Dataset Exploration!

This notebook takes an initial look at the datasets downloaded. It identifies techniques to view the features of each image, as well as to generate videos to understand the movement and change in forms of the cells over time.

Author: Alexander Goudemond, Student Number: 219030365

This Jupyter Notebook aims at exploring the datasets for the project, and finding ways to represent the data in an easy way

Imports

In [1]:
```
from os import getcwd, walk, mkdir, stat, remove
from os import sep # used later on, in a function, to print directory contents
from os.path import exists, basename, join

from shutil import move, copy

import matplotlib.pyplot as plt
import cv2
```

dataset File Heirarchy

This section of the notebook focusses on understanding the file heirarchy of the datasets

Let us start with some imports:

In [2]:
```
print(getcwd())
```

```
c:\Users\G5\Documents\GitHub\COMP700
```

The file path used by author should be: "C:\Users\G5\UKZN\COMP700"

In [3]:
```
desired_directory = "..\\..\\Comp700_DataSets"
current_directory = getcwd()

print( exists(current_directory + "\\" + desired_directory) )
```

```
True
```

The output should be true as the file path we are interested in is:

```
Documents
|_______ ...

|_______Comp700_DataSets
        |_______ ...
```

```
    |_______ ...

    |_______Github
            |_______Comp700

    |_______ ...
```

There are many files inside this dataset folder, so let us get the first 2 'branches' to show the user

In [4]:
```python
path = walk(current_directory + "\\" + desired_directory)

count = 0
for root, dirs, files in path:
    print(dirs)
    print("Length is:", len(dirs), "\n")
    if (count == 1):
        break

    count += 1
```

```
['Extracted', 'OriginalZipped']
Length is: 2

['BF-C2DL-HSC', 'BF-C2DL-HSC (1)', 'BF-C2DL-MuSC', 'BF-C2DL-MuSC (1)', 'DIC-C2DH-HeLa',
'DIC-C2DH-HeLa (1)', 'Fluo-C2DL-Huh7', 'Fluo-C2DL-Huh7 (1)', 'Fluo-C2DL-MSC', 'Fluo-C2DL
-MSC (1)', 'Fluo-N2DH-GOWT1', 'Fluo-N2DH-GOWT1 (1)', 'Fluo-N2DH-SIM+', 'Fluo-N2DH-SIM+
(1)', 'Fluo-N2DL-HeLa', 'Fluo-N2DL-HeLa (1)', 'PhC-C2DH-U373', 'PhC-C2DH-U373 (1)', 'PhC
-C2DL-PSC', 'PhC-C2DL-PSC (1)']
Length is: 20
```

I.e. :

```
   Documents
    |_______ ...

    |_______Comp700_DataSets
            |_______Extracted
                    |_______BF-C2DL-HSC
                    |_______BF-C2DL-HSC (1)
                    |_______ ...

            |_______OriginalZipped
                    |_______BF-C2DL-HSC.zip
                    |_______BF-C2DL-HSC (1).zip
                    |_______ ...

    |_______ ...

    |_______Github

    |_______ ...
```

Zipped files are contained in the directory called *OriginalZipped*

A detailed breakdown of all directories can be shown with following function, adapted from stack overflow:

In [5]:
```python
# from os import sep

# Author: dhobbs on Mar 15, 2012 at 21:29,
# https:\\\\stackoverflow.com\\questions\\9727673\\list-directory-tree-structure-in-pyth
def list_files(startpath):
    for root, dirs, files in walk(startpath):
        level = root.replace(startpath, '').count(sep)
        indent = ' ' * 4 * (level)
        print('{}{}\\'.format(indent, basename(root)))
        subindent = ' ' * 4 * (level + 1)
        for f in files:
            # 3 lines of code added to improve readibility
            if (".tif" in f):
                print('{}{}'.format(subindent, "..."))
                break
            print('{}{}'.format(subindent, f))

path = (current_directory + "\\" + desired_directory)

list_files(path)
```

```
Comp700_DataSets\
    Extracted\
        BF-C2DL-HSC\
            BF-C2DL-HSC\
                01\
                    ...
                01_GT\
                    SEG\
                        ...
                    TRA\
                        man_track.txt
                        ...
                01_ST\
                    SEG\
                        ...
                02\
                    ...
                02_GT\
                    SEG\
                        ...
                    TRA\
                        man_track.txt
                        ...
                02_ST\
                    SEG\
                        ...
        BF-C2DL-HSC (1)\
            BF-C2DL-HSC (1)\
                01\
                    ...
                02\
                    ...
        BF-C2DL-MuSC\
            BF-C2DL-MuSC\
                01\
                    ...
                01_GT\
                    SEG\
                        ...
                    TRA\
                        man_track.txt
                        ...
                01_ST\
                    SEG\
                        ...
                02\
                    ...
                02_GT\
                    SEG\
                        ...
                    TRA\
                        man_track.txt
                        ...
                02_ST\
                    SEG\
                        ...
        BF-C2DL-MuSC (1)\
            BF-C2DL-MuSC (1)\
                01\
                    ...
                02\
                    ...
        DIC-C2DH-HeLa\
            DIC-C2DH-HeLa\
                01\
                    ...
                01_GT\
                    SEG\
                        ...
                    TRA\
                        man_track.txt
                        ...
                01_ST\
                    SEG\
                        ...
                02\
                    ...
                02_GT\
                    SEG\
                        ...
                    TRA\
                        man_track.txt
                        ...
                02_ST\
                    SEG\
                        ...
        DIC-C2DH-HeLa (1)\
            DIC-C2DH-HeLa (1)\
                01\
                    ...
                02\
                    ...
        Fluo-C2DL-Huh7\
            Fluo-C2DL-Huh7\
                01\
                    ...
                01_GT\
                    SEG\
                        ...
                    TRA\
                        man_track.txt
                        ...
                02\
                    ...
                02_GT\
                    SEG\
                        ...
                    TRA\
                        man_track.txt
                        ...
        Fluo-C2DL-Huh7 (1)\
            Fluo-C2DL-Huh7 (1)\
                01\
                    ...
                02\
                    ...
        Fluo-C2DL-MSC\
            Fluo-C2DL-MSC\
                01\
                    ...
                01_GT\
                    SEG\
                        ...
                    TRA\
                        man_track.txt
                        ...
                01_ST\
                    SEG\
                        ...
                02\
                    ...
                02_GT\
                    SEG\
                        ...
                    TRA\
                        man_track.txt
                        ...
                02_ST\
                    SEG\
                        ...
        Fluo-C2DL-MSC (1)\
            Fluo-C2DL-MSC (1)\
                01\
                    ...
                02\
                    ...
        Fluo-N2DH-GOWT1\
            Fluo-N2DH-GOWT1\
                01\
                    ...
                01_GT\
                    SEG\
                        ...
                    TRA\
                        man_track.txt
                        ...
                01_ST\
                    SEG\
                        ...
                02\
                    ...
                02_GT\
                    SEG\
                        ...
                    TRA\
                        man_track.txt
                        ...
                02_ST\
                    SEG\
                        ...
        Fluo-N2DH-GOWT1 (1)\
            Fluo-N2DH-GOWT1 (1)\
                01\
                    ...
                02\
                    ...
        Fluo-N2DH-SIM+\
            Fluo-N2DH-SIM+\
                01\
                    ...
```

```
          01_GT\
              SEG\
                  ...
              TRA\
                  man_track.txt
                  ...
          02\
              ...
          02_GT\
              SEG\
                  ...
              TRA\
                  man_track.txt
                  ...
  Fluo-N2DH-SIM+ (1)\
      Fluo-N2DH-SIM+ (1)\
          01\
              ...
          02\
              ...
  Fluo-N2DL-HeLa\
      Fluo-N2DL-HeLa\
          01\
              ...
          01_GT\
              SEG\
                  ...
              TRA\
                  man_track.txt
                  ...
          01_ST\
              SEG\
                  ...
          02\
              ...
          02_GT\
              SEG\
                  ...
              TRA\
                  man_track.txt
                  ...
          02_ST\
              SEG\
                  ...
  Fluo-N2DL-HeLa (1)\
      Fluo-N2DL-HeLa (1)\
          01\
              ...
          02\
              ...
  PhC-C2DH-U373\
      PhC-C2DH-U373\
          01\
              ...
          01_GT\
              SEG\
                  ...
              TRA\
                  man_track.txt
                  ...
          01_ST\
              SEG\
                  ...
          02\
              ...
          02_GT\
                SEG\
                    ...
                TRA\
                    man_track.txt
                    ...
            02_ST\
                SEG\
                    ...
    PhC-C2DH-U373 (1)\
        PhC-C2DH-U373 (1)\
            01\
                ...
            02\
                ...
    PhC-C2DL-PSC\
        PhC-C2DL-PSC\
            01\
                ...
            01_GT\
                SEG\
                    ...
                TRA\
                    man_track.txt
                    ...
            01_ST\
                SEG\
                    ...
            02\
                ...
            02_GT\
                SEG\
                    ...
                TRA\
                    man_track.txt
                    ...
            02_ST\
                SEG\
                    ...
    PhC-C2DL-PSC (1)\
        PhC-C2DL-PSC (1)\
            01\
                ...
            02\
                ...
OriginalZipped\
    BF-C2DL-HSC (1).zip
    BF-C2DL-HSC.zip
    BF-C2DL-MuSC(1).zip
    BF-C2DL-MuSC.zip
    DIC-C2DH-HeLa(1).zip
    DIC-C2DH-HeLa.zip
    Fluo-C2DL-Huh7(1).zip
    Fluo-C2DL-Huh7.zip
    Fluo-C2DL-MSC(1).zip
    Fluo-C2DL-MSC.zip
    Fluo-N2DH-GOWT1(1).zip
    Fluo-N2DH-GOWT1.zip
    Fluo-N2DH-SIM+(1).zip
    Fluo-N2DH-SIM+.zip
    Fluo-N2DL-HeLa(1).zip
    Fluo-N2DL-HeLa.zip
    PhC-C2DH-U373(1).zip
    PhC-C2DH-U373.zip
    PhC-C2DL-PSC(1).zip
    PhC-C2DL-PSC.zip
```

The ellipses above indicate that images are contained in that location.

In order to view every folder's collection of images, we need to travel individually into each folder and look at the images contained there.

Before we go any further, it is important to acknowledge the structure of the directories.

There are 20 datasets contained here. Every folder that has the suffix *(1)* represents the CHALLENGE dataset, provided by the Cell Tracking Challenge.

Inside the TRAINING datasets, additional folders appear, either as shown on the left OR as shown on the right:

```
01                                          01
01_GT                                       01_GT
01_ST                                       02
02                                          02_GT
02_GT
02_ST
```

If the folder contains the suffix "_GT" of "_ST", then the image contents have the naming convention "man_segXXXX.tif" or "man_trackXXXX.tif", otherwise the images have the naming convention "tXXXX.tif", where XXXX represents the number of the image in that folder

The author suspects that "man_seg" represents a manually segmented cell, whereas "man_track" represents a manually tracked cell. The other files are the "Petri Dish" files, showing the complete landscape of the cells under the microscope

Now, windows photo viewer (and the extension in VS Code) show the "man_" images as pure black images.... This is not helpful or useful. We need to identify how to see the details here!

