

Методы анализа и обработки изображения

ЛР5 - Сегментация изображений в Python

Студентка гр. РИМ-181226

Бабайлова Маргарита Леонидовна

In [1]:

```
import numpy as np
from skimage import data
from skimage.feature import canny
from scipy import ndimage as ndi
import matplotlib.pyplot as plt
from skimage.filters import sobel, try_all_threshold
from skimage.filters.thresholding import threshold_adaptive, threshold_local
from skimage.morphology import watershed
from scipy import ndimage as ndi
from skimage.color import label2rgb, rgb2gray
from skimage.filters import sobel
from skimage.morphology import watershed
from skimage import io
from skimage.segmentation import active_contour, random_walker, slic, felzenszwalb
from skimage.draw import circle_perimeter
import cv2
import warnings
warnings.filterwarnings("ignore")

%matplotlib inline
```

In [2]:

```
import matplotlib.pyplot as plt
from skimage.io import imshow

def plot2(img1, img2, title1, title2):
    fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(12, 4),
                                  sharex=True, sharey=True)
    ax1.imshow(img1, cmap=plt.cm.gray)
    ax1.axis('off')
    ax1.set_title(title1)

    ax2.imshow(img2, cmap=plt.cm.gray)
    ax2.axis('off')
    ax2.set_title(title2)

    fig.tight_layout()

    plt.show()

def plot3(img1, img2, img3, title1, title2, title3):
    fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(12, 4),
                                      sharex=True, sharey=True)
    ax1.imshow(img1, cmap=plt.cm.gray)
    ax1.axis('off')
    ax1.set_title(title1)

    ax2.imshow(img2, cmap=plt.cm.gray)
    ax2.axis('off')
    ax2.set_title(title2)

    ax3.imshow(img3, cmap=plt.cm.gray)
    ax3.axis('off')
    ax3.set_title(title3)

    fig.tight_layout()

    plt.show()

def plot4(img1, img2, img3, img4, title1, title2, title3, title4):
    fig, (ax1, ax2, ax3, ax4) = plt.subplots(nrows=1, ncols=4, figsize=(14, 4),
                                             sharex=True, sharey=True)
    ax1.imshow(img1, cmap=plt.cm.gray)
    ax1.axis('off')
    ax1.set_title(title1)

    ax2.imshow(img2, cmap=plt.cm.gray)
    ax2.axis('off')
    ax2.set_title(title2)

    ax3.imshow(img3, cmap=plt.cm.gray)
    ax3.axis('off')
    ax3.set_title(title3)

    ax4.imshow(img4, cmap=plt.cm.gray)
    ax4.axis('off')
    ax4.set_title(title4)

    fig.tight_layout()
```

```
plt.show()
```

In [3]:

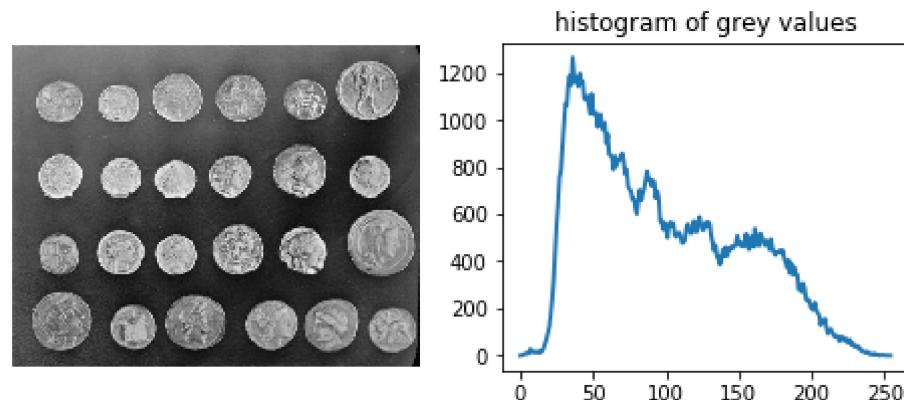
```
coins = data.coins()  
hist = np.histogram(coins, bins=np.arange(0, 256))
```

In [4]:

```
fig, axes = plt.subplots(1, 2, figsize=(8, 3))  
axes[0].imshow(coins, cmap=plt.cm.gray, interpolation='nearest')  
axes[0].axis('off')  
axes[1].plot(hist[1][:-1], hist[0], lw=2)  
axes[1].set_title('histogram of grey values')
```

Out[4]:

```
Text(0.5, 1.0, 'histogram of grey values')
```

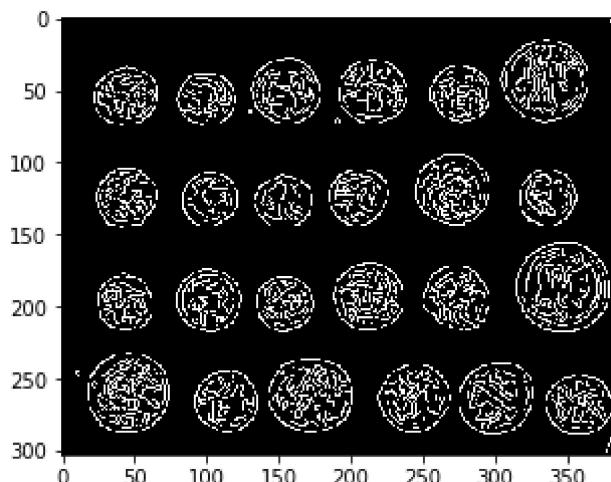


In [5]:

```
edges = canny(coins/255.)  
plt.imshow(edges, cmap=plt.cm.gray)
```

Out[5]:

```
<matplotlib.image.AxesImage at 0x226c5d7c1d0>
```

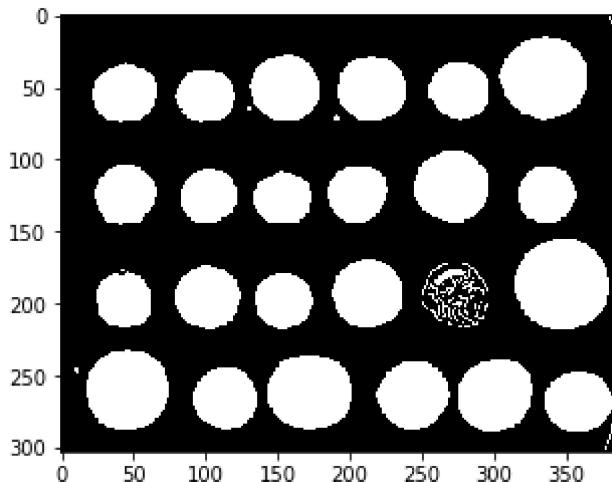


In [6]:

```
fill_coins = ndi.binary_fill_holes(edges)
plt.imshow(fill_coins, cmap=plt.cm.gray)
```

Out[6]:

```
<matplotlib.image.AxesImage at 0x226c5de9240>
```



In [7]:

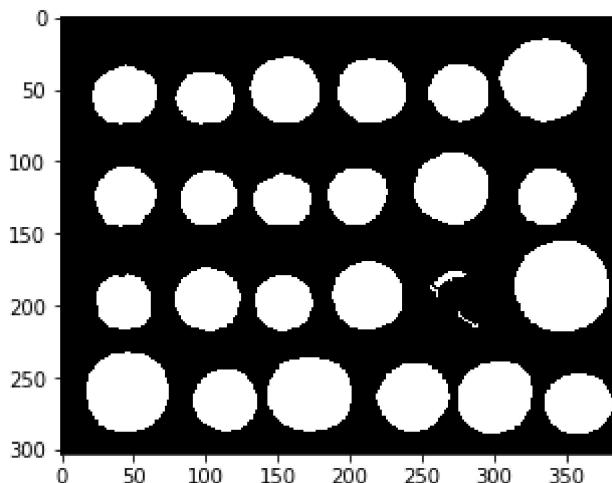
```
label_objects, nb_labels = ndi.label(fill_coins)
sizes = np.bincount(label_objects.ravel())
mask_sizes = sizes > 20
mask_sizes[0] = 0
coins_cleaned = mask_sizes[label_objects]
```

In [8]:

```
plt.imshow(coins_cleaned, cmap=plt.cm.gray)
```

Out[8]:

```
<matplotlib.image.AxesImage at 0x226c5e55128>
```



In [9]:

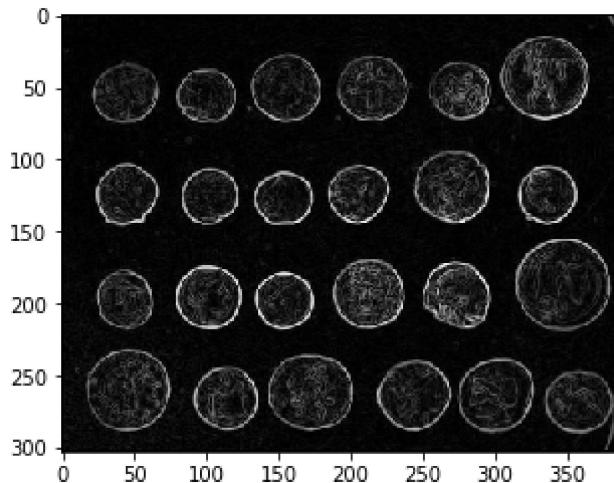
```
markers = np.zeros_like(coins)
markers[coins < 30] = 1
markers[coins > 150] = 2
```

In [10]:

```
elevation_map = sobel(coins)
plt.imshow(elevation_map, cmap=plt.cm.gray)
```

Out[10]:

```
<matplotlib.image.AxesImage at 0x226c5eb7278>
```



In [11]:

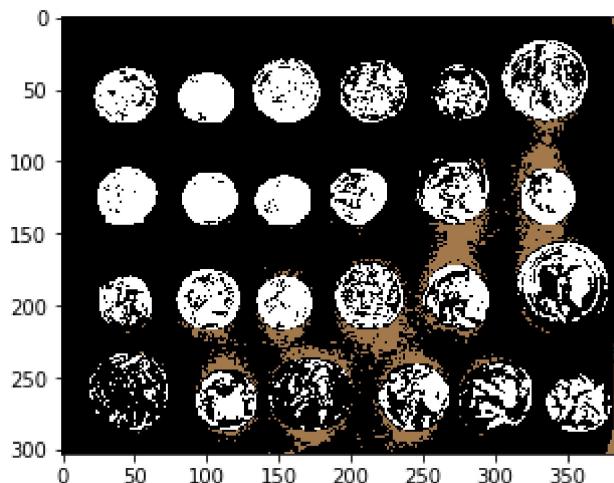
```
markers = np.zeros_like(coins)
markers[coins < 30] = 1
markers[coins > 150] = 2
```

