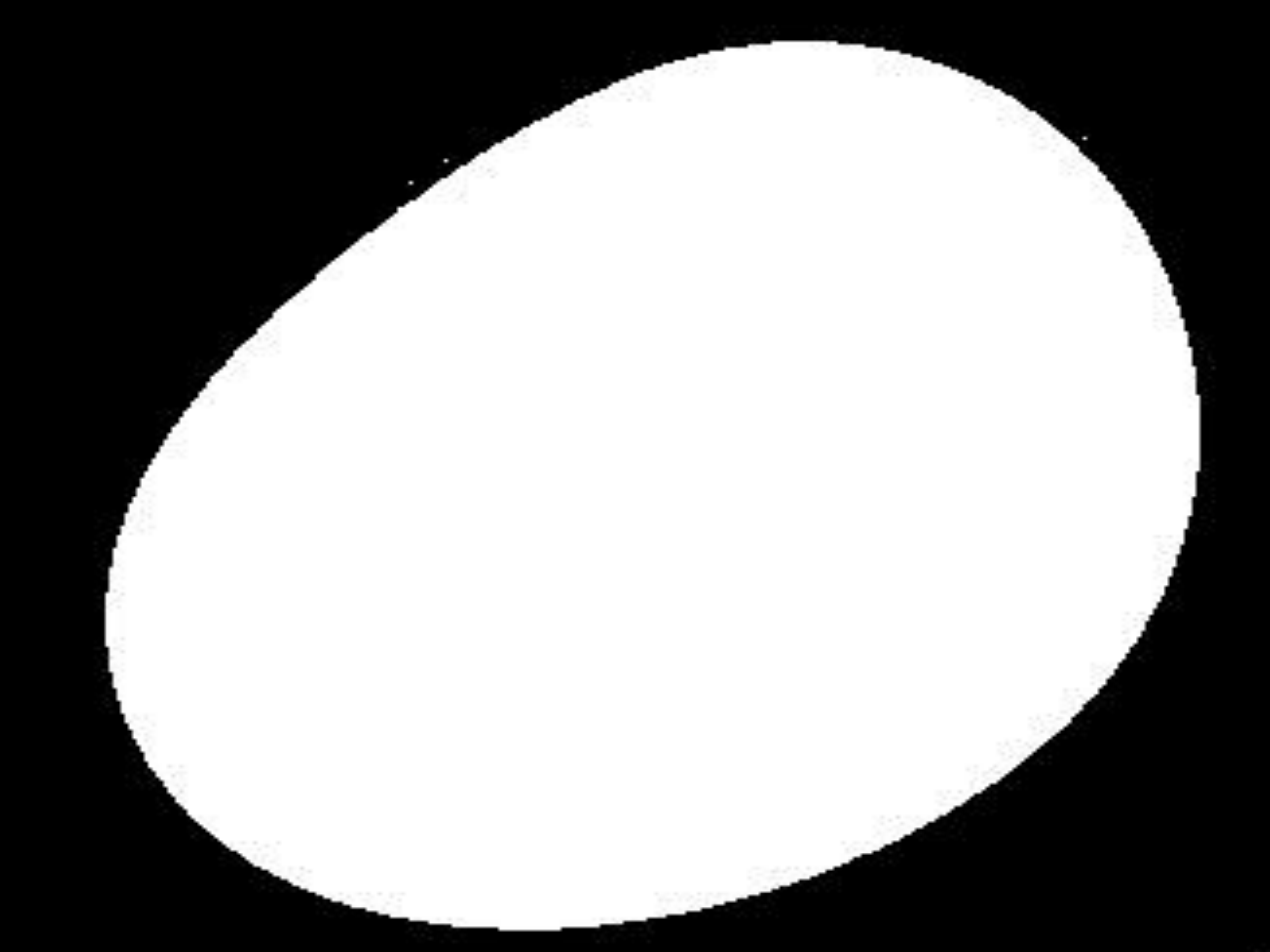
We have already seen binary images, but producing meaningful binary images requires some image understanding.

Often, we are interested in separating something in the foreground from the background.

The area we are interested in is call a *region of interest* (ROI).

Generally, the ROI is set to white while the rest is set to black.

To achieve this segmentation, we begin with a simple process called *thresholding*.



# Thresholding

# Thresholding

By selecting a threshold, we set the boundary between the ROI and the background.

Generally, thresholding is performed on a single channel (often the intensity, but any channel can be used).

This means the image must first be converted to grayscale (2D).

# Thresholding

Thresholding is then performed as:

$$B = \begin{cases} 255 & \text{where } G > \mathrm{T} \\ 0 & \text{where } G < \mathrm{T} \end{cases}$$

Where $B$ is the binary image output, $G$ is the grayscale input and $T$ is the threshold.

