Comp Photography
Final Project

Tanmaya Kumar

Fall 2017

tkumar31@gatech.edu

Painting Photographs

I have been huge fan of Van Gogh, and recently watched "Loving Vincent". While each frame of that movie is hand painted, it struck me as an interesting problem if we could take a set of photographs and convert them to be hand painted pictures. My goal was be take pictures and make them look like they were hand painted.

The Goal of Your Project

Original project scope: The original goal of my project was to create a program to convert photos taken by cameras look like they were hand painted.

Initially I wanted the work to be a set of 10 photographs that could come together to form a continuous GIF image. Making it look like a painted GIF.

What motivated you to do this project?

I recently watched “Loving Vincent” that gave me the idea of a painted video. While all the frames in the movie are hand painted, it felt like something I could do with image processing, because I really cannot paint.

Scope Changes

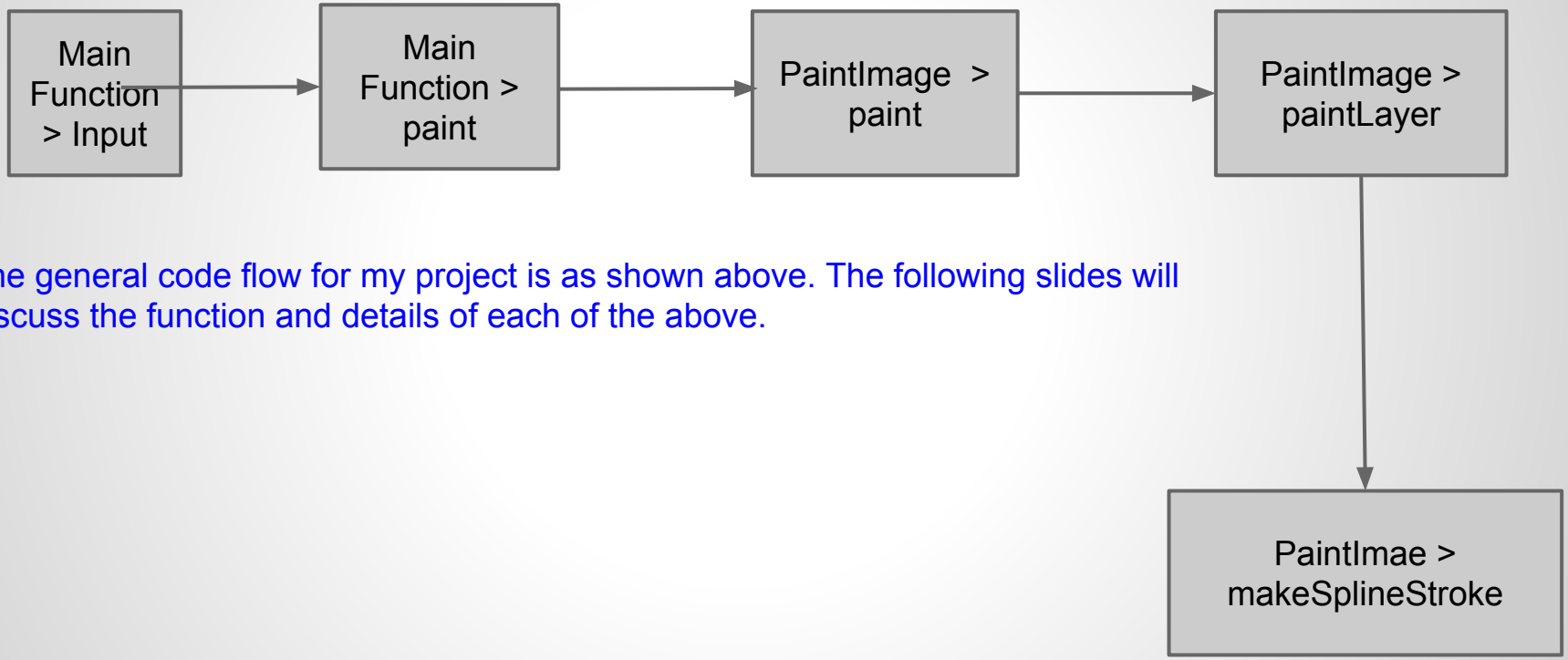
- Did you run into issues that required you to change project scope from your proposal?
  - I **didn't run into issues**, **but I did change scope** for a different reason.
  - I realized that for a painted GIF image, I would have to paint the same/similar image over and over again. Also, after the paint strokes were on there, the paintings were liable to look very different and wouldn't be a great loop.
- Give a detailed explanation of what changed
  - I realized that with a repetitive image I would not get to create a good showcase and attempt new things. Instead I decided to experiment with different kinds of images with different strokes, brushes, and, other parameters.

