1 Dynamical Feature Extraction: Main Execution Block

1.1 Import and Initialization

```
# Get the number of series
if __name__ == '__main__':
    args = docopt(__doc__)
    image_paths = sorted(args['<filenames>'])
    min_thresh = int(args['--min_thresh'])
    max_thresh = int(args['--max_thresh'])
    side = args['--side']

    ensure_directory_exists('PROCESSED')
```

Explanation:

- This block starts by checking if the script is being run directly.
- It uses `docopt` to parse command-line arguments.
- `image_paths` stores the list of image file paths sorted.
- `min_thresh`, `max_thresh`, and `side` are command-line arguments converted to appropriate types.
- `ensure_directory_exists('PROCESSED')` ensures that the output directory exists.

1.2 Output File Naming

```
fileout_filename =
↪  'nh_fracdim_'+'_'.join(image_paths[0].split('/')[-1].replace('.jpg','').split('_')[:-1])+'.dat'
if side in ['left','right']:
    fileout_filename = side + '_' + fileout_filename

fileout_filename_frac =
↪  'nh_fracdim_'+'_'.join(image_paths[0].split('/')[-1].replace('.jpg','').split('_')[:-1])+'_frac.dat'
```

Explanation:

- This block constructs output filenames based on the first image path.
- If `side` is specified (left or right), it prefixes the filename with it.
- Two filenames are created: one for general output (`fileout_filename`) and another for fractional dimension data (`fileout_filename_frac`).

1.3 File Handling and Initialization

```
with open('PROCESSED' + '/' + fileout_filename, 'w') as fileout, open('PROCESSED' + '/' +
↪  fileout_filename_frac, 'w') as fileout_frac:
    output = (f'#{"frame":>8}  {"area":>15}  {"perimeter":>15}  {"circularity":>15}
    ↪  {"fracdim":>15}\r\n')
    output_frac = (f'#{"frame":>8}  {"BSR":>15}  {"NA":>15}  {"NP":>15}\r\n')
```

```
    last_center = None
    check_split = None
```

Explanation:

- The output files are opened for writing.
- The headers for the output files are defined.
- `last_center` and `check_split` are initialized to `None`.

1.4 Image Processing Loop

```
for image_index, image_path in enumerate(image_paths):
    # Load the image
    image = cv2.imread(image_path)
    physarum_only_image = cv2.imread(image_path)
    frame = int(image_path.split('.')[-2].split('_')[-1])

    # Convert to grayscale
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    height, width = gray_image.shape
    channels = 3

    if last_center is None:
        last_center = (int(width/2), int(height/2))
```

Explanation:

- Loop through each image in `image_paths`.
- Load the image using `cv2.imread`.
- Convert the image to grayscale.
- Get the image dimensions.
- Initialize `last_center` to the center of the image if it's `None`.

1.5 Noise Reduction and Mask Application

```
# Apply a median filter to reduce noise and increase connectivity
gray_image = cv2.medianBlur(gray_image, 9)

# Generate and apply mask to remove edges
mask = np.zeros_like(gray_image)
cv2.circle(mask, (int(width/2), int(height/2)), int(0.465*width), (255), -1)
gray_image = cv2.bitwise_and(gray_image, mask)
```

Explanation:

- Apply a median filter to reduce noise.

- Create a mask to remove the edges of the image.
- Apply the mask to the grayscale image.

1.6 Contour Detection and Sorting

```
if image_index == 0:
    first_sorted_contours, first_grouped_contours, first_sorted_indices, first_center, _=
    ↪  find_and_sort_contours(gray_image, min_thresh, max_thresh, last_center)
if side in ['right','left']:
    first_split_y = [i for i in range(100,int(height-100))]
    first_split = []
    for i, val in enumerate(first_split_y):
        if side == 'right':
            first_split.append([int(first_center[0]), val])
        else:
            first_split.append([int(first_center[0]), val])
```

Explanation:

- For the first image, find and sort contours based on the threshold values and center.
- If `side` is specified, generate a vertical split line through the center.

1.7 Side Mask Application

```
# Generate an additional mask to remove the left or right side of the image, optionally
if side == 'right':
    mask = np.zeros_like(gray_image)
    cv2.rectangle(mask, (int(first_center[0]), 0), (int(width), int(height)), (255), -1)
    gray_image = cv2.bitwise_and(gray_image, mask)
if side == 'left':
    mask = np.zeros_like(gray_image)
    cv2.rectangle(mask, (0, 0), (int(first_center[0]), int(height)), (255), -1)
    gray_image = cv2.bitwise_and(gray_image, mask)
```

Explanation:

- Apply a mask to remove either the left or right side of the image based on the `side` argument.
- For `side == 'right'`, the mask is applied to keep only the right side of the image.
- For `side == 'left'`, the mask is applied to keep only the left side of the image.