# Thresholding

By selecting different thresholds, we can set how inclusive or exclusive the ROI is.

For example, a low threshold will let more pixels into the ROI while a high threshold will exclude more pixels.

Threshold selection is often a challenge.

# Threshold Selection

Thresholds can be selected in various ways.

Using a priori knowledge (what you already know about the image) can be reasonable in some applications.

When little is known, a threshold is often selected by statistical analysis of the image.

For example, a suitable threshold might be:

$$T = \bar{I} \pm \sigma_I$$

Where $\bar{I}$ is the mean value and $\sigma_I$ is its standard deviation.

# Other Channels

The thresholding here has been done on the intensity of the image but as mentioned, thresholding can be performed on any channel.

For example, thresholding the Hue channel can lead to colour segmentation.

# Multiple Thresholds

It can be useful to combine segmentations.

For example, we may want to find areas that have a particular colour and brightness.

Or perhaps, we are looking for a colour in a range.

Multiple thresholds can be combined using Boolean logic.

e.g. $B = (H < 20) \,\&\, (I > 150)$

This would give a ROI with low hue and high intensity (bright red).

# Global Threshold

Using a global threshold means having the same threshold for the whole image.

This works well for images with good distinction between the ROI and the background.

However, where there are lighting variations, it can cause problems...

# Sonnet for Lena

O dear Lena, your beauty is so
It is hard sometimes to describe
I thought the entire world I woul
If only your portrait I could com
Alas! First when I tried to use V
I found that your cheeks belong to on
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this. I'll just digitize.'

*Thomas Colthurst*

## Sonnet [illegible]

O dear [illegible]
It is hard [illegible]
I thought the [illegible]
If only your [illegible]
[illegible] First when I [illegible]
[illegible] that your cheeks belong [illegible]
[illegible] hair contains a thousand lines
[illegible] with sums of discrete cosines
[illegible] sensual and tactual
[illegible] the proper fractal.

# Adaptive Thresholding

Adaptive thresholding is a handy solution to the variable lighting problem.

In this method, a local threshold is found for each region of the image.

Each threshold is the mean of the region plus a constant offset.

Setting the region size and constant offset allows control over the exclusivity of the ROI.

# Sonnet for Lena

O dear Lena, your beauty is so vast
[illegible] sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
[illegible] when I tried to use VQ
[illegible] your cheeks belong to only you
[illegible] contains a thousand lines
[illegible] with sums of discrete cosines.
[illegible] sensual and tactual
[illegible] the proper fractal
[illegible]
[illegible]
[illegible]
[illegible]

In Python with OpenCV

# Simple Thresholding

Simple binary thresholding can be performed in OpenCV using the *threshold* function:

```
T, B = cv2.threshold(G, thresh = T, maxval = 255,
                            type = cv2.THRESH_BINARY)
```

`T` is the threshold, `B` is the binary output image and `G` is the grayscale input image.

# Threshold Selection

To select better thresholds based on statistical analysis of the image, we use Numpy.

For example, to implement a threshold of $T = \bar{I} \pm \sigma_I$:

```
T = np.mean(G) + np.std(G)
```

`G` is the grayscale version of the image `I`.

# Multiple Thresholds

To use multiple thresholds, the *inRange* function can be useful:

```
RangeLower = (0, 150, 150)
RangeUpper = (50, 255, 255)
```

```
B = cv2.inRange(I, RangeLower, RangeUpper)
```

This would give an ROI with low blue and high green and red values (if `I` is a standard BGR image).

# Adaptive Thresholding

Adaptive thresholding is performed in OpenCV using the *adaptiveThreshold* function:

```
B = cv2.adaptiveThreshold(G, maxValue = 255,
 adaptiveMethod = cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
  thresholdType = cv2.THRESH_BINARY,
   blockSize = 5,C = 15)
```

This will use adaptive thresholding with a 5 x 5 region for calculating each mean and a value of 15 for the constant offset.