Showcase

Input

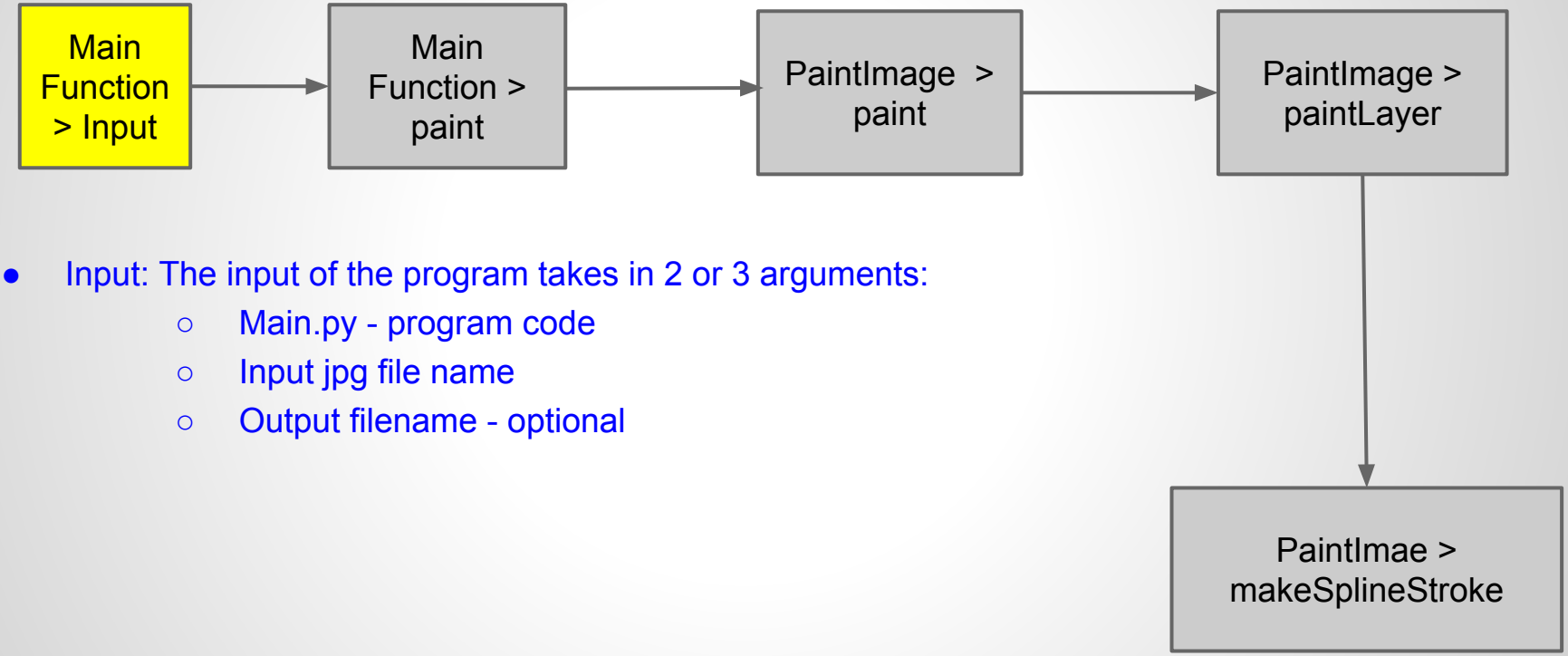
Output

Project Pipeline



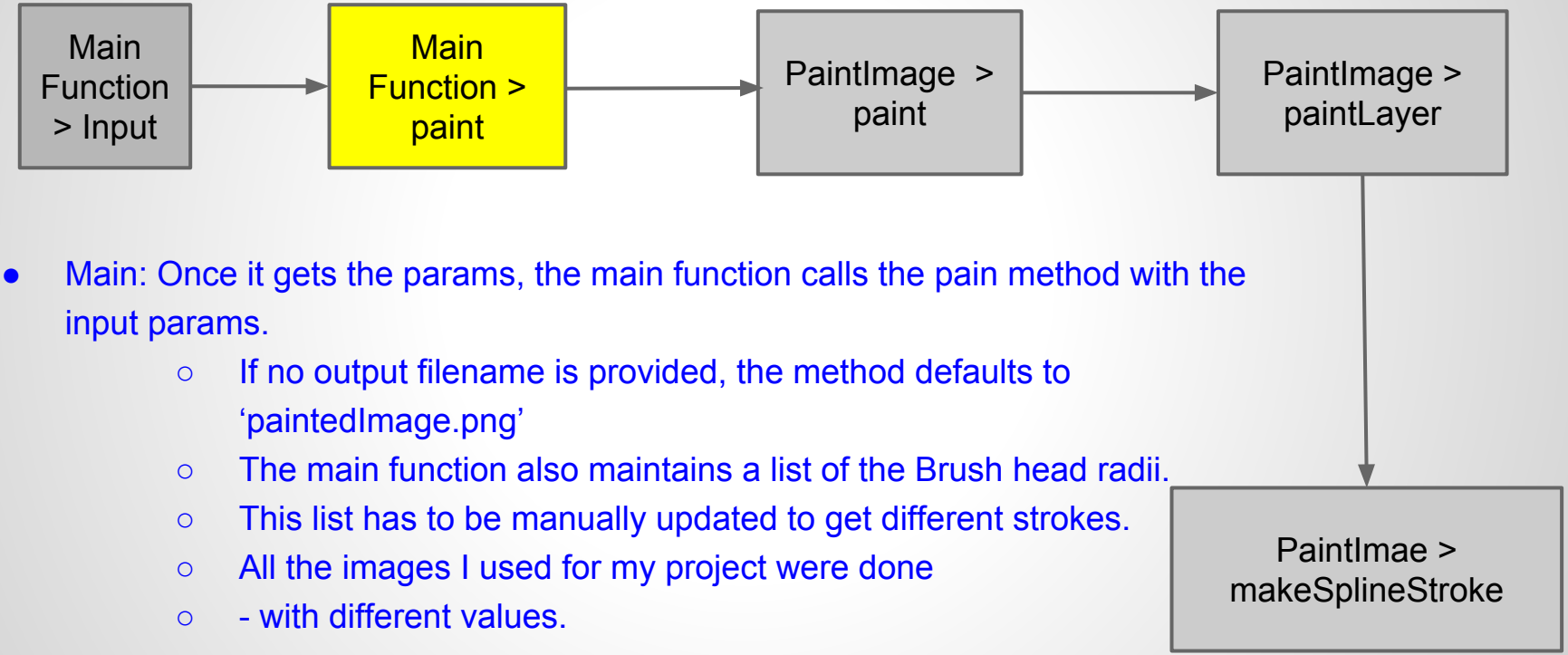
The general code flow for my project is as shown above. The following slides will discuss the function and details of each of the above.