In [12]:

```
plt.imshow(markers, cmap=plt.cm.cubehelix)
```

Out[12]:

```
<matplotlib.image.AxesImage at 0x226c5f20198>
```



In [13]:

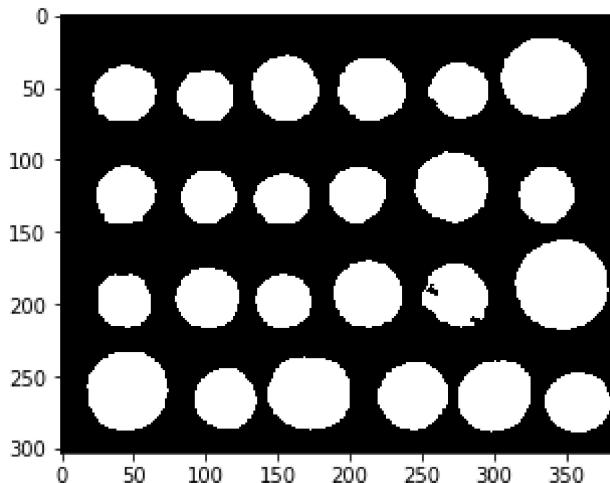
```
segmentation = watershed(elevation_map, markers)
```

In [14]:

```
plt.imshow(segmentation, cmap=plt.cm.gray)
```

Out[14]:

```
<matplotlib.image.AxesImage at 0x226c5f88208>
```



In [15]:

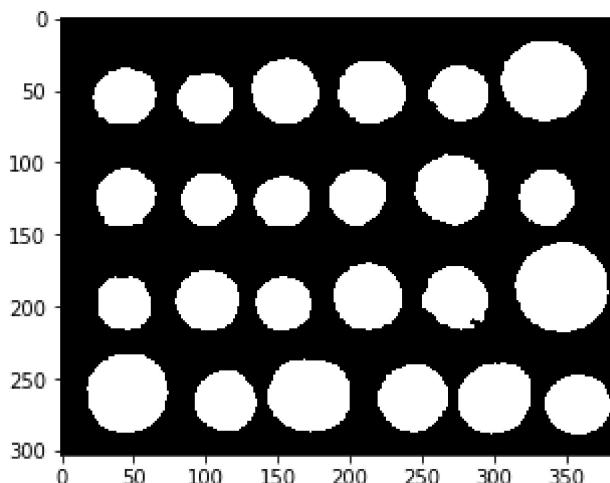
```
segmentation = ndi.binary_fill_holes(segmentation - 1)
```

In [16]:

```
plt.imshow(segmentation, cmap=plt.cm.gray)
```

Out[16]:

```
<matplotlib.image.AxesImage at 0x226c5fef1d0>
```



In [17]:

```
labeled_coins, _ = ndi.label(segmentation)
```

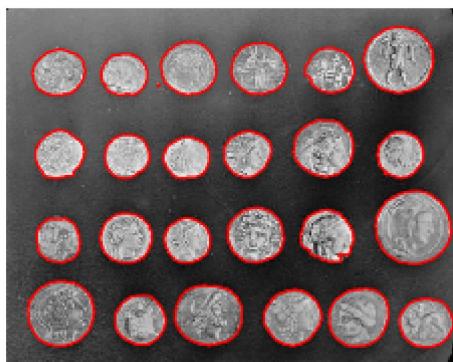
In [18]:

```
image_label_overlay = label2rgb(labeled_coins, image=coins)

fig, axes = plt.subplots(1, 2, figsize=(8, 3), sharey=True)
axes[0].imshow(coins, cmap=plt.cm.gray, interpolation='nearest')
axes[0].contour(segmentation, [0.5], linewidths=1.2, colors='r')
axes[1].imshow(image_label_overlay, interpolation='nearest')

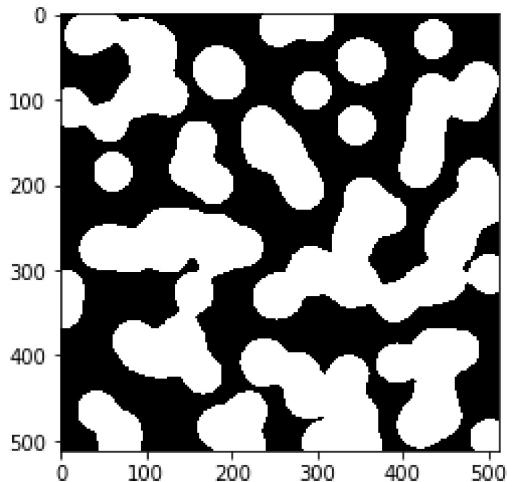
for a in axes:
    a.axis('off')
    a.set_adjustable('box-forced')

plt.tight_layout()
```



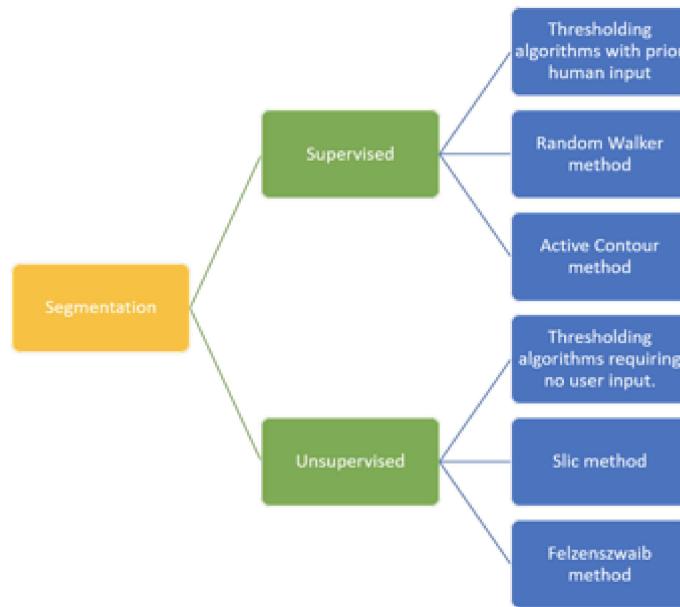
In [19]:

```
image = data.binary_blobs()
plt.imshow(image, cmap = 'gray');
```



Сегментация изображений – это процесс разделения цифрового изображения на несколько сегментов, чтобы упростить и / или изменить представление изображения на что-то более значимое и более простое для анализа.

Классификация алгоритмов сегментации изображений приведена на рисунке ниже.



Сегментация с учителем: для сегментации используются данные из человеческого ввода.

Сегментация без учителя: предварительных знаний не требуется, алгоритмы пытаются автоматически разделить изображения на значимые области. Пользователь по-прежнему может настраивать определенные параметры для получения желаемых

Обзор методов сегментации изображений в библиотеке scikit-image: <https://habr.com/ru/post/441006> (<https://habr.com/ru/post/441006>)

Обзор изображений в Python

Импорт изображений из библиотеки skimage модуль данных `skimage` содержит несколько встроенных примеров наборов данных в формате jpeg или png.

In [20]:

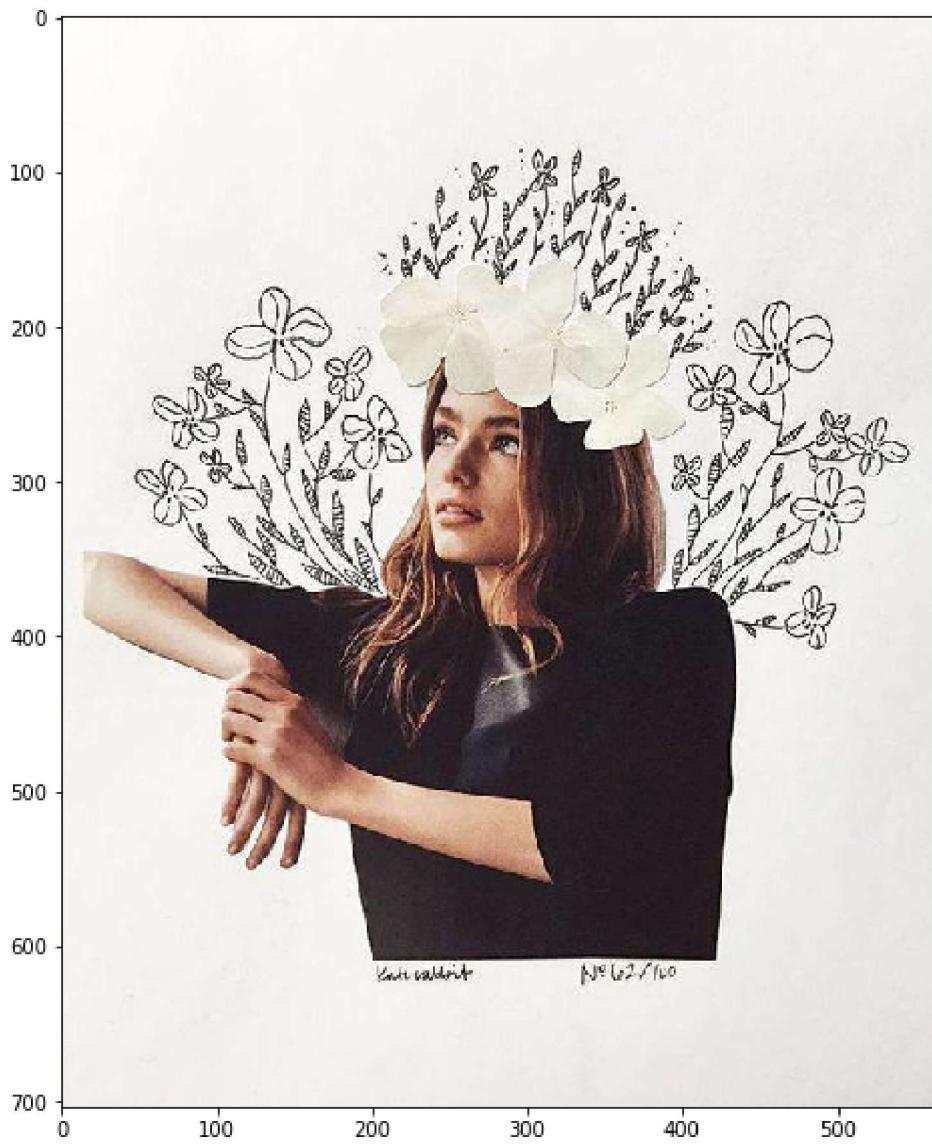
```
image = data.rocket()
plt.figure(figsize = (15, 10))
plt.imshow(image);
```