Image recognition

In the section below, the notebook will explore ways to extract the valuable information from these seemingly black images

To verify, let us bring 3 images to the local directory and try open them using OpenCV

In [6]:
```
sample_directory_1 = "002_DataSetExploration"
sample_directory_2 = "SampleImages"
sample_directory = sample_directory_1 + "\\" + sample_directory_2

try:
    mkdir( join(current_directory, sample_directory_1) )
    mkdir( join( join(current_directory, sample_directory_1) , sample_directory_2) )
except FileExistsError:
    pass
except:
    print("Unknown Error Encountered...")
```

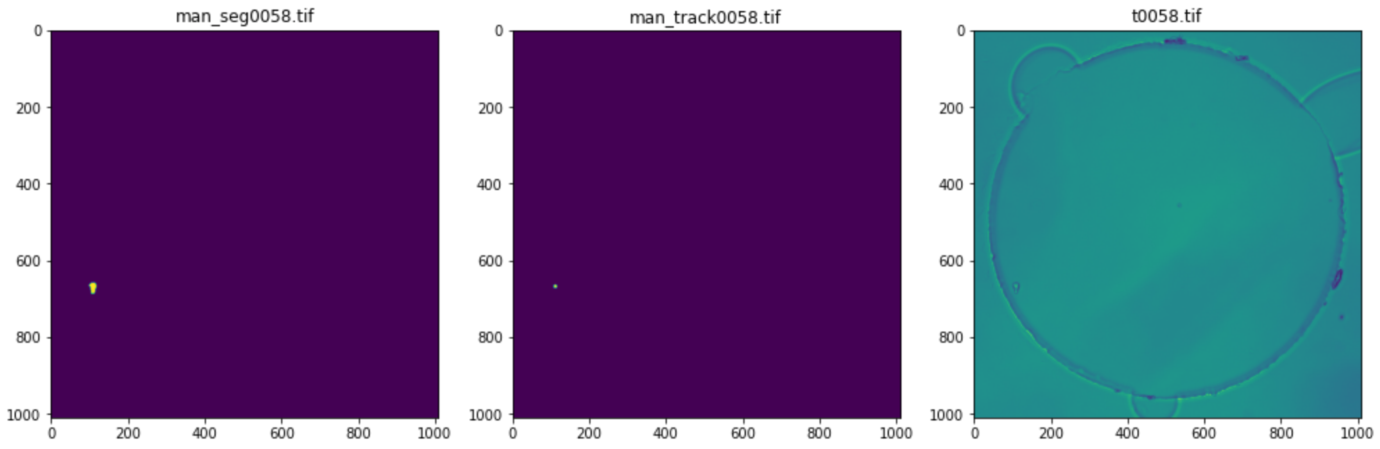
In [7]:
```
# get 3 images and save to local folder

path = walk(desired_directory)
```

```
# hard coded to generate a local copy
locations = [
            "Extracted\\BF-C2DL-HSC\\BF-C2DL-HSC\\01",
            "Extracted\\BF-C2DL-HSC\\BF-C2DL-HSC\\01_GT\\SEG",
            "Extracted\\BF-C2DL-HSC\\BF-C2DL-HSC\\01_GT\\TRA"
            ]

flags = ["t0058.tif", "man_seg0058.tif", "man_track0058.tif"]
flag_count = 0

for root, dirs, files in path:
    # print(files)
    # terminate
    if (flag_count > 2):
        break

    for file in files:
        # terminate
        if (flag_count > 2):
            break
        elif (flags[flag_count] == file):
            # print("Yerp")
            copy( join( join(desired_directory, locations[flag_count]) , file) , sample_
            flag_count += 1
            break
```

However, If we use Matplotlib to view a sample of each of the TIF images, we see the following:

In [8]:
```
path = walk(current_directory + "\\" + sample_directory)

i = 1
for root, dirs, files in path:
    print(files)
    fig = plt.figure(figsize=(14, 8))

    for item in files:
        location = (current_directory + "\\" + sample_directory + "\\" + item)
        img = plt.imread(location)

        fig.add_subplot(1, 3, i)
        plt.title(item)
        plt.imshow(img)
        i += 1

plt.tight_layout()
plt.show()
```

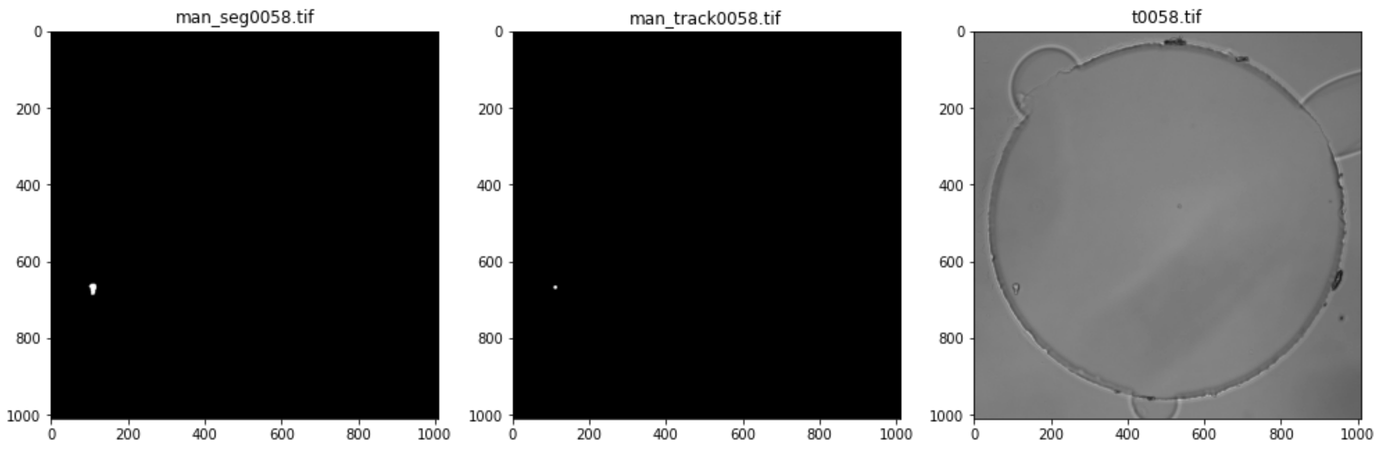
```
['man_seg0058.tif', 'man_track0058.tif', 't0058.tif']
```



Here is the same, as grayscale:

In [9]:
```python
path = walk(current_directory + "\\" + sample_directory)

i = 1
for root, dirs, files in path:
    print(files)
    fig = plt.figure(figsize=(14, 8))

    for item in files:
        location = (current_directory + "\\" + sample_directory + "\\" + item)
        img = plt.imread(location)

        fig.add_subplot(1, 3, i)
        plt.title(item)
        plt.imshow(img, cmap="gray")
        i += 1

plt.tight_layout()
plt.show()
```

```
['man_seg0058.tif', 'man_track0058.tif', 't0058.tif']
```



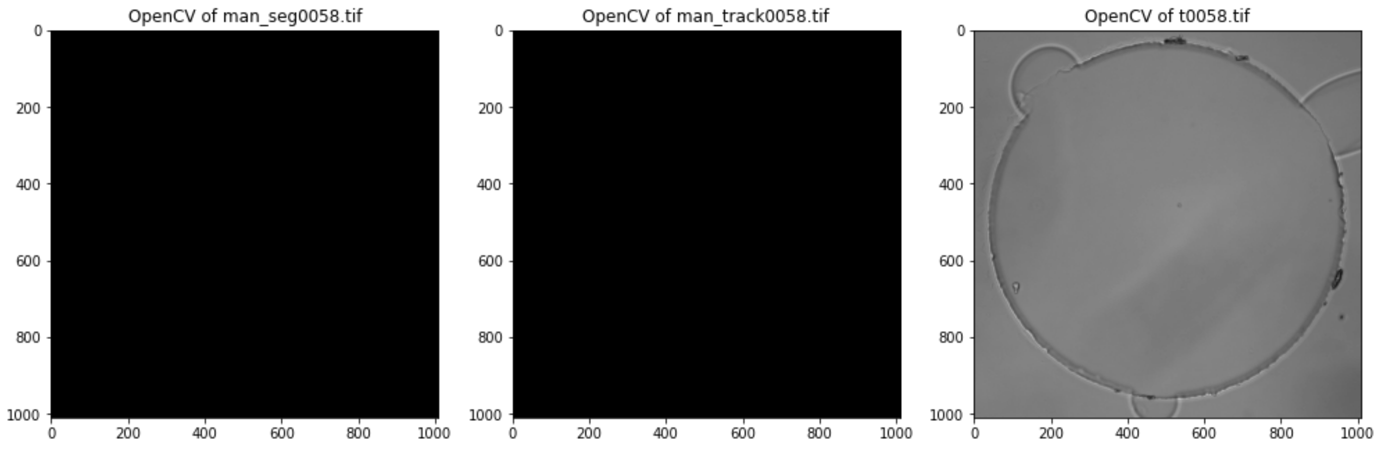
As we can see from the pop-up window - the details in the Opencv window are not being shown... They ARE being shown in the Matplotlib.Pyplot package.

This is not desirable, as Opencv has some useful tools we'd like to use for data visualization. So we need to find a way to get the features shown from Matplotlib, and preserve them for future use with Opencv

Let us see if we can mix and match:

In [10]:
```python
path = walk(current_directory + "\\" + sample_directory)

i = 1
for root, dirs, files in path:
    print(files)
    fig = plt.figure(figsize=(14, 8))

    for item in files:
        location = (current_directory + "\\" + sample_directory + "\\" + item)
        img = cv2.imread(location, cv2.IMREAD_GRAYSCALE)

        fig.add_subplot(1, 3, i)
        plt.title("OpenCV of " + item)
        plt.imshow(img, cmap="gray")
        i += 1

plt.tight_layout()
plt.show()
```

```
['man_seg0058.tif', 'man_track0058.tif', 't0058.tif']
```



NO! We cannot mix and match. What about saving a file using Matplotlib and then opening it using Opencv?

```
In [11]: destination_directory_1 = "002_DataSetExploration"
         destination_directory_2 = "MatplotlibSavedImages"
         destination_directory = destination_directory_1 + "\\" + destination_directory_2

         try:
             mkdir( join( join(current_directory, destination_directory_1) , destination_director
         except FileExistsError:
             pass
         except:
             print("Unknown Error Encountered...")
```

```
In [12]: path = walk(current_directory + "\\" + sample_directory)

         # i = 1
         for root, dirs, files in path:
             print(files)
             # fig = plt.figure(figsize=(14, 8))

             for item in files:
                 location = (current_directory + "\\" + sample_directory + "\\" + item)
                 img = plt.imread(location)
                 name = destination_directory + "\\" + "Matplotlib_" + item[ : -4] + ".jpg"
                 plt.imsave(name, img, cmap="gray")

         path = walk(current_directory + "\\" + destination_directory)

         for root, dirs, files in path:
             print(files)
             for item in files:
                 location = (current_directory + "\\" + destination_directory + "\\" + item)
                 img = cv2.imread(location, cv2.IMREAD_GRAYSCALE)
                 (x, y) = img.shape
                 break

         imgSmall = cv2.resize(img, (x // 2, y // 2))
         cv2.imshow(item, imgSmall)
         cv2.waitKey(0)
```

```
         ['man_seg0058.tif', 'man_track0058.tif', 't0058.tif']
         ['Matplotlib_man_seg0058.jpg', 'Matplotlib_man_track0058.jpg', 'Matplotlib_t0058.jpg']
Out[12]: -1
```

YES! We can do that. So that is one option for how we can address the inconsitent black images. However, this option changes the quality from TIFF to JPG...

Another option is to attempt to read the images in using matplotlib.pyplot and then write them using Opencv

Let us try to do that now:

In [13]:
```
destination_directory_1 = "002_DataSetExploration"
destination_directory_2 = "OpencvSavedImages"
destination_directory = destination_directory_1 + "\\" + destination_directory_2

try:
    mkdir( join( join(current_directory, destination_directory_1) , destination_director
except FileExistsError:
    pass
except:
    print("Unknown Error Encountered...")
```

In [14]:
```
path = walk(current_directory + "\\" + sample_directory)

# i = 1
for root, dirs, files in path:
    print(files)
    # fig = plt.figure(figsize=(14, 8))

    for item in files:
        location = (current_directory + "\\" + sample_directory + "\\" + item)
        img = plt.imread(location)
        name = destination_directory + "\\" + "Opencv_" + item
        cv2.imwrite(name, img)

path = walk(current_directory + "\\" + destination_directory)

for root, dirs, files in path:
    print(files)
    for item in files:
        location = (current_directory + "\\" + destination_directory + "\\" + item)
        img = cv2.imread(location, cv2.IMREAD_GRAYSCALE)
        (x, y) = img.shape
        break

imgSmall = cv2.resize(img, (x // 2, y // 2))
cv2.imshow(item, imgSmall)
cv2.waitKey(0)
```

```
['man_seg0058.tif', 'man_track0058.tif', 't0058.tif']
['Opencv_man_seg0058.tif', 'Opencv_man_track0058.tif', 'Opencv_t0058.tif']
```

Out[14]:
```
-1
```

Unfortunately, that does not work... So our best bet is to read them in and save them using Matplotlib. This is not ideal, as we may lose a bit of quality converting from TIF to JPG

We desire to read in the image using Matplotlib, and write it to disk as a grayscale TIF. Though this does not appear possible, as Matplotlib does not support TIFF extensions. Let's try create Grayscale TIFF's from the sample pictures?