Project Pipeline



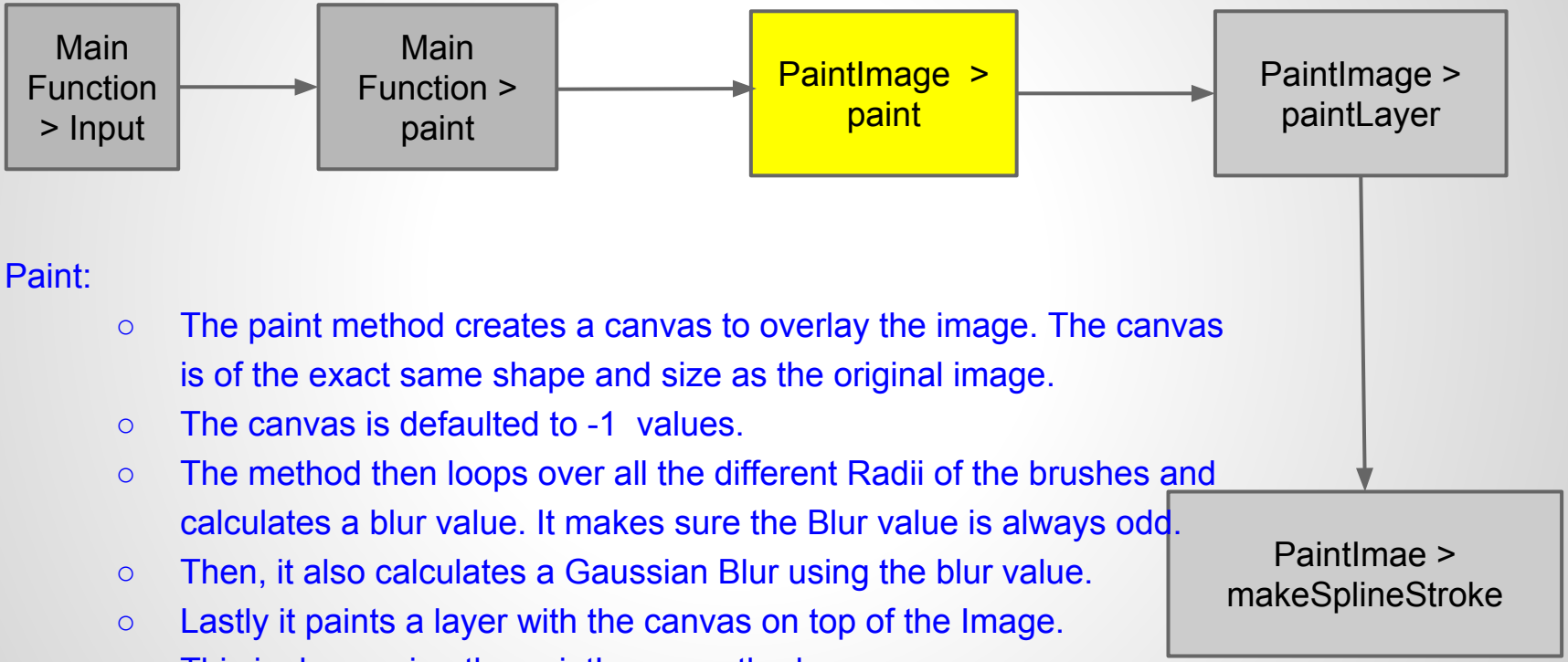
- Input: The input of the program takes in 2 or 3 arguments:
  - Main.py - program code
  - Input jpg file name
  - Output filename - optional