Импорт изображения из внешнего источника

In [21]:

```
image = io.imread('girl.jpg')
plt.figure(figsize = (15, 10))
plt.imshow(image);
```



Загрузка нескольких изображений

In [22]:

```
images = io.ImageCollection('images/*.jpg')
print('Тип переменной images:', type(images), '\nДлина переменной images (кол-во изображений):', len(images))
print('Изображения:', images.files)
```

Тип переменной images: <class 'skimage.io.collection.ImageCollection'>

Длина переменной images (кол-во изображений в коллекции): 0

Изображения: []

Сохранение изображений

In [23]:

```
io.imsave('girl.jpg', image)
```

Сегментация изображений

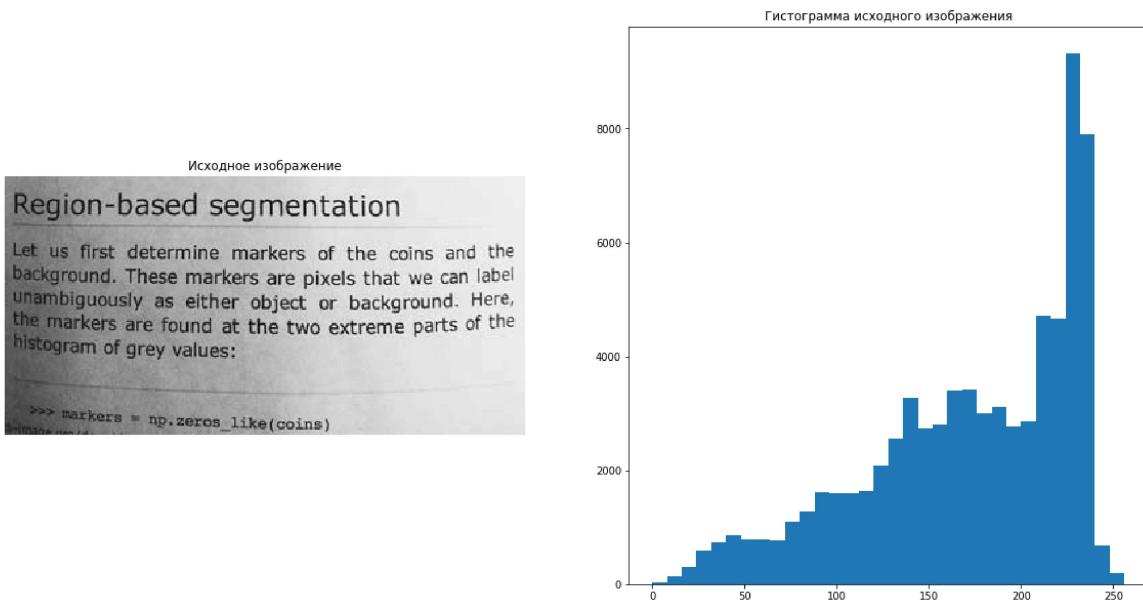
Возможно, реально выбрать значение из исходного изображение, которое даст разумную сегментацию без сложных алгоритмов. Для этого используется гистограмма.

Гистограмма показывает количество пикселей в изображении с различными значениями интенсивности, найденными в этом изображении. Проще говоря, гистограмма — это график, на котором ось X показывает все значения, которые есть на изображении, а ось Y показывает частоту этих значений.

In [24]:

```
text = data.page()

fig, axes = plt.subplots(1, 2, figsize = (20, 10))
axes[0].imshow(text, cmap = 'gray')
axes[0].set_title('Исходное изображение');
axes[0].axis('off')
axes[1].hist(text.ravel(), bins = 32, range = [0, 256])
axes[1].set_title('Гистограмма исходного изображения');
```



Данный пример – 8-битное изображение, поэтому по оси X 256 возможных значений. По гистограмме видно, что существует концентрация довольно светлых пикселей (0: черный, 255: белый). Скорее всего, это довольно светлый текстовый фон, но остальное немного размыто.

Идеальная гистограмма сегментации была бы бимодальной, чтобы было возможно выбрать число посередине.

Контролируемый порог Thresholding самый простой способ отделить объекты от фона, выбрав пиксели выше или ниже определенного порога. Используется, когда нужно сегментировать объекты по их фону.

Контролируемый порог – пороговое значение выбирается самостоятельно.

In [25]:

```
fig, axes = plt.subplots(1, 3, figsize = (20, 10))
axes[0].imshow(text > 50, cmap = 'gray')
axes[0].axis('off')
axes[0].set_title('Порог 50');
axes[1].imshow(text > 70, cmap = 'gray')
axes[1].axis('off')
axes[1].set_title('Порог 70');
axes[2].imshow(text > 120, cmap = 'gray')
axes[2].axis('off')
axes[2].set_title('Порог 120');
```

Порог 50
Region-based segmentation

Let us first determine markers of the coins and the background. These markers are pixels that we can label unambiguously as either object or background. Here, the markers are found at the two extreme parts of the histogram of grey values:

```
>>> markers = np.zeros_like(coins)
```

Порог 70
Region-based segmentation

Let us first determine markers of the coins and the background. These markers are pixels that we can label unambiguously as either object or background. Here, the markers are found at the two extreme parts of the histogram of grey values:

```
>>> markers = np.zeros_like(coins)
```

Порог 120

Region-based segmentation

Let us first determine markers of the coins and the background. These markers are pixels that we can label unambiguously as either object or background. Here, the markers are found at the two extreme parts of the histogram of grey values:

```
>>> markers = np.zeros_like(coins)
```

Достойных результатов не получено.

Неконтролируемый порог: на основе определенных методов (isodata, li, mean и т.д.) порог определяется автоматически.

In [26]:

```
fig, axes = try_all_threshold(text, figsize = (20, 10), verbose = False)
plt.show()
```

Original

Region-based segmentation

Let us first determine markers of the coins and the background. These markers are pixels that we can label unambiguously as either object or background. Here, the markers are found at the two extreme parts of the histogram of grey values:

```
>>> markers = np.zeros_like(coins)
```

Li

Region-based segmentation

Let us first determine markers of the coins and the background. These markers are pixels that we can label unambiguously as either object or background. Here, the markers are found at the two extreme parts of the histogram of grey values:

```
>>> markers = np.zeros_like(coins)
```

Minimum

Region-based segmentation

Let us first determine markers of the coins and the background. These markers are pixels that we can label unambiguously as either object or background. Here, the markers are found at the two extreme parts of the histogram of grey values:

Triangle

Region-based segmentation

Let us first determine markers of the coins and the background. These markers are pixels that we can label unambiguously as either object or background. Here, the markers are found at the two extreme parts of the histogram of grey values:

Isodata

Region-based segmentation

Let us first determine markers of the coins and the background. These markers are pixels that we can label unambiguously as either object or background. Here, the markers are found at the two extreme parts of the histogram of grey values:

```
>>> markers = np.zeros_like(coins)
```

Mean

Region-based segmentation

Let us first determine markers of the coins and the background. These markers are pixels that we can label unambiguously as either object or background. Here, the markers are found at the two extreme parts of the histogram of grey values:

```
>>> markers = np.zeros_like(coins)
```

Otsu

Region-based segmentation

Let us first determine markers of the coins and the background. These markers are pixels that we can label unambiguously as either object or background. Here, the markers are found at the two extreme parts of the histogram of grey values:

```
>>> markers = np.zeros_like(coins)
```

Yen

Region-based segmentation

Let us first determine markers of the coins and the background. These markers are pixels that we can label unambiguously as either object or background. Here, the markers are found at the two extreme parts of the histogram of grey values:

```
>>> markers = np.zeros_like(coins)
```

In [27]:

```
text_threshold = threshold_local(text, block_size = 51, offset = 10)
plt.figure(figsize = (10, 10))
plt.axis('off')
plt.imshow(text > text_threshold, cmap = 'gray');
```

Region-based segmentation

Let us first determine markers of the coins and the background. These markers are pixels that we can label unambiguously as either object or background. Here, the markers are found at the two extreme parts of the histogram of grey values:

```
>>> markers = np.zeros_like(coins)
```

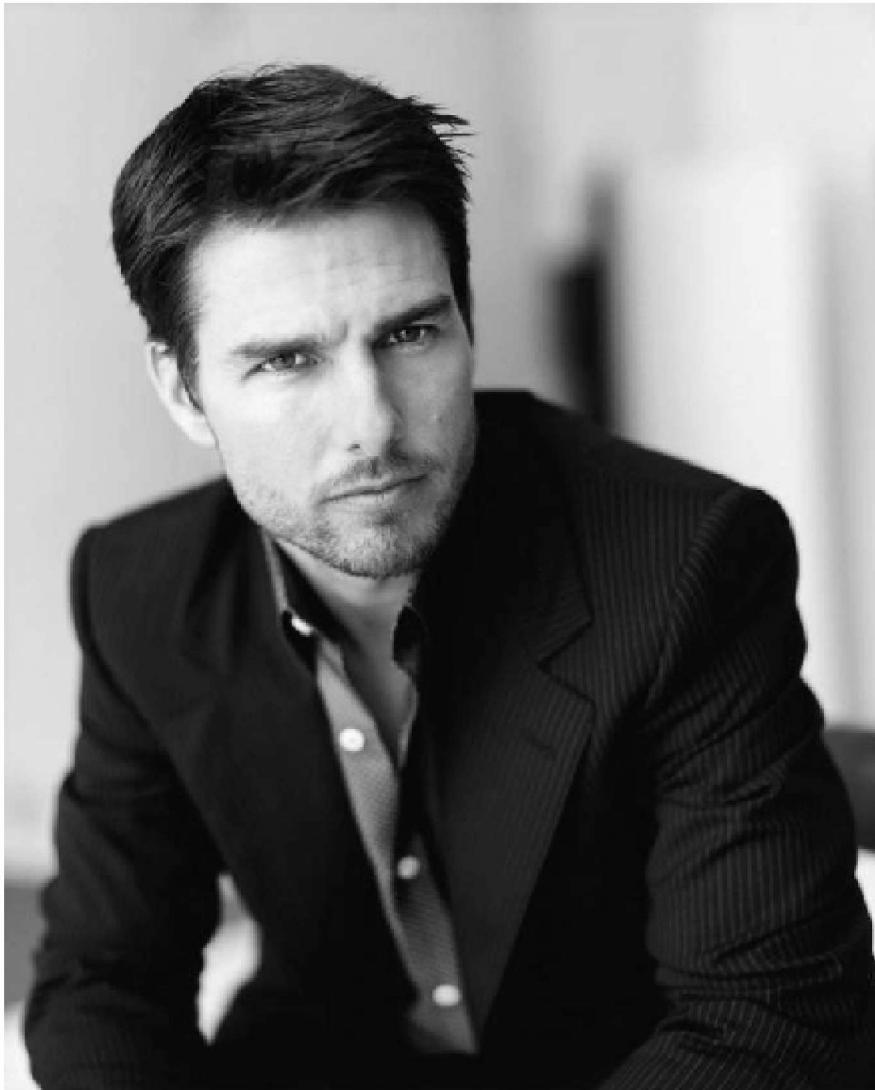
С помощью локального порога достигнут хороший результат.