In [15]:
```
destination_directory_1 = "002_DataSetExploration"
destination_directory_2 = "GrayscaleTiffs"
destination_directory = destination_directory_1 + "\\" + destination_directory_2

try:
    mkdir( join( join(current_directory, destination_directory_1) , destination_director
except FileExistsError:
    pass
except:
    print("Unknown Error Encountered...")
```

In [16]:
```
path = walk(current_directory + "\\" + sample_directory)

# i = 1
for root, dirs, files in path:
    print(files)
    # fig = plt.figure(figsize=(14, 8))

    for item in files:
        location = (current_directory + "\\" + sample_directory + "\\" + item)
        img = cv2.imread(location, cv2.IMREAD_GRAYSCALE)
        name = destination_directory + "\\" + "Grayscale_" + item
        cv2.imwrite(name, img)
```

```
['man_seg0058.tif', 'man_track0058.tif', 't0058.tif']
```

Unfortunately, no difference! Looks like our best bet is to convert it to JPG when we need to... Let's move on

We can use a function to calculate the size of a file like this:

In [17]:
```
# from os import stat

bytes = stat("002_DataSetExploration\\GrayscaleTiffs\\Grayscale_man_seg0058.tif").st_siz
print("File is", bytes / 1024, "kb's")
```

```
File is 19.185546875 kb's
```

If we compare the Sample Images file size to that of our other files, we recognize the following:

In [18]:
```
places_array = ["002_DataSetExploration\\SampleImages", "002_DataSetExploration\\Graysca
                "002_DataSetExploration\\MatplotlibSavedImages", "002_DataSetExploration

spacing_length = len("002_DataSetExploration\\MatplotlibSavedImages") # used to improve

for i in range(len(places_array)):
    location = current_directory + "\\" + places_array[i]
    path = walk(location)

    for root, dirs, files in path:
        print(places_array[i], " " * ( spacing_length - len(places_array[i]) ), end="\t"

        for item in files:
            bytes = stat(location + "\\" + item).st_size
            bytes = bytes / 1024
            print( str(bytes)[ : 6], "kb's", end="\t") # keep first 5 sig figs (ignore f
        print()
```

```
002_DataSetExploration\SampleImages             38.419 kb's     38.281 kb's     509.12 k
b's
002_DataSetExploration\GrayscaleTiffs           19.185 kb's     19.185 kb's     414.22 k
b's
002_DataSetExploration\MatplotlibSavedImages    16.892 kb's     16.738 kb's     42.908 k
b's
002_DataSetExploration\OpencvSavedImages        38.277 kb's     38.201 kb's     414.22 k
b's
```

From the results above, we can see that the Grayscale TIFFs we created result in a loss of information for the manually segmented and tracked images, but only a small loss for the main image.

The file sizes for MatplotlibSavedImages follows a similar pattern, however the original Petri Dish image has a large loss in quality...

OpencvSavedImages should be virtually identical to the original pictures, as Opencv merely read them in and saved them, as is.

From the above, we can infer that a loss in quality in the manually segmented and tracked images is acceptable, however we would like to try preserve the data in the Petri dish image.

Before we conclude this, let us look at the Pillow package to identify if we can see the details in the manually segmented and tracked images

In [19]:
```
# Imports PIL module
from PIL import Image

im = Image.open("002_DataSetExploration\\SampleImages\\man_seg0058.tif")
im.show()
```

Unfortunately not... We have no choice but to convert each image into a grayscale image through Matplotlib.Pyplot, in order to see the details for the manually segmented and tracked images. One option we have is to leave the Petri Dish images as Tiff, and convert the rest to JPG... that solution may be ideal as we preserve the most amount of information for our segmentation software.

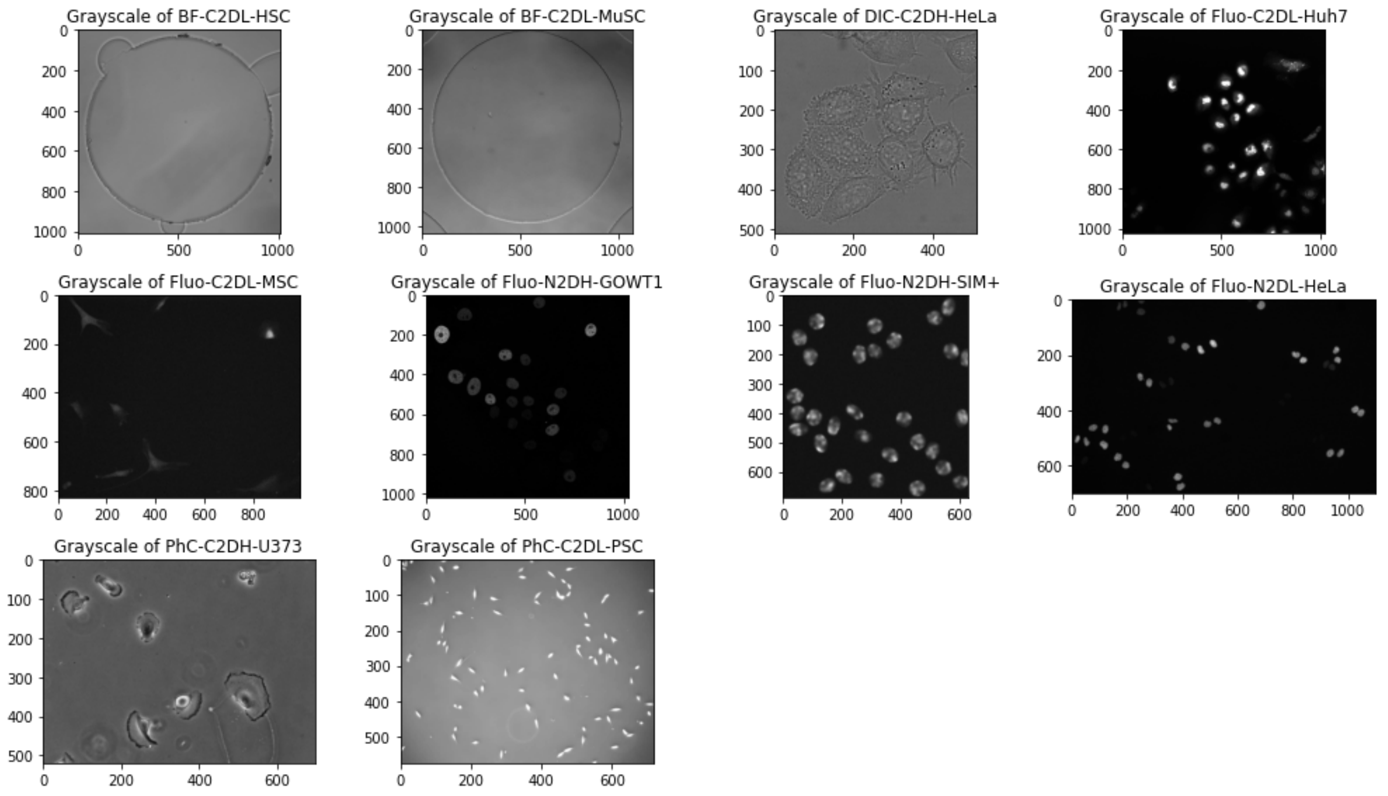
dataset Cell Variation

This section of the notebook focusses on sampling one Perti Dish image from each folder and placing them together, to see the different cells present

In [20]:
```
'''
We only need to show every _OTHER_ folder, as each dataset has a
 training and challenge set. So out of 20 files, we need to show 10

First things first, let us create an array of the directory locations
'''

data_sets = "..\\..\\Comp700_DataSets"

path = walk(current_directory + "\\" + data_sets)

directory_array = [] # contains the main folders

i = 1
for root, dirs, files in path:
    if (i == 2):
        directory_array = dirs
        break

    i += 1

print(directory_array)

print("\nStarting matplotlib\n")

path = walk(current_directory + "\\" + data_sets) # reset path

fig = plt.figure(figsize=(14, 8))
```

```
i = -1
temp = -1
counter = 1 # used for matplotlib subplots
for root, dirs, files in path:
    # print(dirs)
    for item in files:
        # only execute for first picture in directory
        if ("t0000.tif" == item) or ("t000.tif" == item):
            i += 1

            # skips folder "02" in datasets
            if (i % 2 == 1):
                break

            # print(i)
            temp = i // 2

            # skip Challenge datasets
            if ("(1)" in directory_array[temp]):
                break

            location = ( current_directory + "\\" + data_sets + "\\Extracted\\" + direct
                        "\\" + directory_array[temp] + "\\01\\" + item)
            # print(location)

            img = plt.imread(location)

            fig.add_subplot(3, 4, counter)
            plt.title("Grayscale of " + directory_array[temp])
            plt.imshow(img, cmap="gray")

            counter += 1

            break

        else:
            break


plt.tight_layout()

# save file for future use
plt.savefig("002_DataSetExploration\\Visualization_Of_Cells.jpg")

plt.show()

# save file for future use
# plt.savefig("Visualization_Of_Cells.jpg")
```

```
['BF-C2DL-HSC', 'BF-C2DL-HSC (1)', 'BF-C2DL-MuSC', 'BF-C2DL-MuSC (1)', 'DIC-C2DH-HeLa',
'DIC-C2DH-HeLa (1)', 'Fluo-C2DL-Huh7', 'Fluo-C2DL-Huh7 (1)', 'Fluo-C2DL-MSC', 'Fluo-C2DL
-MSC (1)', 'Fluo-N2DH-GOWT1', 'Fluo-N2DH-GOWT1 (1)', 'Fluo-N2DH-SIM+', 'Fluo-N2DH-SIM+
(1)', 'Fluo-N2DL-HeLa', 'Fluo-N2DL-HeLa (1)', 'PhC-C2DH-U373', 'PhC-C2DH-U373 (1)', 'PhC
-C2DL-PSC', 'PhC-C2DL-PSC (1)']

Starting matplotlib
```


Grayscale of BF-C2DL-HSC
Grayscale of BF-C2DL-MuSC
Grayscale of DIC-C2DH-HeLa
Grayscale of Fluo-C2DL-Huh7
Grayscale of Fluo-C2DL-MSC
Grayscale of Fluo-N2DH-GOWT1
Grayscale of Fluo-N2DH-SIM+
Grayscale of Fluo-N2DL-HeLa
Grayscale of PhC-C2DH-U373
Grayscale of PhC-C2DL-PSC

Great! That is a nice visualization of each dataset. If we look at the website, we can confirm the dimensions of each microscope used:

dataset Name	Pixel Size (Microns)	Time Step (Min)
BF-C2DL-HSC	0.645 X 0.645	5
BF-C2DL-MuSC	0.645 X 0.645	5
DIC-C2DH-HeLa	0.19 x 0.19	10
Fluo-C2DL-Huh7	0.65 x 0.65	15
Fluo-C2DL-MSC	0.3 x 0.3	20
Fluo-N2DH-GOWT1	0.240 x 0.240	5
Fluo-N2DH-SIM+	0.125 x 0.125	29
Fluo-N2DL-HeLa	0.645 x 0.645	30
PhC-C2DH-U373	0.65 x 0.65	15
PhC-C2DL-PSC	1.6 x 1.6	10

dataset Videos Variables

This section of the notebook will generate videos for each of the datasets

We desire to have a short video showing the movement of the cells over time, which we can refer to when we want to understand the features of the data.