Project Pipeline



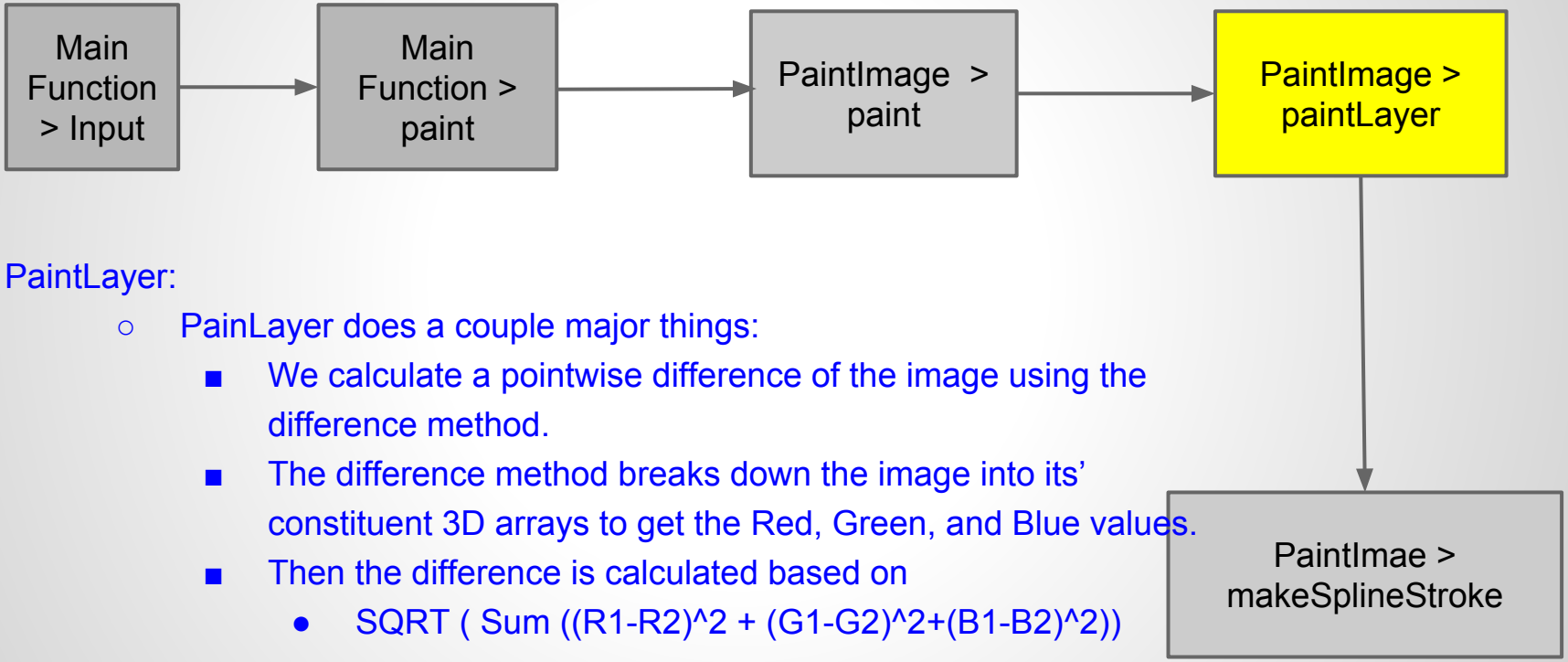
- Main: Once it gets the params, the main function calls the pain method with the input params.
    - If no output filename is provided, the method defaults to ‘paintedImage.png’
    - The main function also maintains a list of the Brush head radii.
    - This list has to be manually updated to get different strokes.
    - All the images I used for my project were done
    - - with different values.