Сегментация с алгоритмом для модели с учителем.

Для сегментации высококонтрастного изображения нужны более совершенные инструменты, чем thresholding.

In [28]:

```
image = image = io.imread('man.jpg') # загрузим изображение
image_gray = rgb2gray(image) # преобразуем в оттенки серого
plt.figure(figsize = (10, 10))
plt.axis('off')
plt.imshow(image_gray, cmap = 'gray');
```



Активная контурная сегментация

Сегментация активного контура ("змея") инициализируется с использованием определенного пользователем контура или линии вокруг интересующей области, а затем этот контур медленно сжимается и притягивается или отталкивается от света и краев.

В данном примере для инициализации змеи вокруг головы человека нарисован круг.

In [29]:

```
def circle_points(resolution, center, radius):
    """
    Generate points which define a circle on an image. Centre refers to the centre of the circle
    """
    radians = np.linspace(0, 2 * np.pi, resolution)
    c = center[1] + radius*np.cos(radians)#polar co-ordinates
    r = center[0] + radius*np.sin(radians)

    return np.array([c, r]).T
# Exclude last point because a closed path should not have duplicate points
points = circle_points(450, [235,245], 200)[:-1]

fig, axes = plt.subplots(1, 1, figsize = (10, 10))
axes.plot(points[:, 0], points[:, 1], '--r', lw=3)
axes.axis('off')
axes.imshow(image);
```

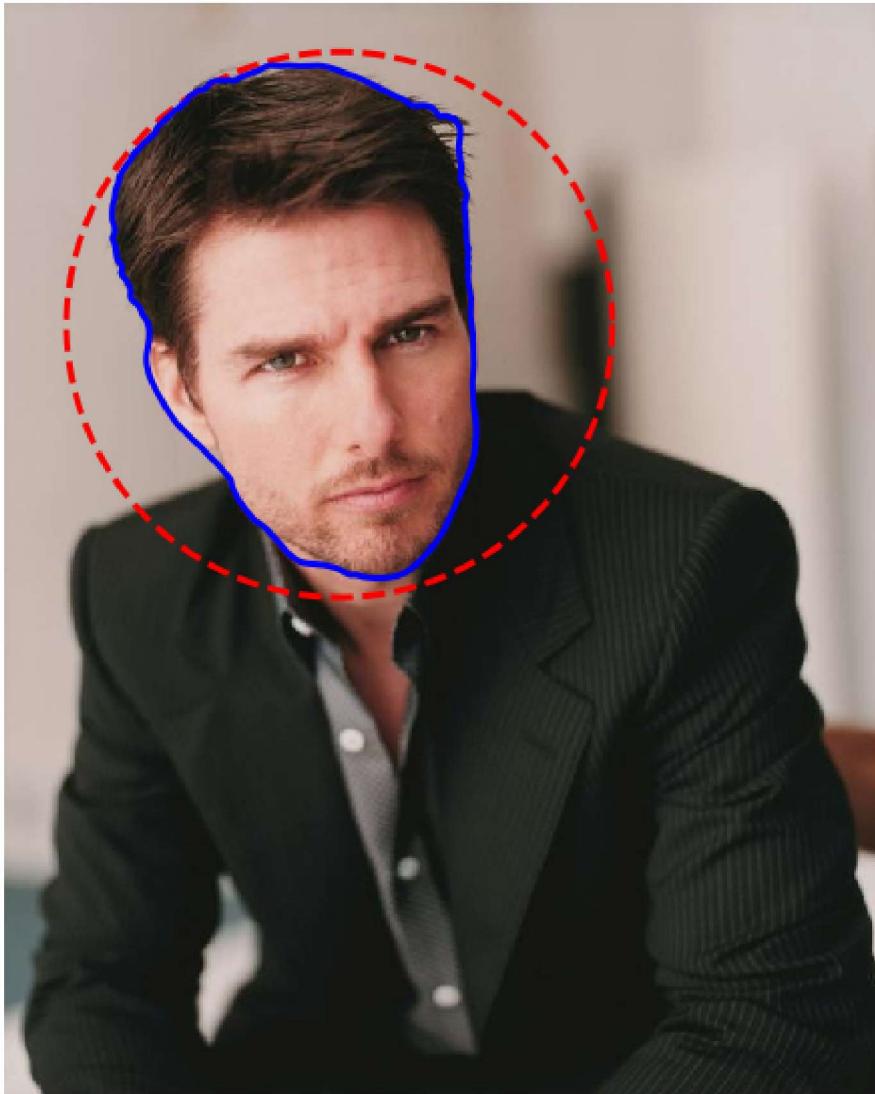


Алгоритм сегментирует лицо человека от остальной части изображения, подгоняя замкнутую кривую к краям лица.

In [30]:

```
snake = active_contour(image_gray, points, alpha = 0.06, beta = 0.3)

fig, axes = plt.subplots(1, 1, figsize = (10, 10))
axes.plot(points[:, 0], points[:, 1], '--r', lw=3)
axes.axis('off')
axes.plot(snake[:, 0], snake[:, 1], '-b', lw=3)
axes.imshow(image);
```

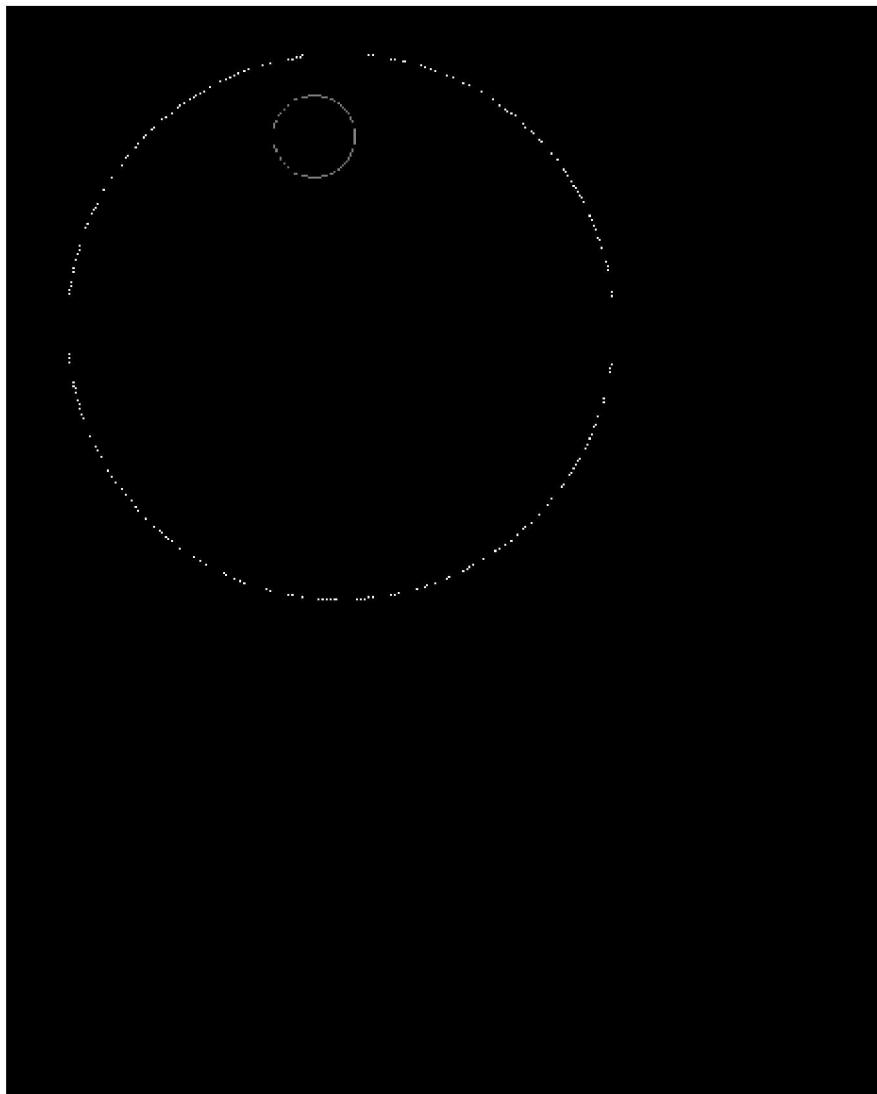


Сегментация случайного прохождения (Random walker algorithm) Пользователь интерактивно помечает небольшое количество пикселей известными метками, например, «объект» и «фон». Для немаркированных пикселей вычисляется вероятность того, к какой из меток он принадлежит. Таким образом, проводится сегментация изображения.

Здесь использованы предыдущие значения из данного примера. Инициализации могут быть разные, но для простоты выбраны круги. Алгоритм случайного прохождения принимает метки в качестве ввода. Таким образом, объявлен большой круг, охватывающий все лицо человека, и еще один меньший круг около середины лица.