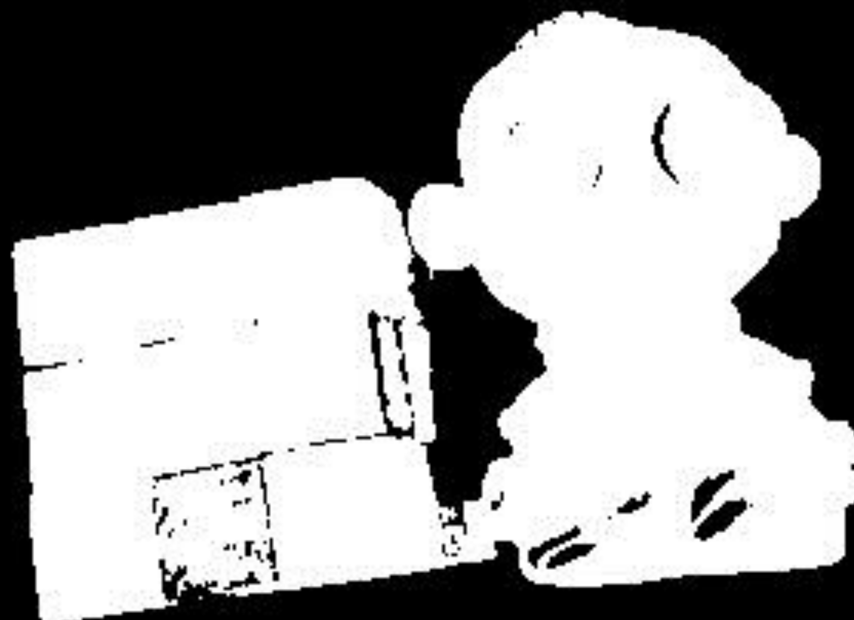
# Task : Thresholding

**Task 1: Simple Thresholding**

1. Open image “Googly.jpg”;
2. Use thresholding to separate Googly and his friend from the background;
3. Use statistical analysis to choose a threshold for better results;
4. Try multiple thresholds on different channels.

**Task 2: Adaptive Thresholding**

1. Open image “Sudoku.jpg”;
2. Use adaptive thresholding to create a black & white image;
3. Modify the region size and constant for better results.



# Conceptis SudoKu

By Dave Green

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 9 | 6 | 8 | 2 | 5 | 3 | 4 | 7 | 1 |
| 7 | 4 | 5 | 6 | 1 | 8 | 3 | 2 | 9 |
| 2 | 1 | 3 | 9 | 7 | 4 | 6 | 8 | 5 |
| 8 | 3 | 1 | 5 | 6 | 2 | 7 | 9 | 4 |
| 5 | 9 | 7 | 3 | 4 | 1 | 8 | 6 | 2 |
| 4 | 2 | 6 | 7 | 8 | 9 | 5 | 1 | 3 |
| 1 | 5 | 4 | 8 | 9 | 7 | 2 | 3 | 6 |
| 6 | 8 | 2 | 1 | 3 | 5 | 9 | 4 | 7 |
| 3 | 7 | 9 | 4 | 2 | 6 | 1 | 5 | 8 |

# Masking

# Masking

Once an ROI has been identified in a binary image or *mask*, this can be used to eliminate the areas that are not of interest.

This means that the original pixel information can be retained for the ROI while the rest of the image is ignored.

It also allows operations to be performed on specific sections of an image.

This technique is referred to as *masking*.