Project Pipeline



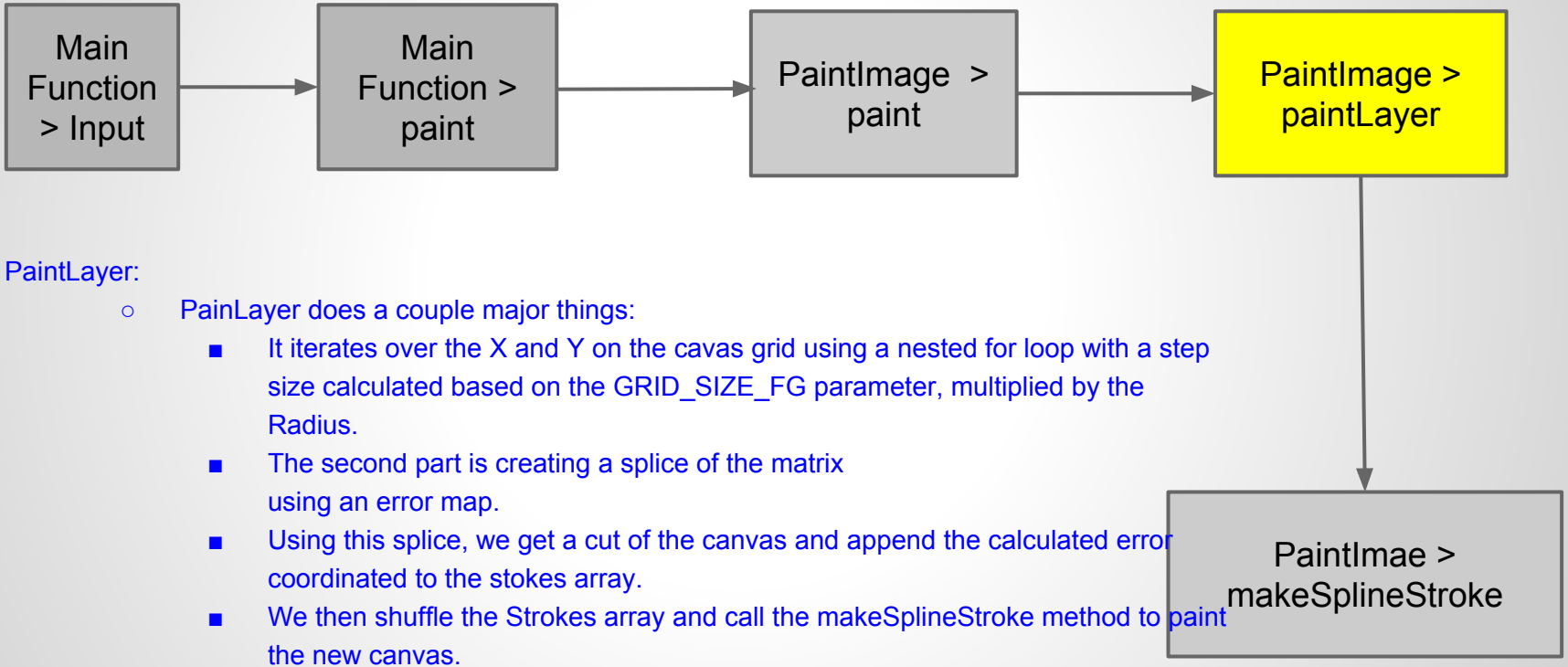
- Paint:
  - The paint method creates a canvas to overlay the image. The canvas is of the exact same shape and size as the original image.
  - The canvas is defaulted to -1 values.
  - The method then loops over all the different Radii of the brushes and calculates a blur value. It makes sure the Blur value is always odd.
  - Then, it also calculates a Gaussian Blur using the blur value.
  - Lastly it paints a layer with the canvas on top of the Image.
  - This is done using the paintLayer method.