In [31]:

```
image_labels = np.zeros(image_gray.shape, dtype = np.uint8)
indices = circle_perimeter(95, 225, 30)
image_labels[indices] = 1
image_labels[points[:, 1].astype(np.int), points[:, 0].astype(np.int)] = 2
plt.figure(figsize = (10,10))
plt.axis('off')
plt.imshow(image_labels, cmap = 'gray');
```



In [32]:

```
image_segmented = random_walker(image_gray, image_labels, beta = 5000)

fig, axes = plt.subplots(1, 1, figsize = (10, 10))
axes.imshow(image_gray, cmap = 'gray')
axes.axis('off')
axes.imshow(image_segmented == 1, alpha = 0.3);
```



Сегментация без учителя

Простая линейно-итеративная кластеризация (Simple Linear Iterative Clustering или SLIC) SLIC использует алгоритм машинного обучения K-means. Он принимает все значения пикселей изображения и пытается разделить их на заданное количество подобластей.

In [33]:

```
image_slic = slic(image, n_segments = 155)
```

Все, что нужно сделать, это просто установить для каждого найденного сегмента среднее значение, что делает его больше похожим на изображение.

In [34]:

```
plt.figure(figsize = (10, 10))
plt.axis('off')
plt.imshow(label2rgb(image_slic, image, kind = 'avg'));
```



Felzenszwalb использует алгоритм кластеризации minimum-spanning tree clustering. Felzenszwaib не сообщает точное количество кластеров, на которые будет разделено изображение. Он будет генерировать столько кластеров, сколько он считает нужным для этого.

In [35]:

```
image_felzenszwalb = felzenszwalb(image)

plt.figure(figsize = (10, 10))
plt.imshow(image_felzenszwalb, cmap = 'gray')
plt.axis('off');
```



На рисунке слишком много регионов. Определим количество уникальных сегментов.

In [36]:

```
np.unique(image_felzenszwalb).size
```

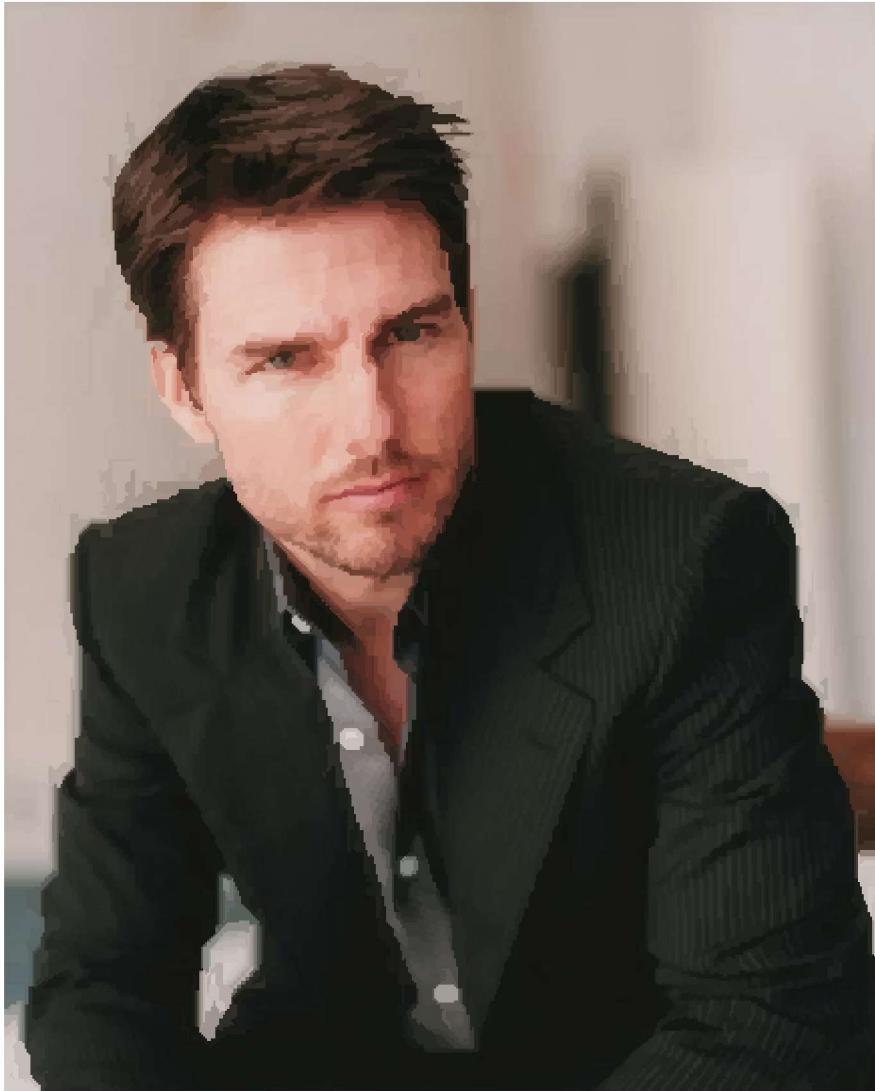
Out[36]:

7658

Теперь их необходимо перекрасить, используя среднее значение по сегменту, как в алгоритме SLIC.

In [37]:

```
image_felzenswalb_colored = label2rgb(image_felzenswalb, image, kind = 'avg')
plt.figure(figsize = (10, 10))
plt.imshow(image_felzenswalb_colored)
plt.axis('off');
```



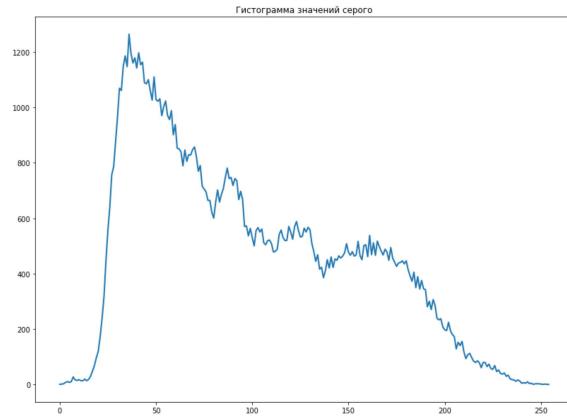
Сегментация изображений

https://code-examples.net/ru/docs/scikit_image/user_guide/tutorial_segmentation (https://code-examples.net/ru/docs/scikit_image/user_guide/tutorial_segmentation)

Использовано изображение `coins` из `skimage.data`, на котором выделяются несколько монет на темном фоне. Сегментация монет не может быть выполнена из гистограммы значений серого, поскольку фон имеет достаточно серого уровня с монетами, что сегментация порога недостаточна.

In [38]:

```
img_coins = data.coins()
hist = np.histogram(img_coins, bins = np.arange(0, 256))
fig, axes = plt.subplots(1, 2, figsize = (30, 10))
axes[0].imshow(img_coins, cmap = plt.cm.gray, interpolation = 'nearest')
axes[0].axis('off')
axes[1].plot(hist[1][:-1], hist[0], lw = 2)
axes[1].set_title('Гистограмма значений серого');
```

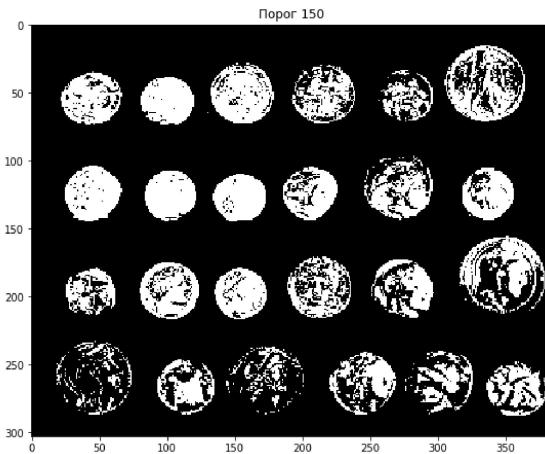
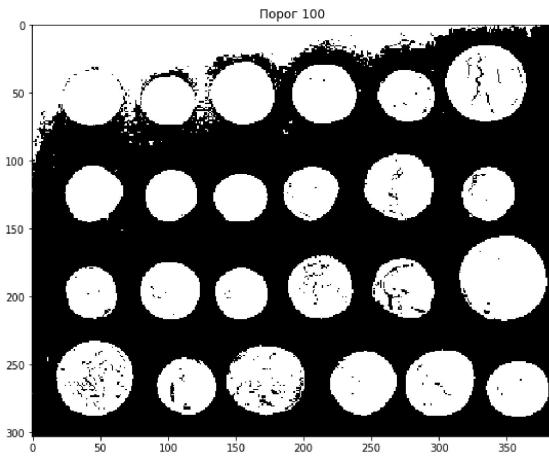


Thresholding

В данном случае пороговое изображение приводит либо к отсутствию значительных частей монет, либо к объединению частей фона с монетами. Это связано с неоднородным освещением изображения.

In [39]:

```
fig, axes = plt.subplots(1, 2, figsize = (20, 10))
axes[0].imshow(img_coins > 100, cmap = plt.cm.gray)
axes[1].imshow(img_coins > 150, cmap = plt.cm.gray)
axes[0].set_title('Порог 100')
axes[1].set_title('Порог 150');
```



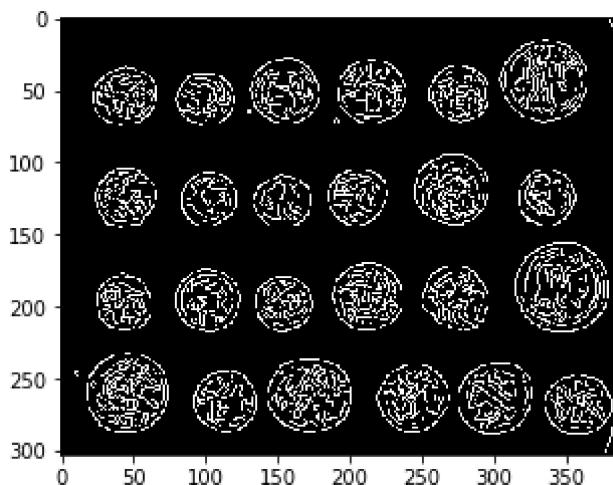
Сегментация на основе краев

Использование градиента, а не серых значений, может дать лучший результат.