# ROIs

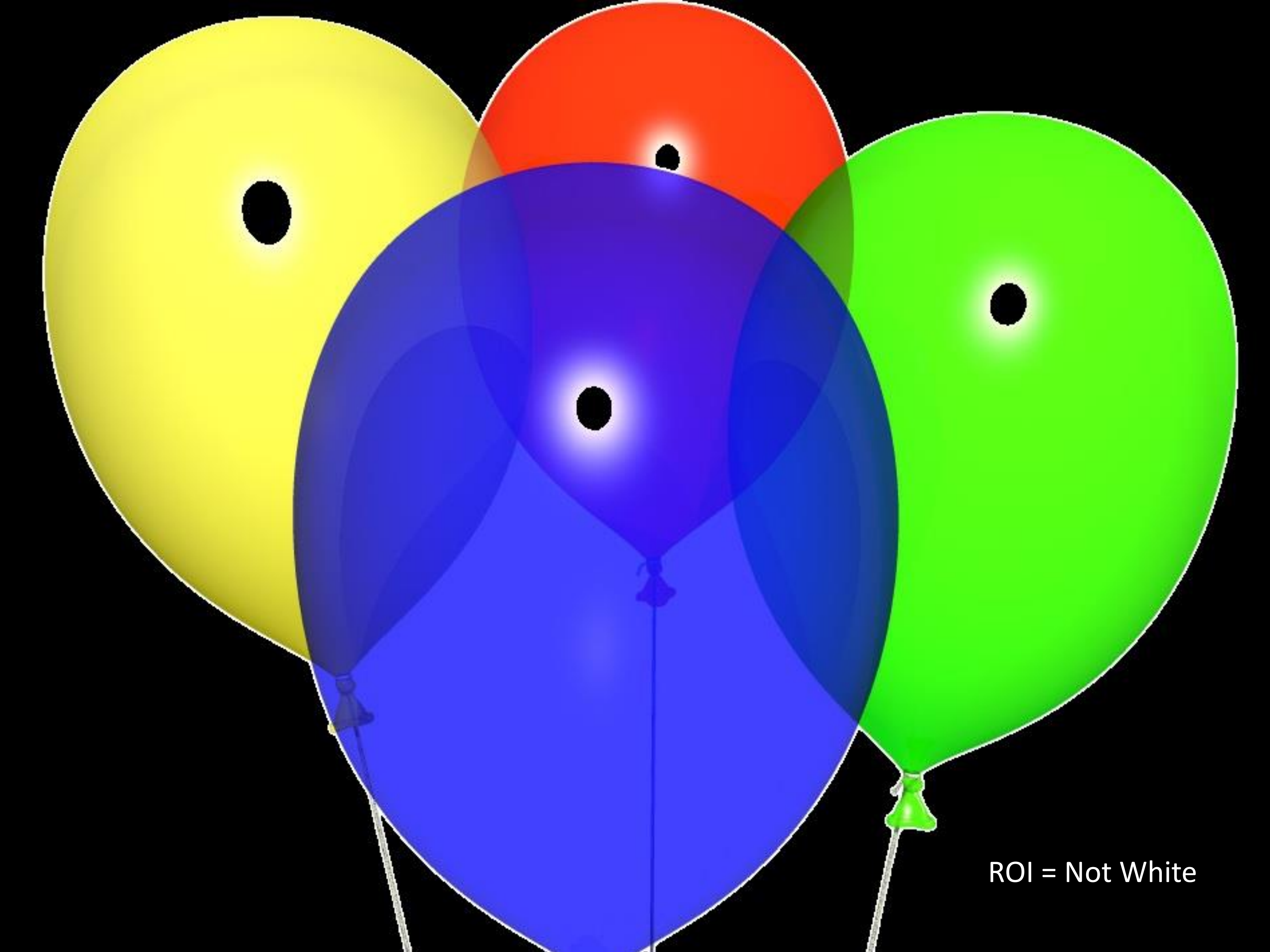
Masking is achieved by using Boolean logic.

Remember that the mask is white (or 1 in logic) while the background is black (or 0 in logic).

By combining this using a Boolean AND (similar to multiplying) we will get an image with zeroes outside the ROI:

$$ROI = I\ AND\ B$$

Where $ROI$ is the output image, $I$ is the input image and $B$ is the binary mask.