1.8 Contour Sorting and Drawing

```
# Find and sort contours again after applying the side mask
sorted_contours, grouped_contours, sorted_indices, last_center, weights =
↪  find_and_sort_contours(gray_image, min_thresh, max_thresh, last_center)
```

```
# Draw the sorted contours on the image
cv2.drawContours(image, sorted_contours, -1, (255, 0, 0), 3)
```

Explanation:

- Find and sort contours again after applying the side mask using the `find_and_sort_contours` function.
- Draw the sorted contours on the image using `cv2.drawContours` with a red color and a thickness of 3.

1.9 Quadrant Definition and Box Sizes

```
# Create quadrants based on image height
if side in ['left','right']:
    top_group, bottom_group = split_group_contours(physarum_only_image, grouped_contours,
    ↪  side, first_split)
    include_group = True
else:
    include_group = False

# Choose box sizes based on the dimensions of the image
if (height > 1000 and width > 1000):
    box_sizes = np.array([32, 48, 64, 80, 96, 112, 128, 160, 192, 224])
else:
    box_sizes = np.array([20, 30, 40, 50, 60, 70, 80, 100, 120, 140])
```

Explanation:

- Define quadrants for the image based on its height and the `side` argument, using the `split_group_contours` function.
- Choose box sizes based on the dimensions of the image to adjust the analysis scale.

1.10 Contour Processing and Metrics Calculation

```
# Create a mask for physarum only
physarum_mask = np.zeros_like(gray_image)
total_area = 0
total_perimeter = 0
points = []

for i, contour in enumerate(sorted_contours):
    M = cv2.moments(contour)
    cX = int(M["m10"] / M["m00"])
    cY = int(M["m01"] / M["m00"])
    if include_group:
        include = check_contour_in_group(cX, cY, top_group, bottom_group)
    else:
        include = True
```

```
    if include:
        cv2.drawContours(physarum_mask, [contour], -1, (255), thickness=-1)
        cv2.drawContours(physarum_only_image, [contour], -1, (255), thickness=-1)
        total_area += cv2.contourArea(contour)
        total_perimeter += cv2.arcLength(contour, True)

        epsilon = 0.01 * cv2.arcLength(contour, True)
        simplified_contours = cv2.approxPolyDP(contour, epsilon, True)
        simplified_points = np.array(simplified_contours).reshape(-1, 2)
        if len(points) == 0:
            points = simplified_points.tolist()
        else:
            points = points + simplified_points.tolist()

# Compute metrics
logbsr, logN = fractal_dimension_from_contour(physarum_mask, box_sizes)
slope, intercept, r_value, p_value, std_err = stats.linregress(logbsr, logN)
fracdim = slope

circularity = 0
if total_perimeter > 0:
    circularity = (4*math.pi*total_area)/(total_perimeter**2)
else:
    circularity = 0

if include_group:
    output += (f'{frame:8d}  {total_area:15.4f}  {total_perimeter:15.4f}
    ↪  {circularity:15.4f}  {fracdim:15.4f}\r\n')

output_frac += (f'{frame:8d}  {" ".join([str(elem) for elem in logbsr]):15}  {"
↪  ".join([str(elem) for elem in logN]):15}\r\n')
```

Explanation:

- Create a mask for the physarum image to isolate contours.
- Process each contour to compute its center, area, perimeter, and check if it should be included.
- Draw contours on the mask and the image.
- Calculate metrics such as area, perimeter, circularity, and fractal dimension, and format results for output.

1.11 Saving Output and Processed Image

```
cv2.imwrite(f'PROCESSED/{image_path.split("/")[-1]}', image)

# Write the output to file
fileout.write(output)
fileout_frac.write(output_frac)
```

Explanation:

- Save the processed image to the `PROCESSED` directory using the filename extracted from the image path.
- Write the computed metrics and fractal dimension data to the respective output files.

Explanation Summary

This script processes a series of images by applying filters and masks, detecting and sorting contours, calculating various metrics such as area, perimeter, circularity, and fractal dimension. It then saves the processed images and computed data to files. The command-line arguments allow for customization of thresholds and side masking options.