Для обнаружения краев, которые заключают монеты, использован детектор canny skimage.feature.canny, основанный на вычислении интенсивности градиентов для удаления шума, присутствующего на изображении.

In [40]:

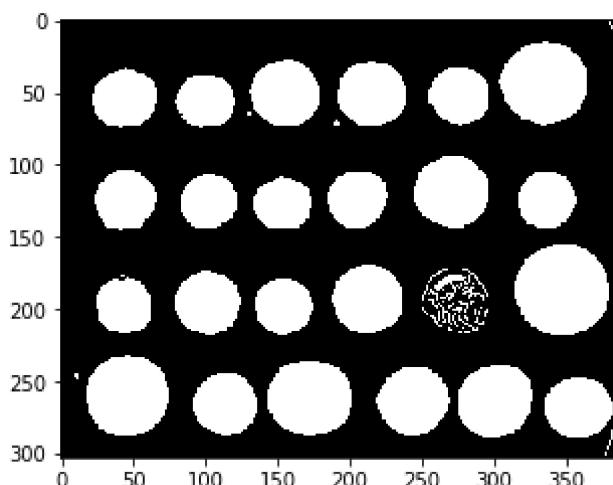
```
edges = canny(img_coins / 255.)
plt.imshow(edges, cmap = plt.cm.gray);
```



Теперь внутреннюю часть монет необходимо заполнить с помощью функции `ndi.binary_fill_holes`, которая использует математическую морфологию для заполнения отверстий.

In [41]:

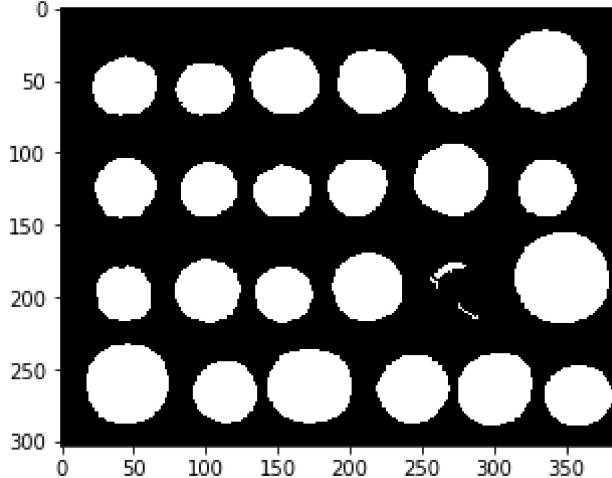
```
fill_coins = ndi.binary_fill_holes(edges)
plt.imshow(fill_coins, cmap = plt.cm.gray);
```



Большинство монет хорошо сегментированы на заднем плане. Маленькие объекты из фона можно легко удалить с помощью `ndi.label` для удаления объектов, меньших небольшого порога.

In [42]:

```
label_objects, nb_labels = ndi.label(fill_coins)
sizes = np.bincount(label_objects.ravel())
mask_sizes = sizes > 20
mask_sizes[0] = 0
coins_cleaned = mask_sizes[label_objects]
plt.imshow(coins_cleaned, cmap = plt.cm.gray);
```



Однако сегментация не очень удовлетворительна, так как одна из монет не была полностью сегментирована. Причина в том, что контур, который был получен от детектора canny, не был полностью закрыт, поэтому функция заполнения не заполнила внутреннюю часть монеты.

Поэтому этот метод сегментации не очень устойчив: если был пропущен один пиксель контура объекта, заполнить его невозможно.

Сегментация по регионам

Сначала необходимо определить маркеры монет и фона – пиксели, которые можно однозначно обозначать как объект или фон. Здесь маркеры находятся в двух крайних частях гистограммы серых значений.

In [43]:

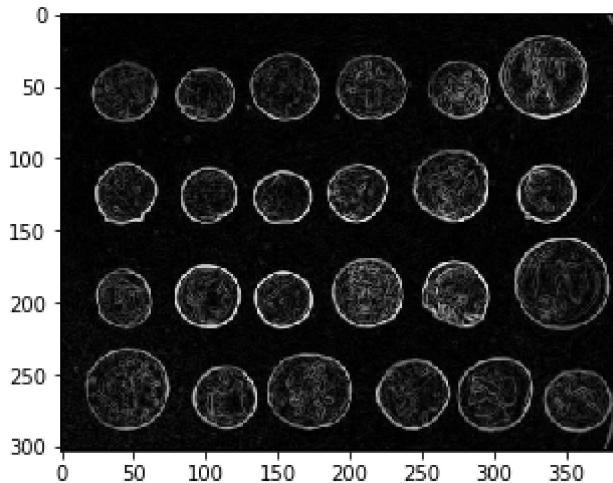
```
markers = np.zeros_like(img_coins)
markers[img_coins < 30] = 1
markers[img_coins > 150] = 2
```

Эти маркеры используются для сегментации водораздела. Название водораздела происходит по аналогии с гидрологией. Преобразование водораздела наводняет изображение высоты, начиная с маркеров, чтобы определить водосборные бассейны этих маркеров. Водораздельные линии разделяют эти водосборные бассейны и соответствуют желаемой сегментации.

Выбор карты высот имеет решающее значение для хорошей сегментации. Здесь амплитуда градиента обеспечивает хорошую карту места, для вычисления амплитуды градиента используется оператор Собеля.

In [44]:

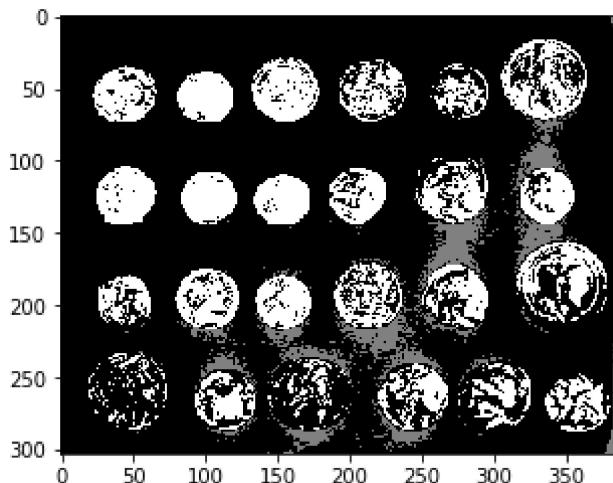
```
elevation_map = sobel(img_coins)
plt.imshow(elevation_map, cmap = plt.cm.gray);
```



Следующий шаг – найти маркеры фона и монет на основе крайних частей гистограммы серых значений.

In [45]:

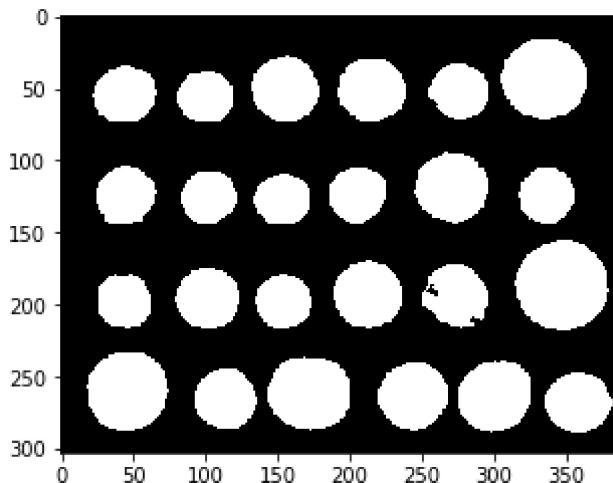
```
plt.imshow(markers, cmap = plt.cm.gray);
```



Теперь необходимо вычислить преобразование водораздела.

In [46]:

```
segmentation = watershed(elevation_map, markers)
plt.imshow(segmentation, cmap = plt.cm.gray);
```

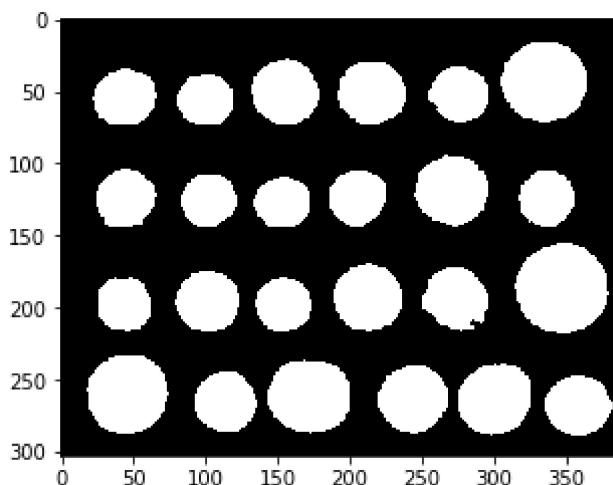


С помощью этого метода результат удовлетворяет всем монетам. Даже если маркеры на фоне не были хорошо распределены, барьеры на карте высот были достаточно высокими, чтобы эти маркеры заливали весь фон.

Необходимо удалить несколько небольших отверстий с помощью математической морфологии.

In [47]:

```
segmentation = ndi.binary_fill_holes(segmentation - 1)
plt.imshow(segmentation, cmap = plt.cm.gray);
```



Теперь нужно пометить все монеты используя ndi.label.