Project Pipeline



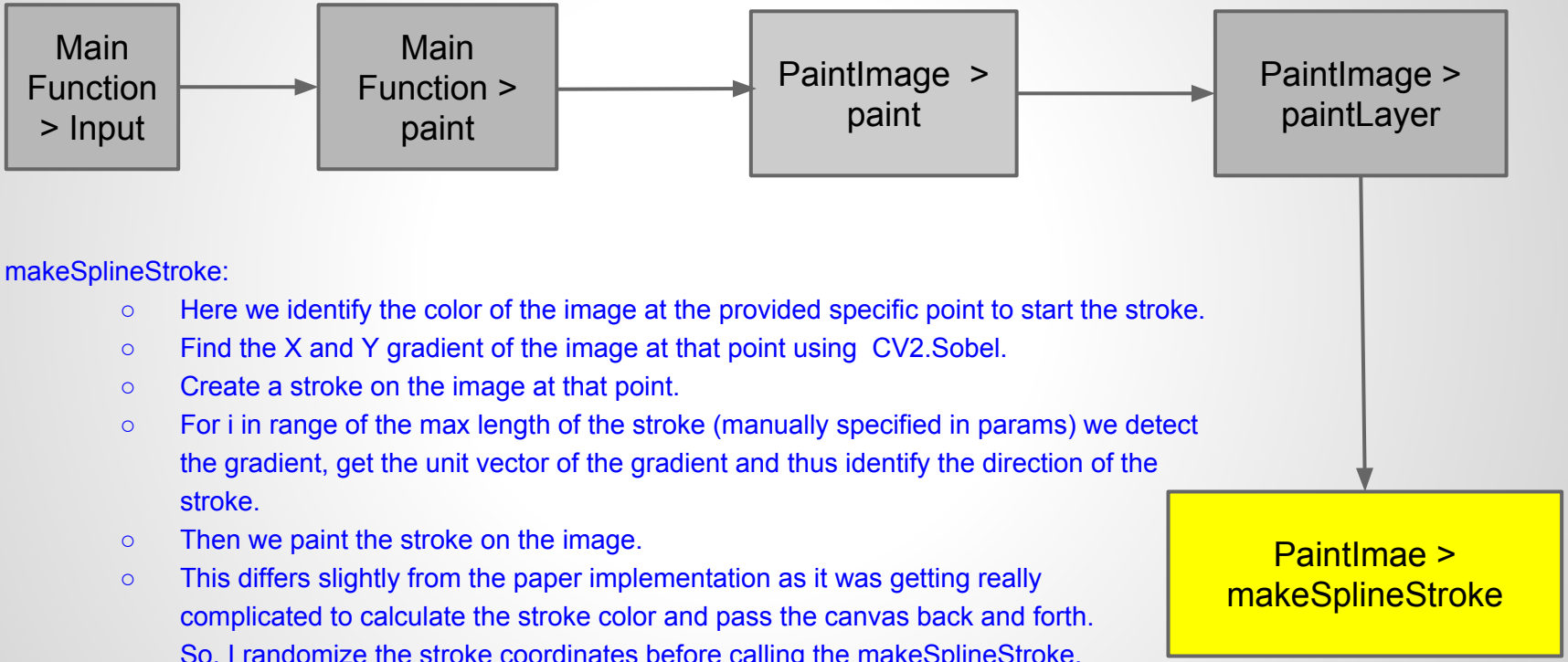
- PaintLayer:
  - PainLayer does a couple major things:
    - We calculate a pointwise difference of the image using the difference method.
    - The difference method breaks down the image into its' constituent 3D arrays to get the Red, Green, and Blue values.
    - Then the difference is calculated based on
      - SQRT (Sum ((R1-R2)^2 + (G1-G2)^2+(B1-B2)^2))

Project Pipeline Cont.



- PaintLayer:
  - PainLayer does a couple major things:
    - It iterates over the X and Y on the cavas grid using a nested for loop with a step size calculated based on the GRID_SIZE_FG parameter, multiplied by the Radius.
    - The second part is creating a splice of the matrix using an error map.
    - Using this splice, we get a cut of the canvas and append the calculated error coordinated to the stokes array.
    - We then shuffle the Strokes array and call the makeSplineStroke method to paint the new canvas.

Project Pipeline Cont.



- makeSplineStroke:
  - Here we identify the color of the image at the provided specific point to start the stroke.
  - Find the X and Y gradient of the image at that point using CV2.Sobel.
  - Create a stroke on the image at that point.
  - For i in range of the max length of the stroke (manually specified in params) we detect the gradient, get the unit vector of the gradient and thus identify the direction of the stroke.
  - Then we paint the stroke on the image.
  - This differs slightly from the paper implementation as it was getting really complicated to calculate the stroke color and pass the canvas back and forth. So, I randomize the stroke coordinates before calling the makeSplineStroke.

Demonstration: Result Sets

- **Image: Apple**
- Here I converted a picture of an apple to a painted image. This was done with one brush of radii 1. The black dots are evident because when painting the grid offset default is 0, which in RGB is black. Hence the black spots. Settings below were updated manually in the code to get the outputs.

- BRUSH_HEAD_RADII = [1]
- BLUR_FACTOR = 3
- GRID_SIZE_FG = 2.5
- T = 50 # Threshold
- MIN_STROKE_LENGTH = 1
- MAX_STROKE_LENGTH = 21
- CURVATURE_FILTER = 0.3

Demonstration: Result Sets

- **Image: Netherlands Gateway**
- Here I converted a picture of a gateway i took in Netherlands to a painted image. This was done with 5 brush strokes, with the following settings. Increasing, then decreasing brush radii, and high curvature. Settings below were updated manually in the code to get the outputs.

- BRUSH_HEAD_RADII = [1, 2, 3, 2, 1]
- BLUR_FACTOR = 3
- GRID_SIZE_FG = 2.5
- T = 50 # Threshold
- MIN_STROKE_LENGTH = 1
- MAX_STROKE_LENGTH = 21
- CURVATURE_FILTER = 0.3

Demonstration: Result Sets

- **Image: Vulture**
- Here I converted a picture of a vulture to a painted image. This was done with 4 brush strokes in decreasing order of radii, with the following settings. This was one of the first pictures that worked for me. Settings below were updated manually in the code to get the outputs.