ROI = Not White

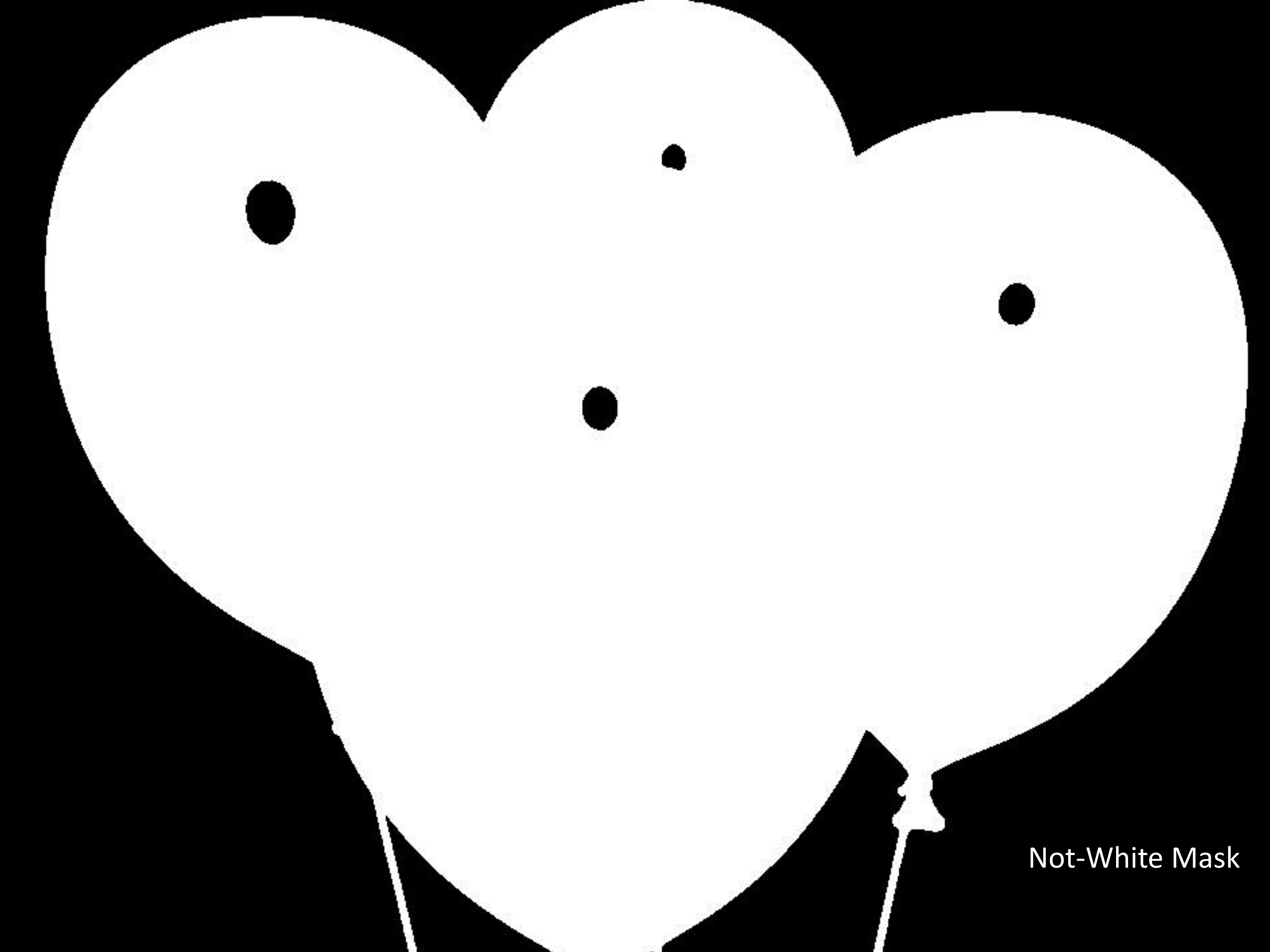
# Reverse Masks

Using the reverse of a binary mask allows holes to be cut in background images.

This allows composite images to be created by combining an ROI from one image with a background from another.

Reverse masks are achieved using a Boolean NOT:

$$B_N = NOT(B)$$



Not-White Mask

# Combining ROIs

Since the background of an ROI is black (zero), ROIs can be combined by simply adding them or by again using a Boolean OR:

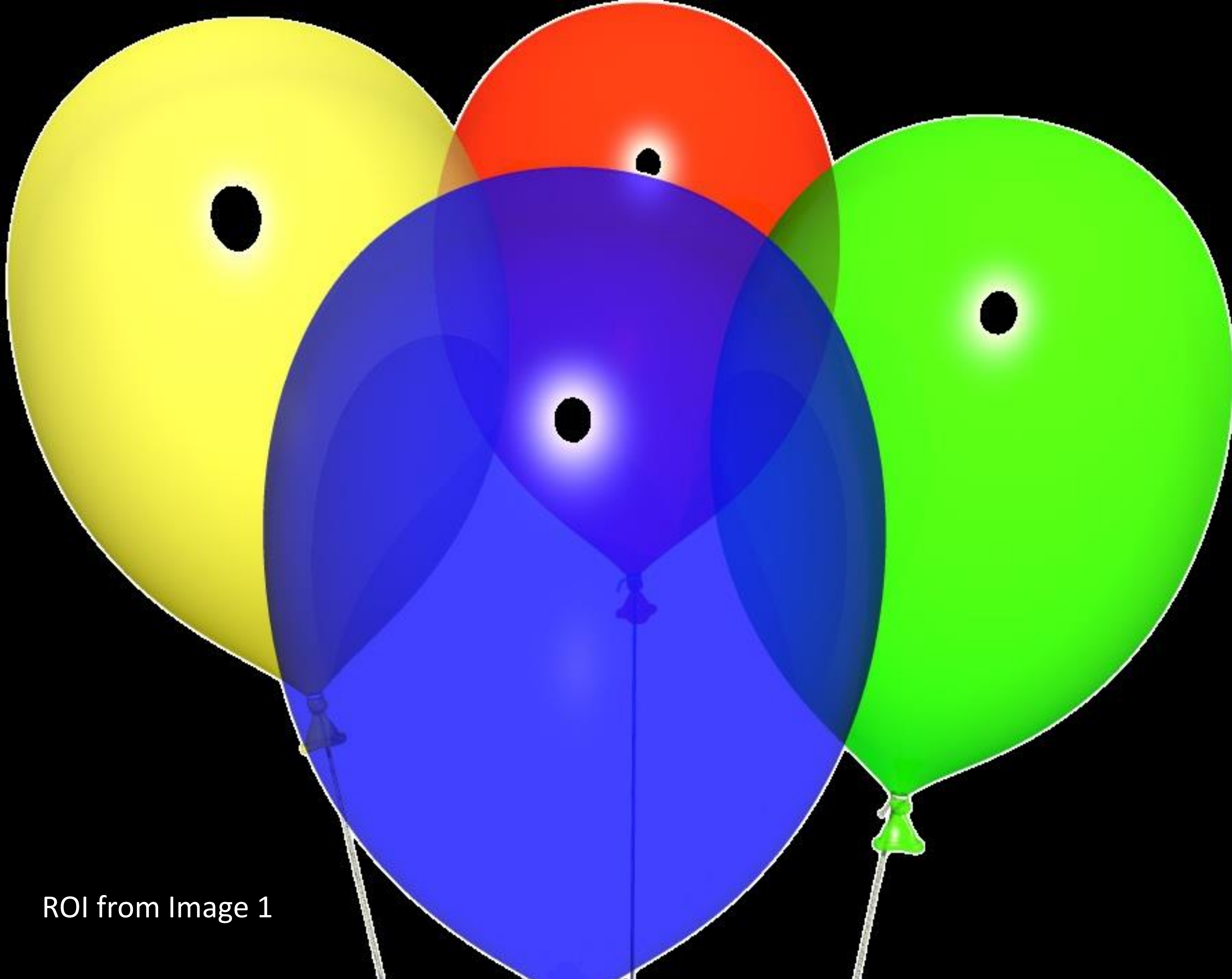
$$ROI = ROI_1 + \ ROI_2$$

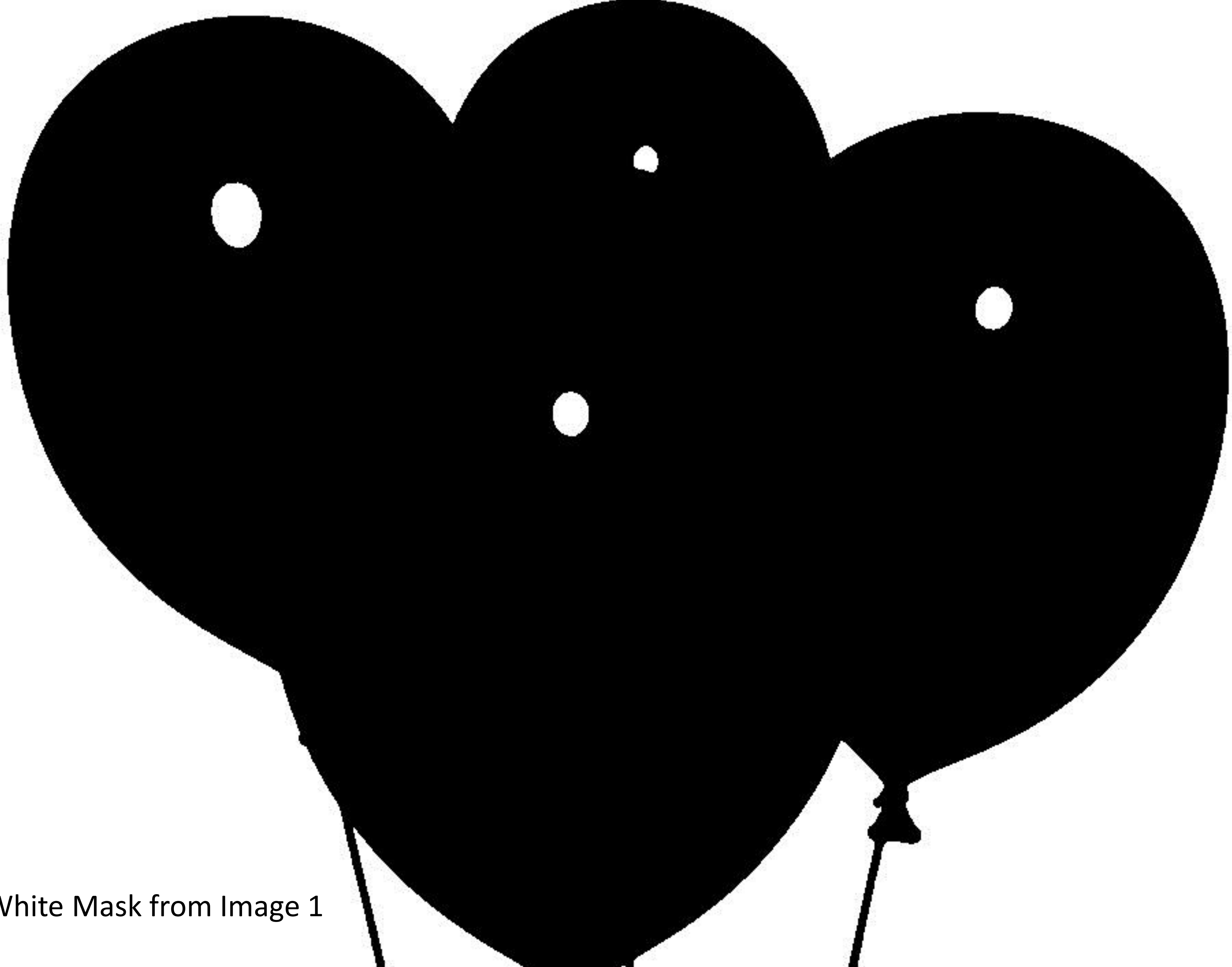
$$or$$

$$ROI = ROI_1 \ OR \ ROI_2$$

This allows an ROI from one image to be inserted in another.

*Note: The images here must be the same size.


ROI from Image 1



White Mask from Image 1


Image 2
White Mask from Image 1
Applied to Image 2


ROI from Image 1
Inserted in Image 2

# Combining Masks

Using the same Boolean logic, masks can be combined to give a total ROI made of different components.

If we have masks which extract different aspects of an image that are of interest, these can be combined using Boolean OR:

$$B = B_1 \; OR \; B_2$$

$$B_1 = white$$

$$B_2 = red$$

$$B = red\ or\ white$$

$ROI$

# In Python with OpenCV

# ROIs

A mask is simply created using thresholding, but to extract an ROI, we apply the mask using the *bitwise_and* function:

```
ROI = cv2.bitwise_and(I,I,mask=B)
```

Where `I` is the original image and `B` is the binary mask.

# Reverse Masks

Reverse masks can be created using the *bitwise_not* function:

```
BN = cv2.bitwise_not(B)
```

The same can be achieved by simply subtracting from 255:

```
BN = 255 - B
```

`B` is the original mask and `BN` is its reverse.

# Combining ROIs

ROIs can also be combined using the *bitwise_or* function:

```
ROI = cv2.bitwise_or(ROI1,ROI2)
```

The same can be achieved by simply adding:

```
ROI = ROI1 + ROI2
```

`ROI` is the combined ROI and `ROI1` and `ROI2` are the originals.

*Note: Remember, images here must be the same size.

# Combining Masks

Masks can be combined using the *bitwise_or* function:

```
B = cv2.bitwise_or(B1,B2)
```

`B` is the combined mask and `B1` and `B2` are the originals.

# Task : Masking

1. Open image “Orange.png”;
2. Using thresholding, create a mask with the orange as ROI;
3. Use this mask to extract the orange from the image;
4. Open “Water.jpg”;
5. Use the inverse of the orange mask to cut an orange-shaped hole in the water picture;
6. Combine the orange and water masked images to create a composite image.

# Morphology

# Morphology

In real images, ROIs are not so easily defined.