We will need to use OpenCV to generate the videos, depending on the length of data present. For example, folders containing 1000 images may run for more time than those with 300 images.

Each video (for Training and Challenge) will be saved in a folder, outside of Github, for ease of reference. This is because the combined file size can be ~200MB, so it is same in the same directory as the dataset images

First, let us create an a few arrays for the information we need:

In [21]:
```
# a list of the directories - will be used to name the videos
print(directory_array)
```

```
['BF-C2DL-HSC', 'BF-C2DL-HSC (1)', 'BF-C2DL-MuSC', 'BF-C2DL-MuSC (1)', 'DIC-C2DH-HeLa',
'DIC-C2DH-HeLa (1)', 'Fluo-C2DL-Huh7', 'Fluo-C2DL-Huh7 (1)', 'Fluo-C2DL-MSC', 'Fluo-C2DL
-MSC (1)', 'Fluo-N2DH-GOWT1', 'Fluo-N2DH-GOWT1 (1)', 'Fluo-N2DH-SIM+', 'Fluo-N2DH-SIM+
(1)', 'Fluo-N2DL-HeLa', 'Fluo-N2DL-HeLa (1)', 'PhC-C2DH-U373', 'PhC-C2DH-U373 (1)', 'PhC
-C2DL-PSC', 'PhC-C2DL-PSC (1)']
```

In [22]:
```
# First, generate a list of the locations for each folder of Petri Dish images

path = walk(current_directory + "\\" + data_sets) # reset path

location_array = []

# used to cycle between 2 folders, present in each directory
sub_directory_choice = ["\\01\\", "\\02\\"]

i = 0 # will grow from 0 to 39
index = 0 # we need to lie in [0, 19]
for root, dirs, files in path:
    # print(dirs)
    for item in files:
        if ("t0000.tif" == item) or ("t000.tif" == item):
            index = i // 2

            location = ( current_directory + "\\" + data_sets + "\\Extracted\\" + direct
                        "\\" + directory_array[index] + sub_directory_choice[i % 2])

            i += 1
            # print(location)
            location_array.append(location)

print(location_array)
```

```
['c:\\Users\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Extracted\\BF-C2D
L-HSC\\BF-C2DL-HSC\\01\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_D
ataSets\\Extracted\\BF-C2DL-HSC\\BF-C2DL-HSC\\02\\', 'c:\\Users\\G5\\Documents\\GitHub
\\COMP700\\..\\..\\Comp700_DataSets\\Extracted\\BF-C2DL-HSC (1)\\BF-C2DL-HSC (1)\\01\\',
'c:\\Users\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Extracted\\BF-C2DL
-HSC (1)\\BF-C2DL-HSC (1)\\02\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Co
mp700_DataSets\\Extracted\\BF-C2DL-MuSC\\BF-C2DL-MuSC\\01\\', 'c:\\Users\\G5\\Documents
\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Extracted\\BF-C2DL-MuSC\\BF-C2DL-MuSC\\02
\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Extracted\\BF
-C2DL-MuSC (1)\\BF-C2DL-MuSC (1)\\01\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700
\\..\\..\\Comp700_DataSets\\Extracted\\BF-C2DL-MuSC (1)\\BF-C2DL-MuSC (1)\\02\\', 'c:\\U
sers\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Extracted\\DIC-C2DH-HeLa
\\DIC-C2DH-HeLa\\01\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_Data
Sets\\Extracted\\DIC-C2DH-HeLa\\DIC-C2DH-HeLa\\02\\', 'c:\\Users\\G5\\Documents\\GitHub
\\COMP700\\..\\..\\Comp700_DataSets\\Extracted\\DIC-C2DH-HeLa (1)\\DIC-C2DH-HeLa (1)\\01
\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Extracted\\DI
C-C2DH-HeLa (1)\\DIC-C2DH-HeLa (1)\\02\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700
\\..\\..\\Comp700_DataSets\\Extracted\\Fluo-C2DL-Huh7\\Fluo-C2DL-Huh7\\01\\', 'c:\\Users
\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Extracted\\Fluo-C2DL-Huh7\\F
luo-C2DL-Huh7\\02\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSe
ts\\Extracted\\Fluo-C2DL-Huh7 (1)\\Fluo-C2DL-Huh7 (1)\\01\\', 'c:\\Users\\G5\\Documents
\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Extracted\\Fluo-C2DL-Huh7 (1)\\Fluo-C2DL-Hu
h7 (1)\\02\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Ext
racted\\Fluo-C2DL-MSC\\Fluo-C2DL-MSC\\01\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700
\\..\\..\\Comp700_DataSets\\Extracted\\Fluo-C2DL-MSC\\Fluo-C2DL-MSC\\02\\', 'c:\\Users
\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Extracted\\Fluo-C2DL-MSC (1)
\\Fluo-C2DL-MSC (1)\\01\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_
DataSets\\Extracted\\Fluo-C2DL-MSC (1)\\Fluo-C2DL-MSC (1)\\02\\', 'c:\\Users\\G5\\Docume
nts\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Extracted\\Fluo-N2DH-GOWT1\\Fluo-N2DH-GO
WT1\\01\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Extrac
ted\\Fluo-N2DH-GOWT1\\Fluo-N2DH-GOWT1\\02\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700
\\..\\..\\Comp700_DataSets\\Extracted\\Fluo-N2DH-GOWT1 (1)\\Fluo-N2DH-GOWT1 (1)\\01\\',
'c:\\Users\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Extracted\\Fluo-N2
DH-GOWT1 (1)\\Fluo-N2DH-GOWT1 (1)\\02\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700
\\..\\..\\Comp700_DataSets\\Extracted\\Fluo-N2DH-SIM+\\Fluo-N2DH-SIM+\\01\\', 'c:\\Users
\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Extracted\\Fluo-N2DH-SIM+\\F
luo-N2DH-SIM+\\02\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSe
ts\\Extracted\\Fluo-N2DH-SIM+ (1)\\Fluo-N2DH-SIM+ (1)\\01\\', 'c:\\Users\\G5\\Documents
\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Extracted\\Fluo-N2DH-SIM+ (1)\\Fluo-N2DH-SI
M+ (1)\\02\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Ext
racted\\Fluo-N2DL-HeLa\\Fluo-N2DL-HeLa\\01\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP70
0\\..\\..\\Comp700_DataSets\\Extracted\\Fluo-N2DL-HeLa\\Fluo-N2DL-HeLa\\02\\', 'c:\\User
s\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Extracted\\Fluo-N2DL-HeLa
(1)\\Fluo-N2DL-HeLa (1)\\01\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp
700_DataSets\\Extracted\\Fluo-N2DL-HeLa (1)\\Fluo-N2DL-HeLa (1)\\02\\', 'c:\\Users\\G5
\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Extracted\\PhC-C2DH-U373\\PhC-C2
DH-U373\\01\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Ex
tracted\\PhC-C2DH-U373\\PhC-C2DH-U373\\02\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700
\\..\\..\\Comp700_DataSets\\Extracted\\PhC-C2DH-U373 (1)\\PhC-C2DH-U373 (1)\\01\\',
'c:\\Users\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Extracted\\PhC-C2D
H-U373 (1)\\PhC-C2DH-U373 (1)\\02\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700
\\..\\..\\Comp700_DataSets\\Extracted\\PhC-C2DL-PSC\\PhC-C2DL-PSC\\01\\', 'c:\\Users\\G5
\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Extracted\\PhC-C2DL-PSC\\PhC-C2D
L-PSC\\02\\', 'c:\\Users\\G5\\Documents\\GitHub\\COMP700\\..\\..\\Comp700_DataSets\\Extr
acted\\PhC-C2DL-PSC (1)\\PhC-C2DL-PSC (1)\\01\\', 'c:\\Users\\G5\\Documents\\GitHub\\COM
P700\\..\\..\\Comp700_DataSets\\Extracted\\PhC-C2DL-PSC (1)\\PhC-C2DL-PSC (1)\\02\\']
```

The cell below verifies that all cells have exactly 1 set of dimensions across the folder. It may take a few minutes to run

In [23]:
```
# # Now, use the location_array to determine if each dataset has consistent image sizes

# path = walk(current_directory + "\\" + data_sets) # reset path

# temp_array = []

# i = -1 # will grow from 0 to 39
# for root, dirs, files in path:
#     # print(files)
#     for item in files:
#         if ("man_" not in item) and (".zip" not in item):
#             # update on first element only
#             if ("t0000.tif" == item) or ("t000.tif" == item):
#                 i += 1

#             # print(location_array[i] + item, exists(location_array[i] + item))
#             # break

#             img = cv2.imread( (location_array[i] + item), cv2.IMREAD_GRAYSCALE)
#             (x, y) = img.shape

#             # only keep distinct sizes
#             if ([x, y] not in temp_array):
#                 temp_array.append([x, y])


#         # skip "man_" images
#         else:
```

```
#               break

#     if (len(temp_array) != 0):
#         print(temp_array, end="\t")

#     temp_array = []
```

From the test above, we can see that the images are already consistent in their dimensions, however the dimensions vary per dataset. We can use this knowledge to just find the dimensions of the first image in each folder, which will be used further on

In [24]:
```
image_size_array = []

path = walk(current_directory + "\\" + data_sets) # reset path

i = -1 # will grow from 0 to 39
for root, dirs, files in path:
    for item in files:
        if ("man_" not in item) and (".zip" not in item):
            # update on first element only
            if ("t0000.tif" == item) or ("t000.tif" == item):
                i += 1

            img = cv2.imread( (location_array[i] + item), cv2.IMREAD_GRAYSCALE)
            (x, y) = img.shape

            image_size_array.append([x, y])
            break


        # skip "man_" images
        else:
            break

print(image_size_array)
```

```
[[1010, 1010], [1010, 1010], [1010, 1010], [1010, 1010], [1036, 1070], [1036, 1070], [10
36, 1070], [1036, 1070], [512, 512], [512, 512], [512, 512], [512, 512], [1024, 1024],
[1024, 1024], [1024, 1024], [1024, 1024], [832, 992], [782, 1200], [832, 992], [782, 120
0], [1024, 1024], [1024, 1024], [1024, 1024], [1024, 1024], [690, 628], [773, 739], [71
8, 660], [790, 664], [700, 1100], [700, 1100], [700, 1100], [700, 1100], [520, 696], [52
0, 696], [520, 696], [520, 696], [576, 720], [576, 720], [576, 720], [576, 720]]
```

OKAY! We now have useful arrays we can use to generate our videos: location_array is used to find the position of the images, and image_size_array is used to find the dimensions of the folder. Next, we need to determine the quantity of pictures in each folder, as that will become our frame rate:

In [25]:
```
quantity_images_per_folder = []

path = walk(current_directory + "\\" + data_sets) # reset path

count = 0
for root, dirs, files in path:
    for item in files:
        if ("man_" not in item) and (".zip" not in item):
            count += 1

        # skip "man_" images
        else:
            break
    # zero from previous cycle, thus item is empty
    if (count != 0):
        quantity_images_per_folder.append(count)
    count = 0 # reset

print(quantity_images_per_folder)
```

```
[1764, 1764, 1763, 1763, 1375, 1376, 1376, 1375, 84, 84, 115, 115, 30, 30, 30, 30, 48, 4
8, 48, 48, 92, 92, 92, 92, 65, 150, 110, 138, 92, 92, 92, 92, 115, 115, 115, 115, 300, 3
00, 300, 300]
```

dataset Colour options

This section of the notebook explores if changing the colour scheme of the image reveals the details in the simulated images

Before going any further, recall that OpenCV is unable to show the details on a black screen when reading the image in as Colour:

In [26]:
```
sample_directory_1 = "002_DataSetExploration"
sample_directory_2 = "SimulatedSamples"
sample_directory = sample_directory_1 + "\\" + sample_directory_2

try:
    mkdir( join( join(current_directory, sample_directory_1) , sample_directory_2) )
except FileExistsError:
    pass
except:
    print("Unknown Error Encountered...")
```

In [27]:
```
# copy 1 image to desired folder
location = "Extracted\\Fluo-N2DH-SIM+\\Fluo-N2DH-SIM+\\01"
file = "t000.tif"

copy( join( join( join(current_directory, data_sets) , location) , file) , sample_direct
```

Out[27]:
```
'002_DataSetExploration\\SimulatedSamples\\t000.tif'
```

In [29]:
```
# BGR
path = walk(current_directory + "\\002_DataSetExploration\\SimulatedSamples")

img = cv2.imread( current_directory + "\\002_DataSetExploration\\SimulatedSamples\\t000.
cv2.imshow("BGR", img)
cv2.waitKey(0)
```

Out[29]:
```
-1
```

We recognize that OpenCV uses the BGR colour order, so perhaps inverting the colours is better?