- BRUSH_HEAD_RADII = [5, 3, 2, 1]
- BLUR_FACTOR = 2
- GRID_SIZE_FG = 2.5
- T = 25 # Threshold
- MIN_STROKE_LENGTH = 1
- MAX_STROKE_LENGTH = 10
- CURVATURE_FILTER = 0.2

Demonstration: Result Sets

- **Image: Sunflower**
- Here I converted a picture of a sunflower to a painted image. This is one of my favorites, partly because I am huge Van Gogh fan, who painted a lot of sunflowers. Done with 5 brushes again but with a different grid pattern, a lower blur and smaller curvature stroke. This helped keep the original image more intact. Settings below were updated manually in the code to get the outputs.

- BRUSH_HEAD_RADII = [1, 2, 3, 2, 1]
- BLUR_FACTOR = 2
- GRID_SIZE_FG = 2.5
- T = 25 # Threshold
- MIN_STROKE_LENGTH = 1
- MAX_STROKE_LENGTH = 10
- CURVATURE_FILTER = 0.2

Demonstration: Result Sets

- **Image: Cat**
- Here I converted a picture of a cat to a painted image. This was done with a fibonacci sequenced brush stroke and a high blur factor. This was the second of two attempts at this picture. Settings below were updated manually in the code to get the outputs.

- BRUSH_HEAD_RADII = [1, 2, 3, 5, 8, 13, 21, 34]
- BLUR_FACTOR = 2
- GRID_SIZE_FG = 3
- T = 50 # Threshold
- MIN_STROKE_LENGTH = 1
- MAX_STROKE_LENGTH = 13
- CURVATURE_FILTER = 0.3

Demonstration: Result Sets

- **Image: Flowers**
- Here I converted a picture of a flower bed to a painted image. Like the cat, this was also done with a fibonacci sequence brush with high curvature. Settings below were updated manually in the code to get the outputs.

- BRUSH_HEAD_RADII = [1, 2, 3, 5, 8, 13, 21, 34]
- BLUR_FACTOR = 2
- GRID_SIZE_FG = 3
- T = 50 # Threshold
- MIN_STROKE_LENGTH = 1
- MAX_STROKE_LENGTH = 13
- CURVATURE_FILTER = 0.3

Project Development

Use several slides for a detailed discussion of how you developed your project outputs. Tell your story. Difficulties with developing your code may also be discussed here, or in the following Computation section.

Include:

- Narrative of your progress and issues you faced.
  - The inspiration for the project came from the movie ‘Loving Vincent’
  - I started by searching for ‘painting images in python’ and came across the research paper Aaron Hertzmann
  - I started by breaking down the research paper and identifying the individual methods to make. I took a lot of inspiration for the structure from the projects we had done in class before.
  - It was easy enough to break down the work but there were two issues that I faced that were my biggest problems.

Project Development (cont'd)

- Include descriptions of problems, and how you handled them.
  - First issue I realized was not being able to actually identify the right boundaries of the region splice near x,y. I had a lot of issues because I interpreted the paper's x-grid/2...x+grid/2 instruction literally. It was only after many - many index out of bound errors that I realized the images in python array are y,x (height, width) so I was putting the index backwards.
  - The second set of issues I had was my program was just painting a 255 x 255 grid of the image instead of the whole image. It took a while before I realized I was clipping the canvas to a smaller size, I had misinterpreted np.clip, to refer to values not length.

Project Development (cont'd)

- Include both good and failed interim results (from various parts of your work and pipeline)

Here you can see a clear set of lines making a square. When sized it is a square of 255 x 255. Took a while for me to figure that out.

Project Development (cont'd)

- What did you finish, what is not finished?
  - I managed to finish what my intention for the project was, even though I did end up changing scope a bit. I can now comfortably produce images that look roughly painted, and do understand the impact that blur, and curvature have on the product.
  - I wanted to completely parameterize the constraints on the program to create images on the fly that looked the way I wanted to. That will definitely be some future work for me.
  - Also, the method only works for images greater than 255 x 255. I would like to account for that in the code.
- What would you do differently?
  - Account for the image size in the code.
  - I would really like to simplify some of the logic in the make spline stroke method.
  - I also would really like to fix the dots that are appearing on the painted image.

Computation: Code Functional Description

Walk through your code functions

- Difference Method:

```
def difference(canvas, referenceImage):
    r1, g1, b1 = np.split(canvas, 3, axis=2)
    r2, g2, b2 = np.split(referenceImage, 3, axis=2)

    rdiff = (r1 - r2) ** 2
    gdiff = (g1 - g2) ** 2
    bdiff = (b1 - b2) ** 2

    sqrootVal = np.sqrt(rdiff + gdiff + bdiff)
    return sqrootVal
```

  - The difference method calculates a point by point difference between two images, based on their RGB values.