Thresholding will give a general segmentation but there are likely to be imperfections.

These include holes (missing ROI pixels) and blobs (extra ROI pixels).

To fix these issues, we use a technique called *morphology*.

# Morphology

Morphology means shape analysis.

In image processing, we use morphology to look at the shape of the ROI.

This can then be modified using morphological processing.

This is useful for clearly delineating the ROI and removing any imperfections in the shape.

# Morphology

Here we will look at a few basic morphological processes, including:

- Erosion
- Dilation
- Opening
- Closing

These can be combined and extended to develop morphological algorithms such as Boundary Extraction.

# Structuring Elements

To perform morphological processing, we use a *structuring element*.

This is the razor with which we will tidy the ROI.

The shape of this structuring element will determine the look of the final ROI.

A sharp shape such as a rectangle will give a blocky result while an ellipse can give a curvy result.

# Erosion

As it’s name suggests, erosion erodes the ROI.

This is achieved by shaving away the boundaries of the ROI using a razor in the shape of the structuring element.

This is useful for smoothing boundaries and eliminating blobs.

However, in eroding, it is important not to lose too much of our ROI.

# Erosion

# Erosion

Flesh Threshold Applied

# Erosion

Eroded with Rectangular SE

# Erosion

Eroded with Elliptical SE

# Dilation

As it’s name suggests, dilation dilates the ROI.

This is achieved by shaving away the *background* of the ROI using a razor in the shape of the structuring element.

This is useful for smoothing boundaries and filling gaps.

However, in dilating, it is important not to accentuate unwanted blobs.

# Dilation

# Dilation

Dilated with Rectangular SE

# Dilation

Dilated with Elliptical SE

# Opening

In both Erosion and Dilation, there are gains and losses to the ROI.

Also, in both cases, the general size of the ROI is changed.

Because of this, it is better to use a combination of the two processes to achieve the desired results.