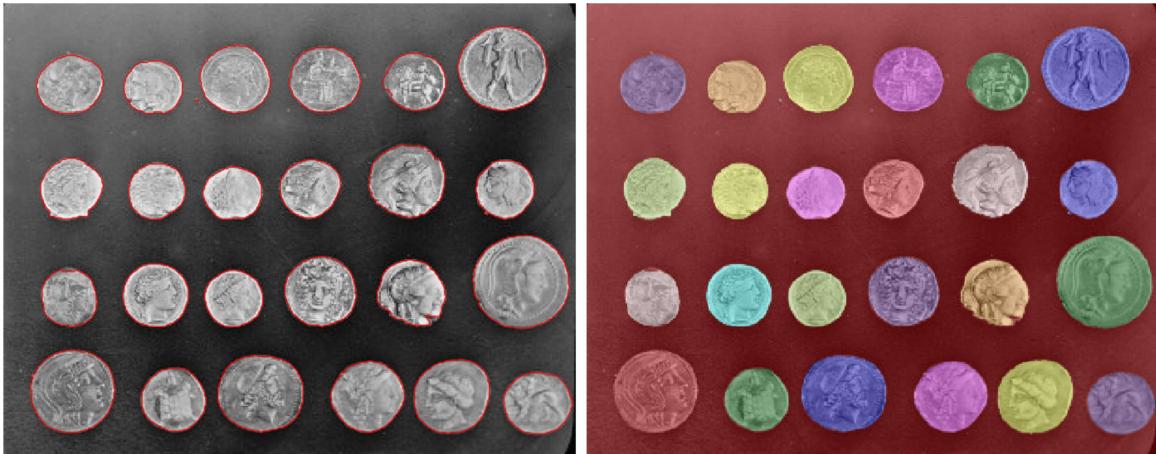
In [48]:

```
labeled_coins, _ = ndi.label(segmentation)
image_label_overlay = label2rgb(labeled_coins, image=img_coins)

fig, axes = plt.subplots(1, 2, figsize = (20, 10), sharey = True)
axes[0].imshow(img_coins, cmap=plt.cm.gray, interpolation = 'nearest')
axes[0].contour(segmentation, [0.5], linewidths = 1.2, colors = 'r')
axes[1].imshow(image_label_overlay, interpolation = 'nearest')

for a in axes:
    a.axis('off')
    a.set_adjustable('box-forced')

plt.tight_layout();
```



Основные операции с изображениями в OpenCV 3 Python:

<https://arboook.com/kompyuternoe-zrenie/osnovnye-operatsii-s-izobrazheniyami-v-opencv-3-python/>
(<https://arboook.com/kompyuternoe-zrenie/osnovnye-operatsii-s-izobrazheniyami-v-opencv-3-python/>).

Открытие локального файла в OpenCV 3 Python

Команда `imread()` возвращает NumPy массив, который содержит представление данных из изображения.

Метод для отображения `imshow()` принимает в себя два аргумента:

1. Название окна, в котором будет отрисовано изображение.
2. Имя переменной, которая хранит данное изображение.

Однако выполнение только данной команды отрисует изображение и сразу же закроет программу. Для того, чтобы изображение можно было увидеть и работать с ним, добавлена команда `waitKey(0)`.

Данная команда останавливает выполнение скрипта до нажатия клавиши на клавиатуре. Параметр 0 означает что нажатие любой клавиши будет засчитано.

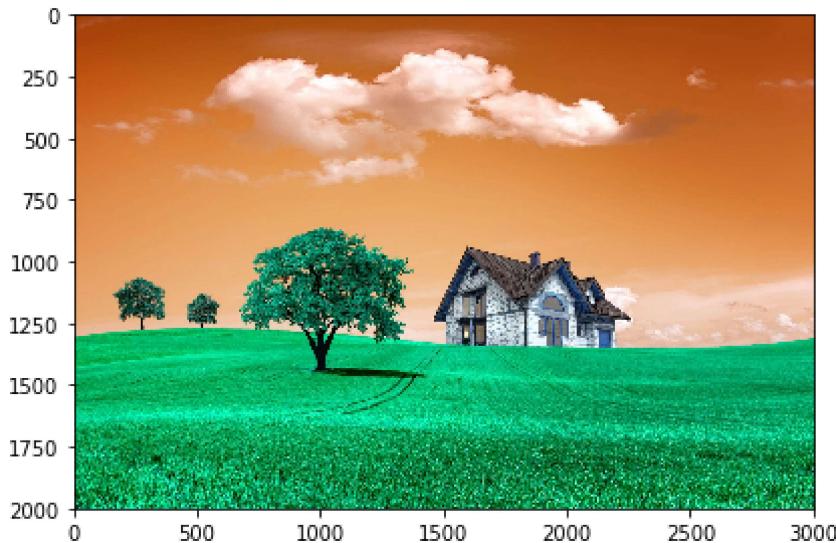
Команда `destroyAllWindows()` закрывает созданное окно.

In [49]:

```
image = cv2.imread("1.jpg")
imshow(image)
```

Out[49]:

```
<matplotlib.image.AxesImage at 0x226c676c240>
```



Уменьшаем изображение

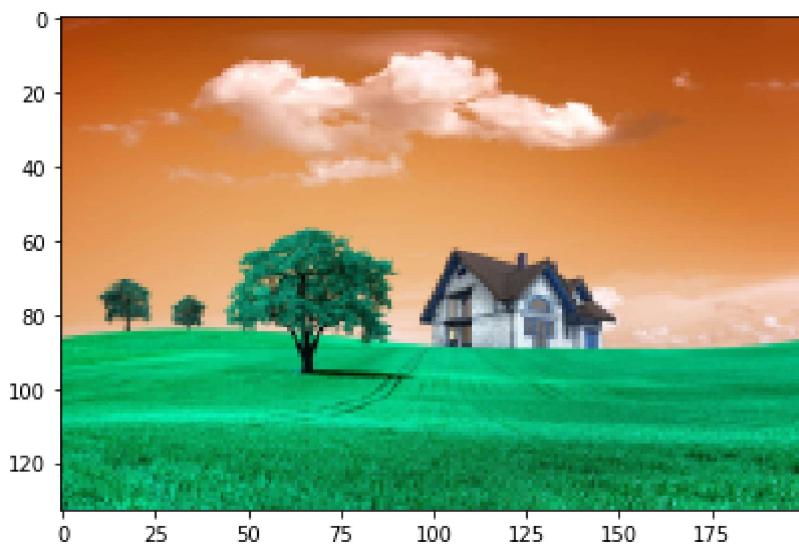
In [50]:

```
final_wide = 200
r = float(final_wide) / image.shape[1]
dim = (final_wide, int(image.shape[0] * r))

# уменьшаем изображение до подготовленных размеров
resized = cv2.resize(image, dim, interpolation = cv2.INTER_AREA)
imshow(resized)
```

Out[50]:

```
<matplotlib.image.AxesImage at 0x226c868aa20>
```



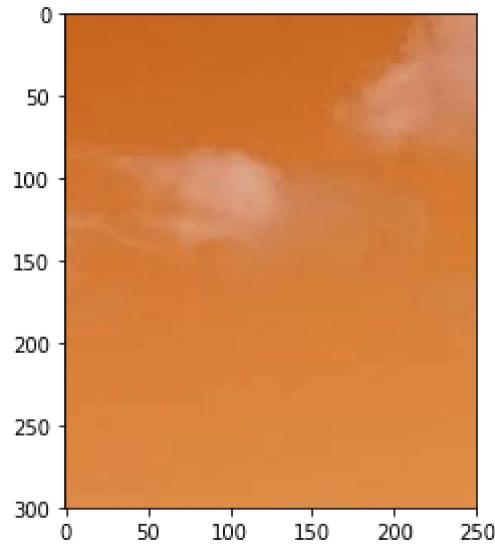
Вырезаем нужный фрагмент изображения

In [51]:

```
cropped = image[300:600, 350:600]  
imshow(cropped)
```

Out[51]:

```
<matplotlib.image.AxesImage at 0x226c8d511d0>
```



Поворот изображения

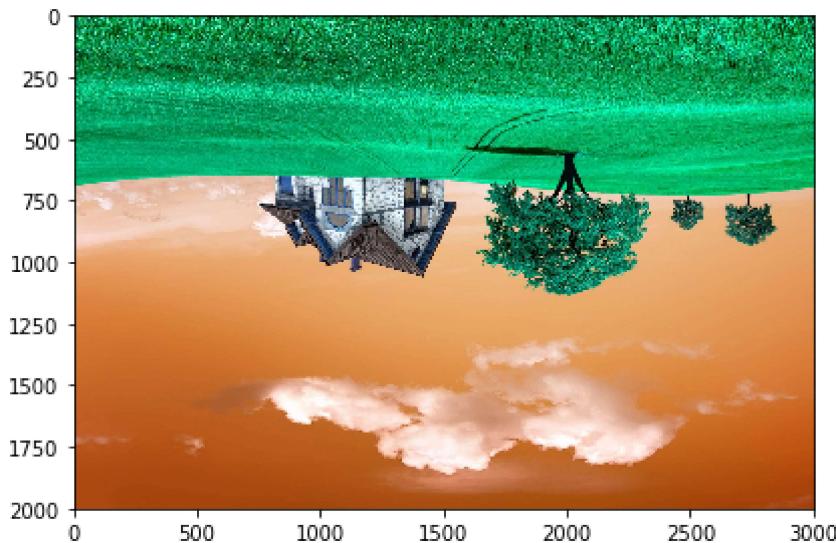
In [52]:

```
# получим размеры изображения для поворота
# и вычислим центр изображения
(h, w) = image.shape[:2]
center = (w / 2, h / 2)

# повернем изображение на 180 градусов
M = cv2.getRotationMatrix2D(center, 180, 1.0)
rotated = cv2.warpAffine(image, M, (w, h))
imshow(rotated)
```