In [30]:
```
# RGB
path = walk(current_directory + "\\002_DataSetExploration\\SimulatedSamples")

img = cv2.imread( current_directory + "\\002_DataSetExploration\\SimulatedSamples\\t000.
img = img[ ... , ::-1 ] # invert array order, because based on Numpy array
cv2.imshow("RGB", img)
cv2.waitKey(0)
```

Out[30]:
```
-1
```

Still nothing... What about different combinations of the colour channels?

In [31]:
```python
# RBG
path = walk(current_directory + "\\002_DataSetExploration\\SimulatedSamples")

bgr_img = cv2.imread( current_directory + "\\002_DataSetExploration\\SimulatedSamples\\t
b, g, r = cv2.split(bgr_img)
rbg_img = cv2.merge( [r, b, g] )
cv2.imshow("RBG", rbg_img)
cv2.waitKey(0)
```

Out[31]:
```
-1
```

In [32]:
```python
# GBR
path = walk(current_directory + "\\002_DataSetExploration\\SimulatedSamples")

bgr_img = cv2.imread( current_directory + "\\002_DataSetExploration\\SimulatedSamples\\t
b, g, r = cv2.split(bgr_img)
gbr_img = cv2.merge( [g, b, r] )
cv2.imshow("GBR", gbr_img)
cv2.waitKey(0)
```

Out[32]:
```
-1
```

In [33]:
```python
# GRB
path = walk(current_directory + "\\002_DataSetExploration\\SimulatedSamples")

bgr_img = cv2.imread( current_directory + "\\002_DataSetExploration\\SimulatedSamples\\t
b, g, r = cv2.split(bgr_img)
grb_img = cv2.merge( [g, r, b] )
cv2.imshow("GRB", grb_img)
cv2.waitKey(0)
```

Out[33]:
```
-1
```

In [34]:
```python
# BRG
path = walk(current_directory + "\\002_DataSetExploration\\SimulatedSamples")

bgr_img = cv2.imread( current_directory + "\\002_DataSetExploration\\SimulatedSamples\\t
b, g, r = cv2.split(bgr_img)
brg_img = cv2.merge( [b, r, g] )
cv2.imshow("BRG", brg_img)
cv2.waitKey(0)
```

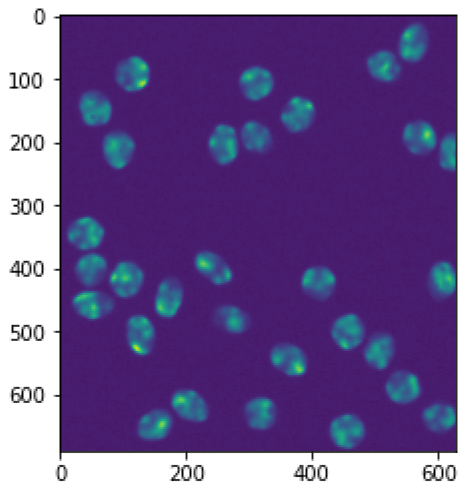
Out[34]:
```
-1
```

None of those combinations produce the hidden data... so OpenCV is not accessing the information present. NOW, Matploylib.pyplot is a different story:

If we read in the data and plot it via plt we see the following:

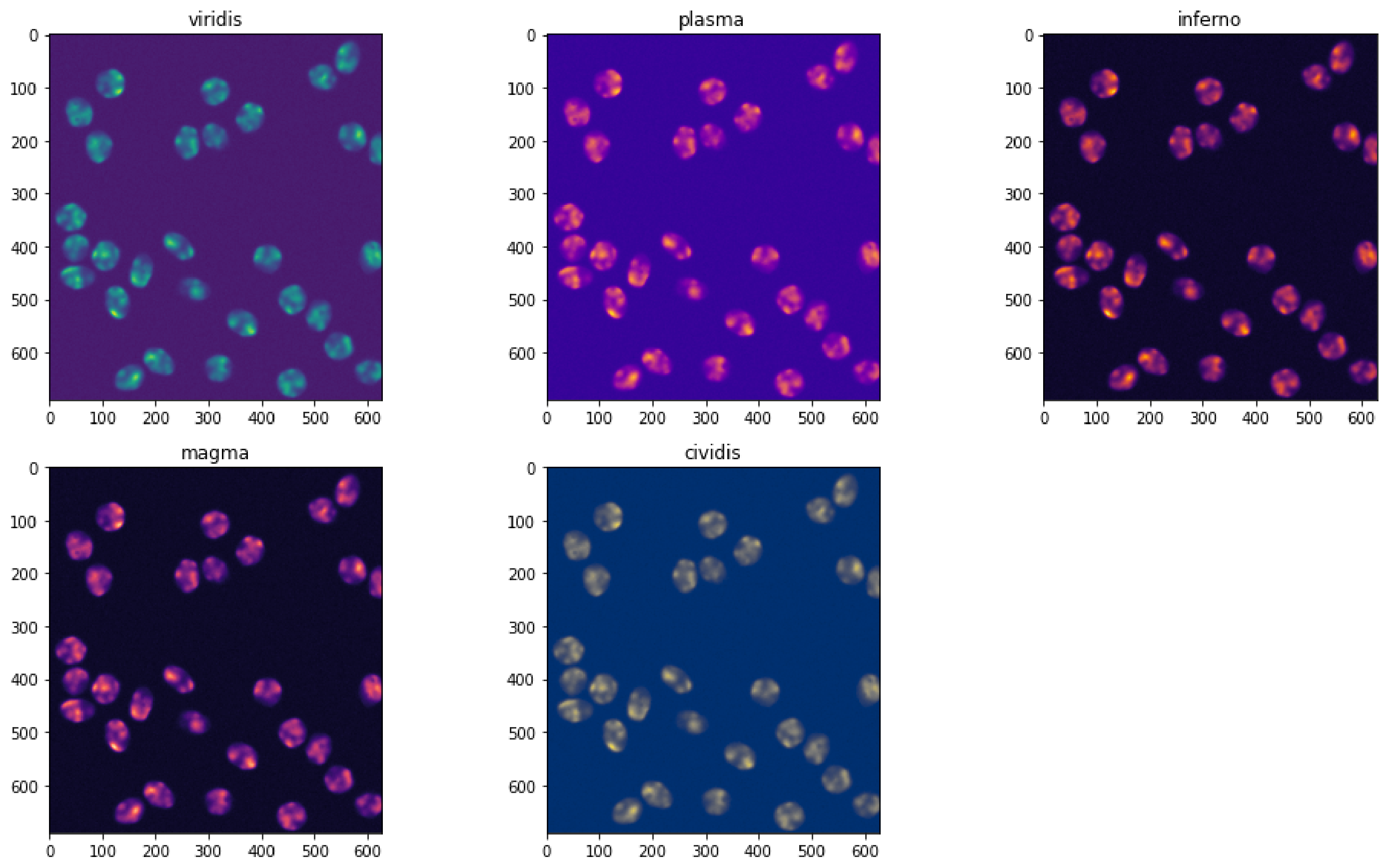
In [35]:
```python
path = walk(current_directory + "\\002_DataSetExploration\\SimulatedSamples")

img = plt.imread( current_directory + "\\002_DataSetExploration\\SimulatedSamples\\t000.
plt.imshow(img)
```

Out[35]:
```
<matplotlib.image.AxesImage at 0x2d2b1106440>
```



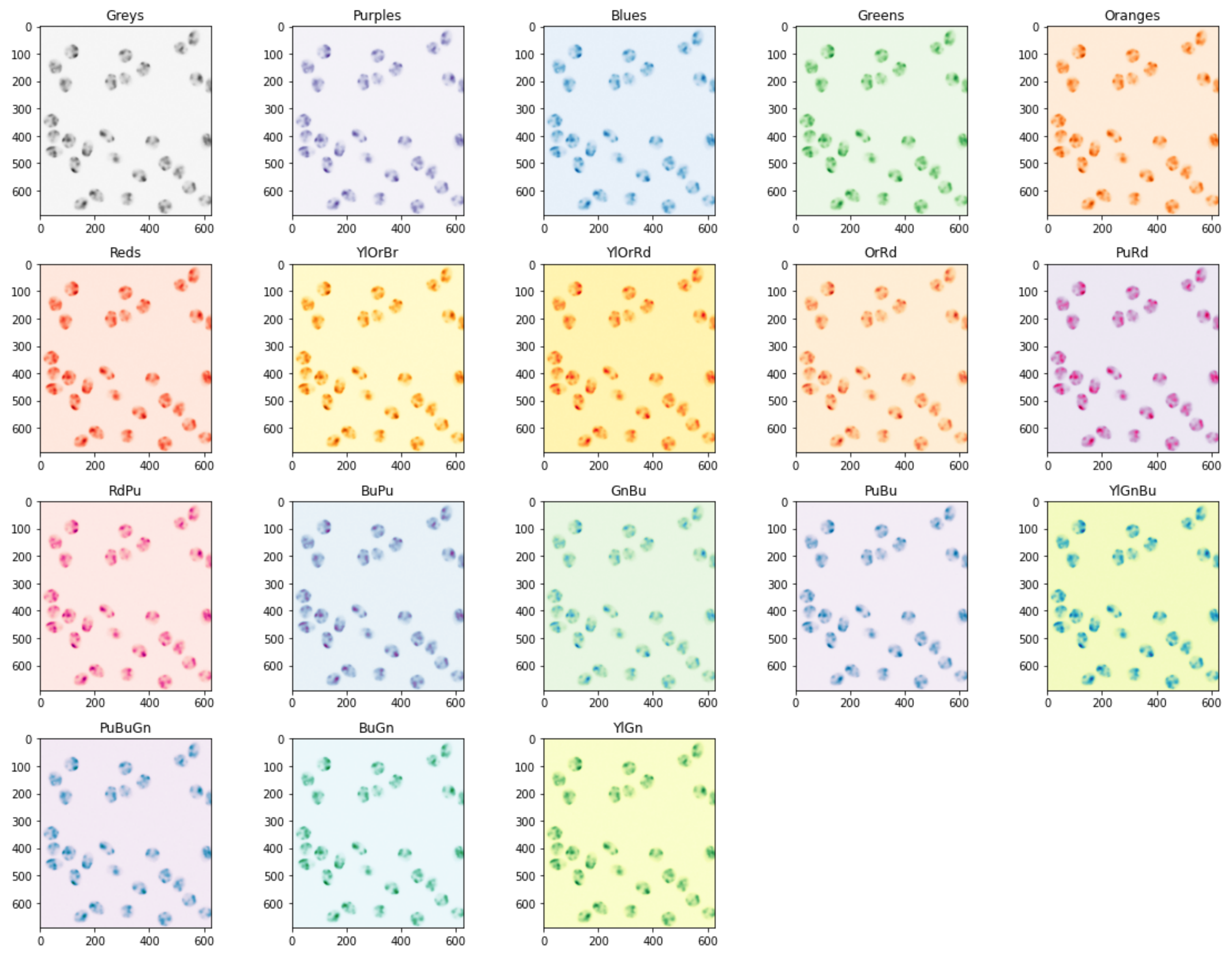
So the colour scheme is off... The default colour scheme for Pyplot is a Sequential colour 'Viridis'. Let us explore the other choices:

In [36]:
```
# Perceptually Uniform Sequential Colourmaps
path = walk(current_directory + "\\002_DataSetExploration\\SimulatedSamples")

rgb_img = plt.imread( current_directory + "\\002_DataSetExploration\\SimulatedSamples\\t
colour_choices = ["viridis", "plasma", "inferno", "magma", "cividis"]

fig = plt.figure(figsize=(14, 8))

for i in range(5):
    fig.add_subplot(2, 3, i+1)
    plt.title(colour_choices[i])
    plt.imshow(rgb_img, cmap=colour_choices[i])

plt.tight_layout()
plt.show()
```

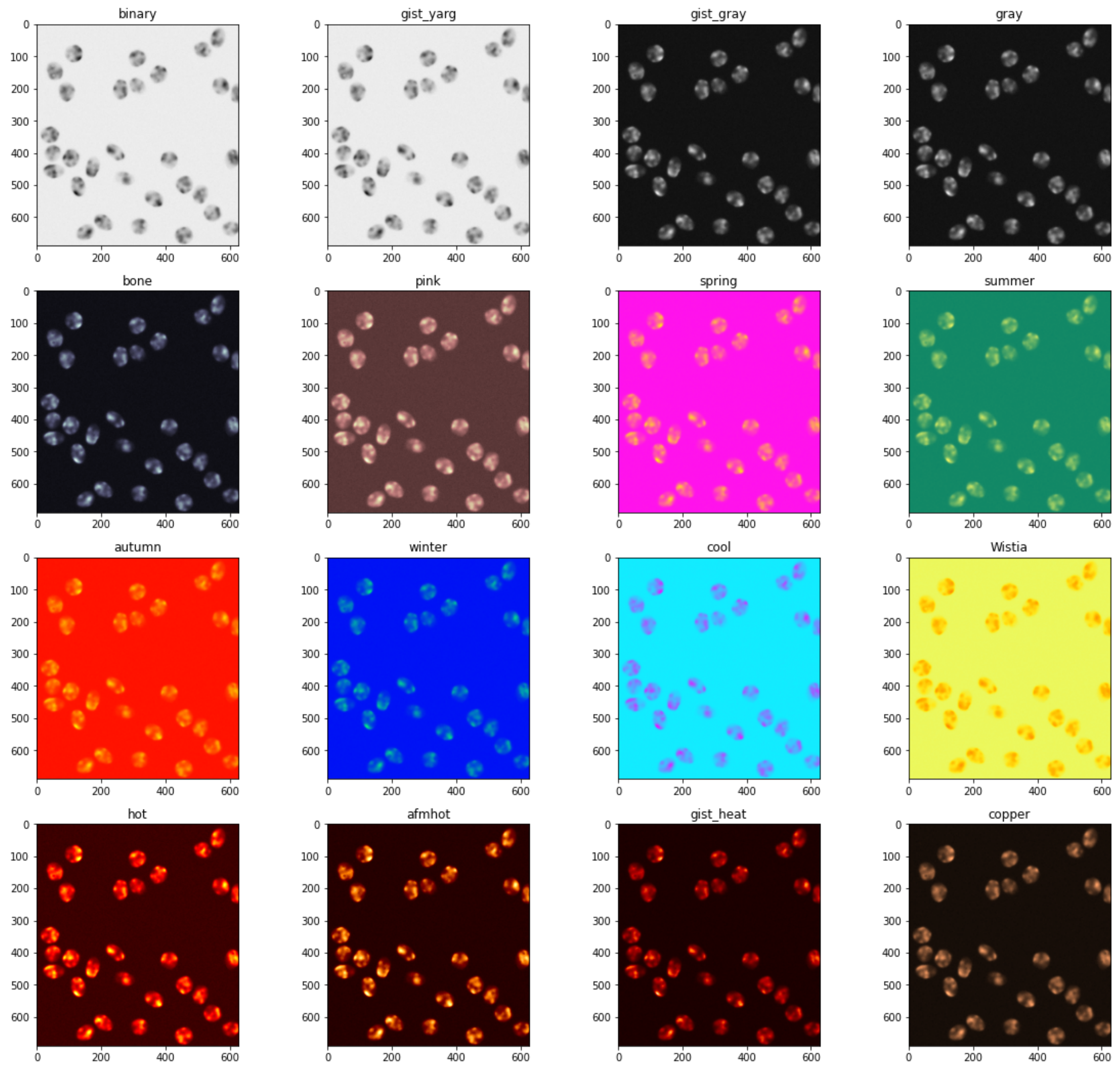


```
# Sequential
```

In [37]:
```
path = walk(current_directory + "\\002_DataSetExploration\\SimulatedSamples")

rgb_img = plt.imread( current_directory + "\\002_DataSetExploration\\SimulatedSamples\\t
colour_choices = ['Greys', 'Purples', 'Blues', 'Greens', 'Oranges', 'Reds',
                  'YlOrBr', 'YlOrRd', 'OrRd', 'PuRd', 'RdPu', 'BuPu',
                  'GnBu', 'PuBu', 'YlGnBu', 'PuBuGn', 'BuGn', 'YlGn']

fig = plt.figure(figsize=(16, 12))

for i in range(len(colour_choices)):
    fig.add_subplot(4, 5, i+1)
    plt.title(colour_choices[i])
    plt.imshow(rgb_img, cmap=colour_choices[i])

plt.tight_layout()
plt.show()
```

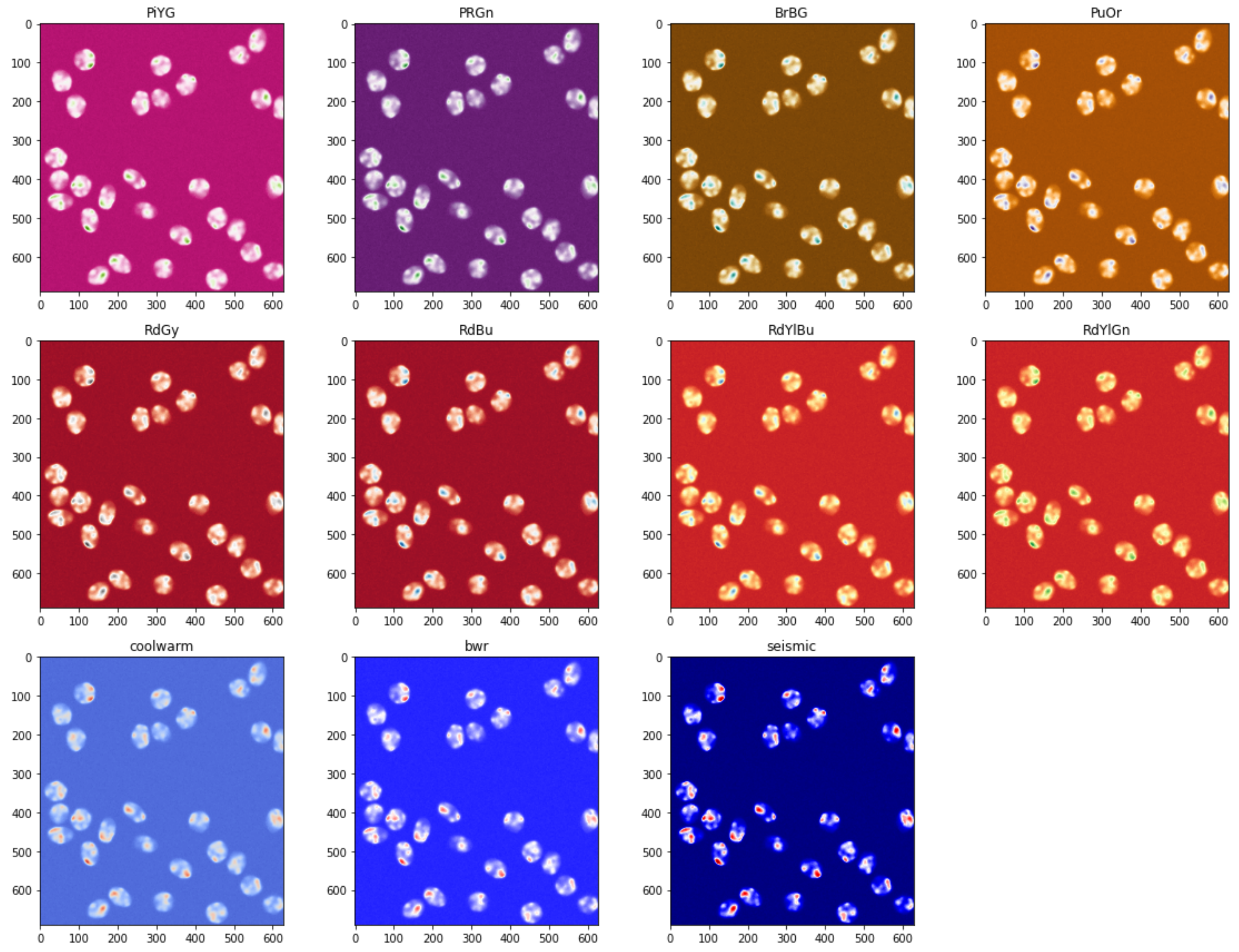


In [38]:
```
# Sequential2
path = walk(current_directory + "\\002_DataSetExploration\\SimulatedSamples")

rgb_img = plt.imread( current_directory + "\\002_DataSetExploration\\SimulatedSamples\\t
colour_choices = ['binary', 'gist_yarg', 'gist_gray', 'gray', 'bone',
                  'pink', 'spring', 'summer', 'autumn', 'winter', 'cool',
                  'Wistia', 'hot', 'afmhot', 'gist_heat', 'copper']

fig = plt.figure(figsize=(16, 18))

for i in range(len(colour_choices)):
    fig.add_subplot(5, 4, i+1)
    plt.title(colour_choices[i])
    plt.imshow(rgb_img, cmap=colour_choices[i])
plt.tight_layout()
plt.show()
```

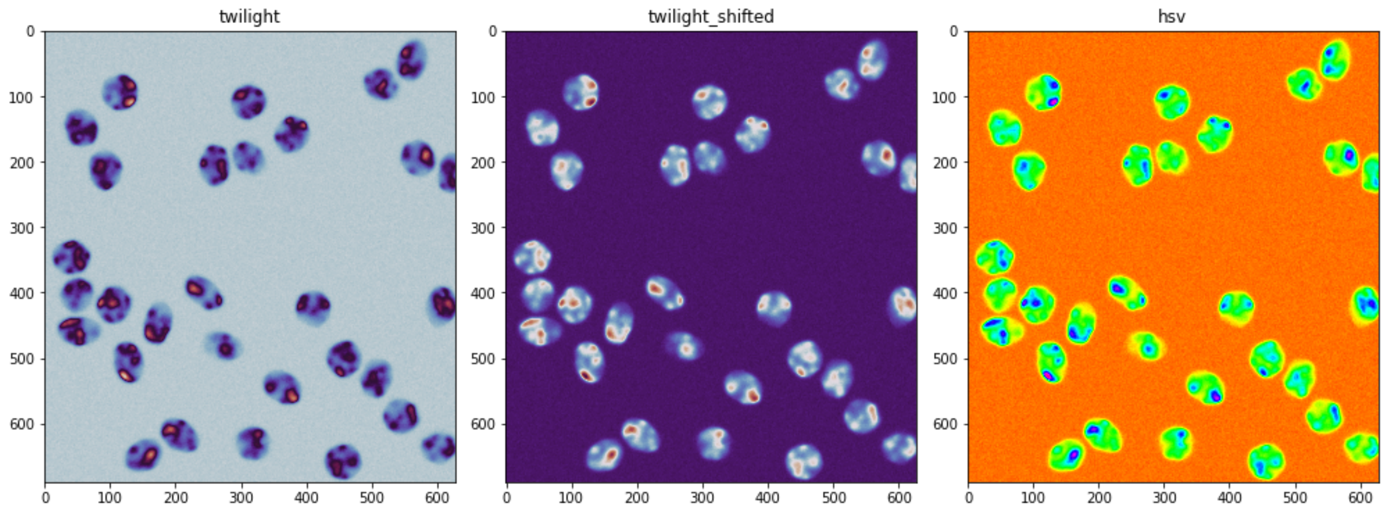


The sequential colours are pretty, but not realistic... Let us explore the other options:

In [39]:
```
# Diverging
path = walk(current_directory + "\\002_DataSetExploration\\SimulatedSamples")

rgb_img = plt.imread( current_directory + "\\002_DataSetExploration\\SimulatedSamples\\t
colour_choices = ["PiYG", "PRGn", "BrBG", "PuOr", "RdGy", "RdBu", "RdYlBu", "RdYlGn", "c

fig = plt.figure(figsize=(16, 12))

for i in range(len(colour_choices)):
    fig.add_subplot(3, 4, i+1)
    plt.title(colour_choices[i])
    plt.imshow(rgb_img, cmap=colour_choices[i])

plt.tight_layout()
plt.show()
```

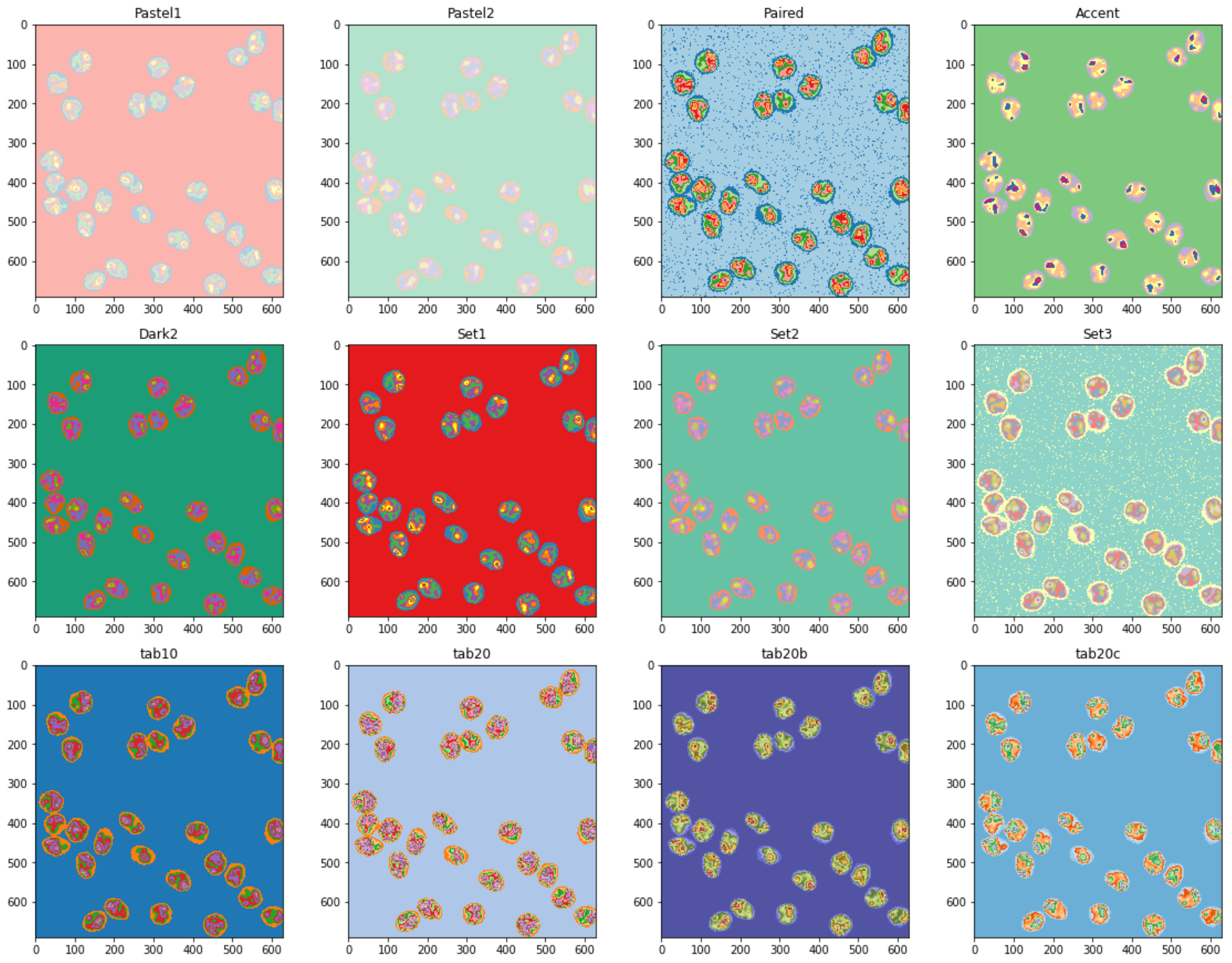


```
In [40]: # Cyclic
         path = walk(current_directory + "\\002_DataSetExploration\\SimulatedSamples")

         rgb_img = plt.imread( current_directory + "\\002_DataSetExploration\\SimulatedSamples\\t
         colour_choices = ["twilight", "twilight_shifted", "hsv"]

         fig = plt.figure(figsize=(14, 8))

         for i in range(len(colour_choices)):
             fig.add_subplot(1, 3, i+1)
             plt.title(colour_choices[i])
             plt.imshow(rgb_img, cmap=colour_choices[i])

         plt.tight_layout()
         plt.show()
```



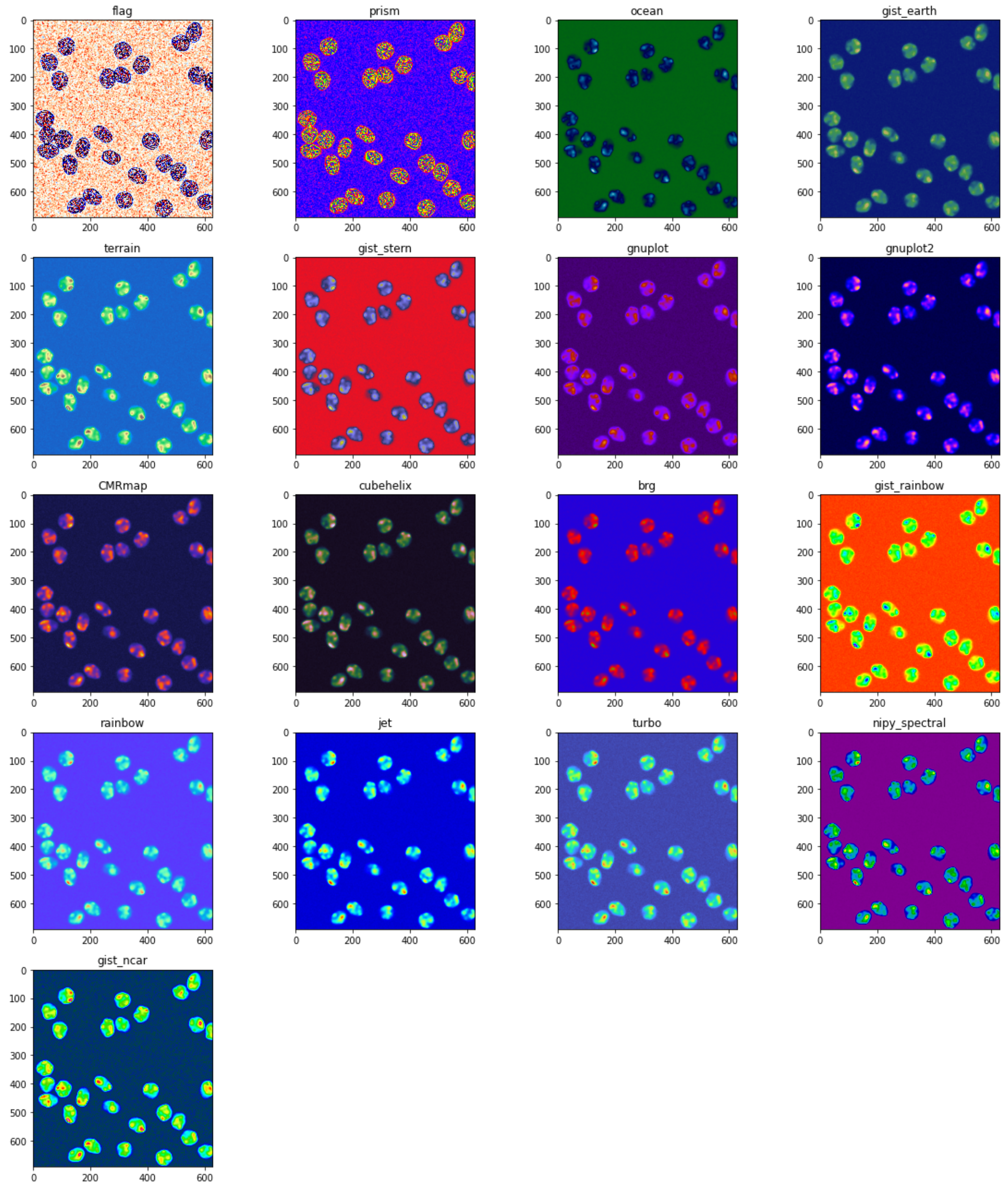
In [41]:

```
# Qualitative
path = walk(current_directory + "\\002_DataSetExploration\\SimulatedSamples")

rgb_img = plt.imread( current_directory + "\\002_DataSetExploration\\SimulatedSamples\\t
colour_choices = ["Pastel1", "Pastel2", "Paired", "Accent", "Dark2", "Set1", "Set2", "Se

fig = plt.figure(figsize=(16, 16))

for i in range(len(colour_choices)):
    fig.add_subplot(4, 4, i+1)
    plt.title(colour_choices[i])
    plt.imshow(rgb_img, cmap=colour_choices[i])

plt.tight_layout()
plt.show()
```



In [42]:

```
# Miscellaneous
path = walk(current_directory + "\\002_DataSetExploration\\SimulatedSamples")

rgb_img = plt.imread( current_directory + "\\002_DataSetExploration\\SimulatedSamples\\t
colour_choices = ['flag', 'prism', 'ocean', 'gist_earth', 'terrain',
                  'gist_stern', 'gnuplot', 'gnuplot2', 'CMRmap',
                  'cubehelix', 'brg', 'gist_rainbow', 'rainbow', 'jet',
                  'turbo', 'nipy_spectral', 'gist_ncar']

fig = plt.figure(figsize=(16, 18))

for i in range(len(colour_choices)):
    fig.add_subplot(5, 4, i+1)
    plt.title(colour_choices[i])
    plt.imshow(rgb_img, cmap=colour_choices[i])

plt.tight_layout()
plt.show()
```



From the above pictures, we can see a lot of options, some of which capture more information than others!