An *Opening* is an Erosion followed by a Dilation.

This is useful for removing unwanted blobs.

# Opening

Flesh Threshold Applied

# Opening

Opened with Elliptical SE

# Closing

*Closing* is a Dilation followed by an Erosion.

This is useful for filling small holes.

# Closing

Flesh Threshold Applied

# Closing

# Final ROI

Again, both Opening and Closing have their merits but to achieve a final satisfying ROI, a combo of the two gives best results.

Good results can also be achieved by iteratively repeating morphological operations.

Finally, the size and shape of the structuring element plays a significant part in determining the quality of the ROI.

# Final ROI

Flesh Threshold Applied

# Final ROI

ROI achieved by opening after closing with a large elliptical SE

# Boundary Extraction

The Boundary of an ROI can be useful.

This is very easily extracted by subtracting the eroded mask from the original:

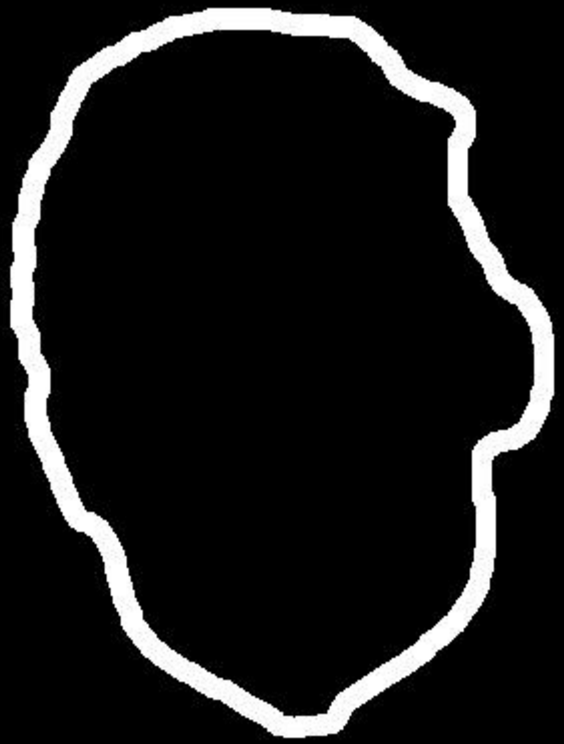
$$B = M - eroded(M)$$

The thickness of the boundary will be determined by the size of the structuring element.

# Boundary Extraction

Final ROI

# Boundary Extraction



Boundary of ROI

# Boundary Extraction

# In Python with OpenCV

# Structuring Elements

Structuring Elements can be constructed manually but are much easier created using the *getStructuringElement* function:

```
shape =
cv2.getStructuringElement(cv2.MORPH_RECT,(2,2))
```

This will create a 2x2 rectangular structuring element, `shape`.

The type `MORPH_RECT` can be replaced by `MORPH_ELLIPSE` for an elliptical element.

# Erosion

Erosion is achieved using the *erode* function:

```
NewMask = cv2.erode(OldMask,shape)
```

Where `OldMask` is the original binary mask and `NewMask` is the eroded version and `shape` is the structuring element used.

# Dilation

Dilation is achieved using the *dilate* function:

```
NewMask = cv2.dilate(OldMask,shape)
```

Where `OldMask` is the original binary mask and `NewMask` is the dilated version and `shape` is the structuring element used.

# Opening

Opening is part of the *morphologyEx* function:

```
NewMask = 
cv2.morphologyEx(OldMask,cv2.MORPH_OPEN,shape)
```

Where `OldMask` is the original binary mask and `NewMask` is the opened version and `shape` is the structuring element used.

# Closing

Closing is also part of the *morphologyEx* function:

```
NewMask = cv2.morphologyEx(OldMask,cv2.MORPH_CLOSE,shape)
```

Where `OldMask` is the original binary mask and `NewMask` is the closed version and `shape` is the structuring element used.

# Boundary Extraction

Boundary extraction is included in the *morphologyEx* function as MORPH_GRADIENT:

```
Boundary =
cv2.morphologyEx(mask,cv2.MORPH_GRADIENT,shape)
```

Where `mask` is the original binary mask and `Boundary` is the extracted boundary (as a binary mask) and `shape` is the structuring element used.

# Task : Morphology

1. Open “Trump.jpg”;
2. Use an appropriate colour space and range to segment out areas of flesh (you will need to research this);
3. Use morphological processes to clean up the ROI.

**Advanced Task:**

Place Trump’s face somewhere entertaining!

# Task : Morphology

# Review

**In this section you have learned about :**

- Binary Images
- Thresholding
- Masking
- Morphology

These topics were **implemented** and **tested** in Python with OpenCV.