Out[52]:

```
<matplotlib.image.AxesImage at 0x226c67e5400>
```



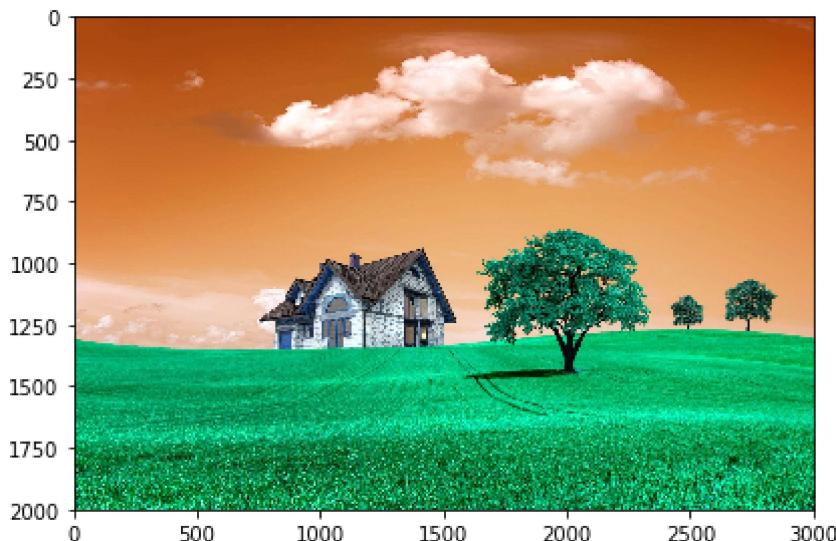
Отражение изображения по осям

In [53]:

```
flip_image = cv2.flip(image, 1)
imshow(flip_image)
```

Out[53]:

```
<matplotlib.image.AxesImage at 0x226c6759be0>
```



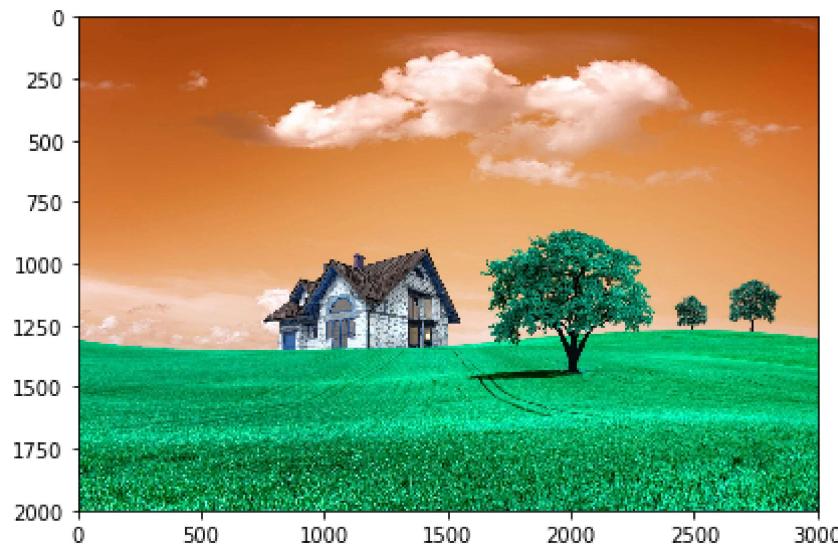
Сохранение изображения

In [54]:

```
cv2.imwrite("flip.png", flip_image)
flip_image = cv2.imread("flip.png")
imshow(flip_image)
```

Out[54]:

```
<matplotlib.image.AxesImage at 0x226c8723550>
```



Пишем скрипт для поиска книг на изображениях с помощью Python и OpenCV:

<https://tproger.ru/translations/finding-books-python-opencv> (<https://tproger.ru/translations/finding-books-python-opencv>)

Загрузка изображения с диска обрабатывается функцией `cv2.imread`. Затем цвета изображения преобразуются из RGB в оттенки серого. Также изображение размывается, чтобы уменьшить высокочастотные шумы и повысить точность сегментации.

In [55]:

```
image = cv2.imread('3.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (3, 3), 0)

plt.figure(figsize=(10,10))
plt.axis('off')
plt.imshow(gray, cmap = plt.cm.gray);
```

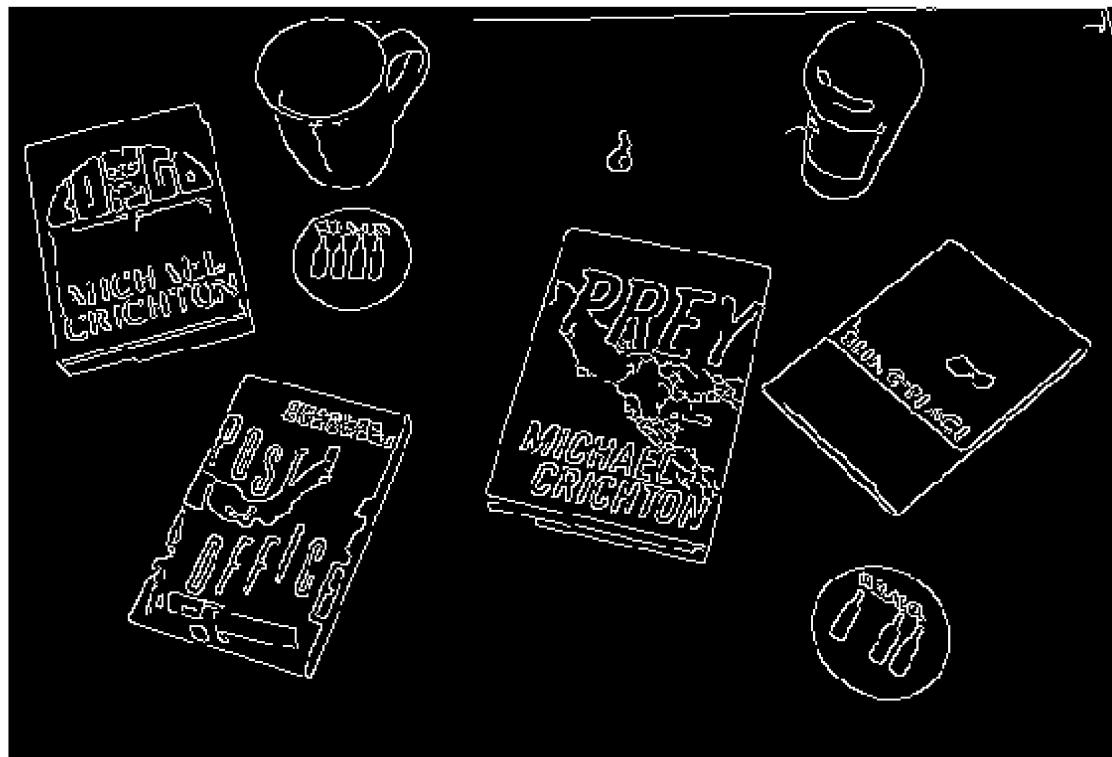


Теперь необходимо определить края (т.е. контуры) объектов на изображении.)

In [56]:

```
# распознавание контуров
edged = cv2.Canny(gray, 10, 250)

plt.figure(figsize=(10,10))
plt.axis('off')
plt.imshow(edged, cmap = plt.cm.gray);
```

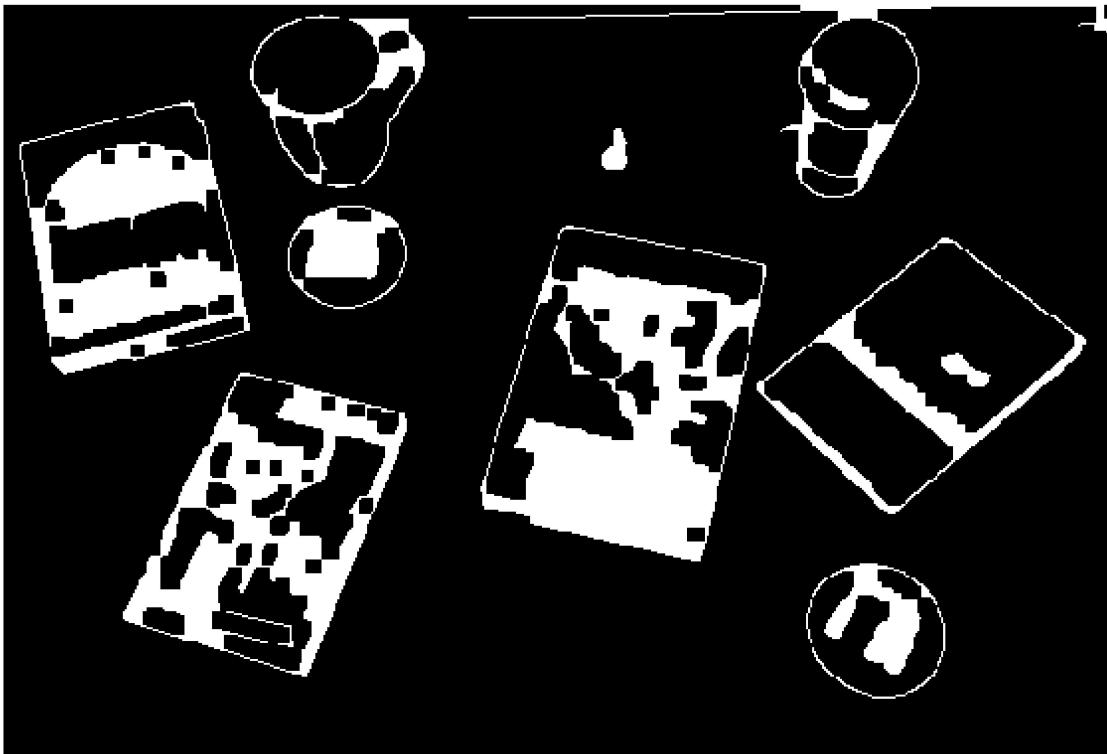


Некоторые из контуров не закрыты — между контурами существуют промежутки. Чтобы убрать промежутки между белыми пикселями изображения, необходимо применить операцию «закрытия».

In [57]:

```
# создайте и примените закрытие
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (7, 7))
closed = cv2.morphologyEx(edged, cv2.MORPH_CLOSE, kernel)

plt.figure(figsize=(10,10))
plt.axis('off')
plt.imshow(closed, cmap = plt.cm.gray);
```



Следующим шагом является фактическое обнаружение контуров объектов на изображении. Для этого используется функция cv2.findContours.

In [58]:

```
# найдите контуры в изображении и подсчитайте количество книг
(cnts, _) = cv2.findContours(closed.copy(), cv2.RETR_EXTERNAL,\n                             cv2.CHAIN_APPROX_SIMPLE)
total = 0
```

Книга представляет собой прямоугольник. У прямоугольника четыре вершины. Поэтому, если контур имеет четыре вершины, то можно предположить, что это книга.

Для каждого из контуров вычисляется периметр, с помощью cv2.arcLength, а затем контур аппроксимируется (он может быть не идеальным прямоугольником из-за шума и теней на фото), с помощью cv2.approxPolyDP.

Наконец, выполняется проверка, что у аппроксимируемого контура действительно четыре вершины. Если это так, то вокруг книги книги рисуется контур, а затем увеличиваем счётчик общего количества книг.

In [59]:

```
# цикл по контурам
for c in cnts:
    # аппроксимируем (сглаживаем) контур
    peri = cv2.arcLength(c, True)
    approx = cv2.approxPolyDP(c, 0.02 * peri, True)

    # если у контура 4 вершины, предполагаем, что это книга
    if len(approx) == 4:
        cv2.drawContours(image, [approx], -1, (0, 255, 0), 4)
        total += 1

print("Книг на картинке:",total)
plt.figure(figsize=(10,10))
plt.axis('off')
plt.imshow(image);
```

Книг на картинке: 4