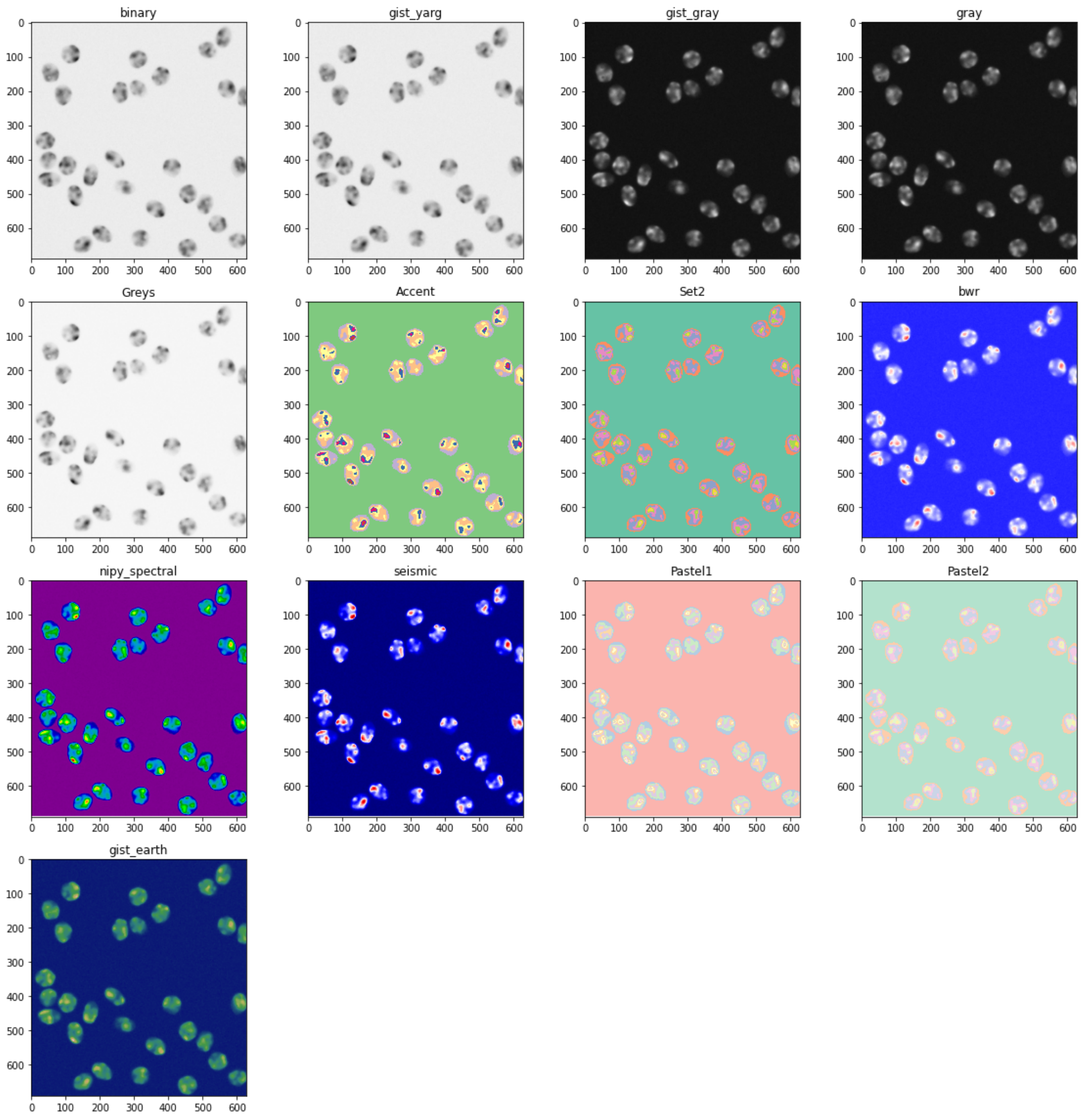
'binary', 'gist_yarg', 'gist_gray', 'gray', 'Greys' are the obvious ones we can use.

However, some promising colours may include: 'Accent', 'Set2', 'bwr', 'nipy_spectral', 'seismic', 'pastel1', 'pastel2', 'gist_earth'

These combined choices are shown here:

In [43]:
```
# Mix and Match
path = walk(current_directory + "\\002_DataSetExploration\\SimulatedSamples")
```

```
rgb_img = plt.imread( current_directory + "\\002_DataSetExploration\\SimulatedSamples\\t
colour_choices = ['binary', 'gist_yarg', 'gist_gray', 'gray', 'Greys', 'Accent', 'Set2',
                  'nipy_spectral', 'seismic', 'Pastel1', 'Pastel2', 'gist_earth']



fig = plt.figure(figsize=(16, 16))

for i in range(len(colour_choices)):
    fig.add_subplot(4, 4, i+1)
    plt.title(colour_choices[i])
    plt.imshow(rgb_img, cmap=colour_choices[i])

plt.tight_layout()
plt.show()
```



In another notebook, we can compare the colour choices for the collection of datasets - but for this notebook, we will use *gray* for the simulated colour images, for consistency

dataset Video Generation

This part of the notebook generates the videos we will use to understand the data

That seems to fix it! Let's try use that with the simulated folder of images

Upon closer experimentation, using the quantity of images in a directory as a frame rate results in a video that is super duper short in length! Instead, the author used a frame rate of 10, in order to more clearly see the journey of the cells.

Below is the code to generate the 40 videos showing the data. It may take 15 minutes to run

In [44]:
```
# only progress if files don't exist
desired_folder = "..\\..\\Comp700_VideosOfDataSets_Colour"
makeVideos = False

if (exists(current_directory + "\\" + desired_folder)):
    # Now, go to directory and verify all is there
    path = walk(current_directory + "\\" + desired_folder)

    count = 0
    for root, dirs, files in path:
        for item in files:
            count += 1

    if (count == 40):
        print("All Videos exist already!")
    else:
        print("Not all Videos exist")
        makeVideos = True
else:
    makeVideos = True

if (makeVideos):
    path = walk(current_directory + "\\" + data_sets) # reset path

    sub_directory_choice = ["01", "02"]

    i = -1
    output_video = cv2.VideoWriter()
    frames_per_second = 10
    petri_dish_images = False

    # Generates Colour Videos
    for root, dirs, files in path:
        for item in files:
            if ("man_" not in item) and (".zip" not in item):
                # update on first element only
                if ("t0000.tif" == item) or ("t000.tif" == item):
                    petri_dish_images = True
                    i += 1
                    index = i // 2 # used for output video as 2 copies for each director

                    size = (image_size_array[i][1], image_size_array[i][0] ) # notice or
                    fileName = "Color_" + directory_array[index] + "_" + sub_directory_c

                    output_video = cv2.VideoWriter(
                        fileName,
                        cv2.VideoWriter_fourcc(*'DIVX'),
                        frames_per_second,
                        size
                    )

                # used to update the pictures for the Simulated Videos
                if ("+" in directory_array[index]):
                    img = plt.imread( location_array[i] + item ) # grayscale
                    plt.imsave("temp.jpg", img, cmap='gray')
                    img = cv2.imread( "temp.jpg")
                else:
                    # Colour Video
                    img = cv2.imread( (location_array[i] + item) )

                output_video.write(img)

            else:
                petri_dish_images = False
                break


        if (petri_dish_images):
            cv2.destroyAllWindows()
            output_video.release()
            print("Video finished for ", fileName, sep="")
            petri_dish_images = False # update incase next iteration containes empty arr
```

```
All Videos exist already!
```

Now, remove the final temporary picture

In [45]:
```
# from os import remove
if (exists("temp.jpg")):
    remove("temp.jpg")
```

ouput_video.release() saves the contents of the files into the current directory, so the next block of code moves them to our a seperate folder

In [46]:
```
# from os.path import join
# from shutil import move # moves and replaces files

# only progress if files don't exist
desired_folder = "..\\..\\Comp700_VideosOfDataSets_Colour"
if (exists(current_directory + "\\" + desired_folder)):
    print("Videos already exist!")
else:
    try:
        mkdir(desired_folder)
    except FileExistsError:
        pass
    except:
        print("Unknown Error Encountered...")

    path = walk(current_directory)

    # count = 0
    for root, dirs, files in path:
        for item in files:
            if (".mp4" in item):
                # count += 1
                origin_path = current_directory
                new_destination = current_directory + "\\" + desired_folder
                move(join(current_directory, item), join(new_destination, item)) # shoul

    # Now, go to directory and verify all is there
    path = walk(current_directory + "\\" + desired_folder)
    count = 0
    for root, dirs, files in path:
        for item in files:
            count += 1

    if (count == 40):
        print("All Videos Moved Successfully!")
    else:
        print("Not all Videos Moves Successfully")
```

```
Videos already exist!
```

The simulated videos have a bit of noise. This probably comes from the Conversion between plt and cv2. To try rememdy this in the future, let us explore if we can create grayscale videos:

Here is the same cell block as above, but generates Grayscale Videos

In [47]:
```
# only progress if files don't exist
desired_folder = "..\\..\\Comp700_VideosOfDataSets_Grayscale"
makeVideos = False

if (exists(current_directory + "\\" + desired_folder)):
    # Now, go to directory and verify all is there
    path = walk(current_directory + "\\" + desired_folder)

    count = 0
    for root, dirs, files in path:
        for item in files:
            count += 1

    if (count == 40):
        print("All Videos already exist!")
    else:
        print("Not all Videos exist")
        makeVideos = True
else:
    makeVideos = True

if (makeVideos):
    path = walk(current_directory + "\\" + data_sets) # reset path

    sub_directory_choice = ["01", "02"]

    i = -1
    output_video = cv2.VideoWriter() # needed for compiler to process loop below, update
    frames_per_second = 10
    petri_dish_images = False

    # Grayscale Videos
    for root, dirs, files in path:
        for item in files:
            if ("man_" not in item) and (".zip" not in item):
                # update on first element only
                if ("t0000.tif" == item) or ("t000.tif" == item):
                    petri_dish_images = True
                    i += 1
                    index = i // 2 # used for output video as 2 copies for each director

                    size = (image_size_array[i][1], image_size_array[i][0] ) # notice or
                    fileName = "Grayscale_" + directory_array[index] + "_" + sub_directo

                    # isColor below is what we use to create grayscale videos!
                    output_video = cv2.VideoWriter(
                        fileName,
                        cv2.VideoWriter_fourcc(*'DIVX'),
                        frames_per_second,
                        size,
                        isColor=False
                    )

                # used to update the pictures for the Simulated Videos
                # write to disk as grayscale, then read in again
                if ("+" in directory_array[index]):
                    img = plt.imread( location_array[i] + item ) # grayscale
                    plt.imsave("temp.jpg", img, cmap="gray")
                    img = cv2.imread( "temp.jpg", cv2.IMREAD_GRAYSCALE)

                else:
                    img = cv2.imread( (location_array[i] + item), cv2.IMREAD_GRAYSCALE)

                output_video.write(img)

            else:
                petri_dish_images = False
                break


        if (petri_dish_images):
            cv2.destroyAllWindows()
            output_video.release()
            print("Video finished for ", fileName, sep="")
            petri_dish_images = False # update incase next iteration containes empty arr
```

```
All Videos already exist!
```

Now, let us move to a grayscale video folder:

In [48]:
```
# from os.path import join
# from shutil import move # moves and replaces files

# only progress if files don't exist
desired_folder = "..\\..\\Comp700_VideosOfDataSets_Grayscale"
if (exists(current_directory + "\\" + desired_folder)):
    print("Videos already exist!")
else:

    try:
        mkdir(desired_folder)
    except FileExistsError:
        pass
    except:
        print("Unknown Error Encountered...")

    path = walk(current_directory)

    # count = 0
    for root, dirs, files in path:
        for item in files:
            if (".mp4" in item):
                # count += 1
                origin_path = current_directory
                new_destination = current_directory + "\\" + desired_folder
                move(join(current_directory, item), join(new_destination, item)) # shoul

    # Now, go to directory and verify all is there
    path = walk(current_directory + "\\" + desired_folder)

    count = 0
    for root, dirs, files in path:
        for item in files:
            count += 1

    if (count == 40):
        print("All Videos Moved Successfully!")
    else:
        print("Not all Videos Moves Successfully")
```

```
Videos already exist!
```

Now, remove the final temporary picture

In [49]:
```
# from os import remove
if (exists("temp.jpg")):
    remove("temp.jpg")
```

The grayscale videos successfully generate and the noise present in the simulated videos goes away! Fantastic! We may need to incorporate these findings later on

Conclusions

This section of the notebook summarises the findings so far

The dataset has 2 folders, one contains the original zipped data, and the other the extracted data.

The extracted data is broken up into 2 sets: Training Data, and Challenge Data. The Challenge Data is indicated with a (1) in the folder name.

The distinction between the Training Data and the Challenge Data is that the Training Data contains additional folders of manual segmentation and manual tracking of the images. This will be useful later on when training the models.

To assist with understanding of the data, the author has created a Matplotlib plot showing 10 samples of the images from the 10 data sets (Training Data)

The author has also generated Videos of each folder, in both colour and grayscale, to show the movement of the cells over time.

Much later on, we will create these files in a similar way in order to show tracking and segmentation!

The most significant findings from the work is: The pictures may need to be read in via Matplotlib, and saved onto disk in order for OpenCV to register the information in the image.

The author is not entirely sure why this is necessary, but it may be the case that the images are saved in a format that OpenCV struggles to process. Whatever the reason is though, one of the processing tips may be to do a bulk save of all images via Matplotlib first!

Thank You!