- Explain the major algorithms. If you used them, show that you understand them in your own words. Do not copy from your technical paper.
- Include descriptions of libraries/packages that you used
- Code discussion may be incorporated in the Project Development section if that works better for

Computation: Code Functional Description

- Paint Method:

```
def paint(image_array, brush_head_radii):
    canvas = np.zeros(image_array.shape)
    print image_array.shape, 'image shape'
    canvas.fill(-1)

    for radius in brush_head_radii:
        blur = BLUR_FACTOR * radius
        blur_val = int(blur)
        if blur_val % 2 == 0:
            blur_val += 1
        referenceImage = cv2.GaussianBlur(image_array, (blur_val, blur_val), 0.4,0)
        paintLayer(canvas, referenceImage, radius)
    return canvas
```

  - The paint method creates a canvas the same size as an image with default value of -1.
  - The iterate through the list of Radii and calculates an odd blur value for a GaussianBlur.

Computation: Code Functional Description

- PaintLayer Method:
  - The goal for the paintlayer is to Create a grid overlaying the image And iterating over the image to pick A min and max y,x indicating grid of Pixels on the image with a large error Factor.
  - This is then used to calculate the Grid from the over image difference Layer
  - We then append it to an array that Maintains these points for painting Over later.

```
def paintLayer(canvas, referenceImage, Ri):
    S = [
    img_diff = difference(canvas, referenceImage)
    img_diff = np.squeeze(img_diff)
    step_size = int(GRID_SIZE_FG * Ri)

    for x in xrange(0, canvas.shape[1], step_size):
        for y in xrange(0, canvas.shape[0], step_size):
            x_region_min = max(int(x - (step_size / 2)), 0)
            if (x_region_min > canvas.shape[1]):
                x_region_min = canvas.shape[1] - 1
            x_region_max = min(int(x + (step_size / 2)), canvas.shape[1] - 1)
            y_region_min = max(int(y - (step_size / 2)), 0)

            if (y_region_min > canvas.shape[0]):
                y_region_min = canvas.shape[0] - 1
            y_region_max = min(int(y + (step_size / 2)), canvas.shape[0] - 1)
            M_coordinates = np.ix_([y_region_min, y_region_max], [x_region_min, x_region_max])
            M = img_diff[M_coordinates]
            areaError = np.sum(M) / (step_size ** 2)
            if (areaError > T):
                (areaError_y, areaError_x) = np.unravel_index(M.argmax(), M.shape)
                areaError_y += int(y - (float(step_size) / 2))
                areaError_x += int(x - (float(step_size) / 2))

                S.append((areaError_x, areaError_y, Ri, referenceImage, canvas))

    np.random.shuffle(S)
    for arx, ary, rad, ref_im, can in S:
        makeSplineStroke(arx, ary, rad, ref_im, can)
```

Computation: Code Functional Description

- MakeSplineStroke Method:

  This method takes the stroke coordinates and
  Calculates the gradients at that point.
  Uses the coordinates to calculate a circle and
  Paints over the image.
  Then it loops through the length of the strokes
  And calculates refernce points across the image
  By identifying the color differences and grad-
  -ients. The maxStrokeLenght is to avoid an
  Infinite loop.
  We then alight the gradient vectors on the
  Image and create a stroke at that point.

```
def makeSplineStroke(x0, y0, R, refImage, canvas):
    S = []
    strokeColor = np.squeeze(refImage[y0, x0, :]).astype(int).tolist()
    x, y = x0, y0
    lastDx, lastDy = 0, 0
    Gx = cv2.Sobel(refImage, cv2.CV_64F, 1, 0).mean(axis=2)
    Gy = cv2.Sobel(refImage, cv2.CV_64F, 0, 1).mean(axis=2)
    K = [(x0, y0)]
    cv2.circle(canvas, (int(x), int(y)), R, strokeColor, -1)
    for i in xrange(MAX_STROKE_LENGTH):
        if (i > MIN_STROKE_LENGTH) and (
                    np.abs(
                        np.sum(np.squeeze(refImage[int(y), int(x), :]).astype(int) - np.squeeze(
                            canvas[int(y), int(x), :]).astype(int))) <
                    np.abs(np.sum(np.squeeze(refImage[int(y), int(x), :]).astype(int) - strokeColor))):
            return K
        if np.sqrt(((Gy[int(y)][int(x)] ** 2) + (Gx[int(y)][int(x)] ** 2))) == 0:
            return K
        gx, gy = Gx[int(y)][int(x)], Gy[int(y)][int(x)]
        dx, dy = -gy, gx

        if (lastDx * dx) + (lastDy * dy) < 0:
            dx, dy = -dx, -dy

        dx = (CURVATURE_FILTER * dx) + ((1 - CURVATURE_FILTER) * (lastDx))
        dy = (CURVATURE_FILTER * dy) + ((1 - CURVATURE_FILTER) * (lastDy))

        dx /= np.sqrt(np.power(dx, 2) + np.power(dy, 2))
        dy /= np.sqrt(np.power(dx, 2) + np.power(dy, 2))

        x += R * dx
        y += R * dy

        if canvas.shape[1] > 255:
            x = max(min(x, canvas.shape[1]-1), 0)
        else:
            x = max(min(x, 254), 0)
        if canvas.shape[0] > 255:
            y = max(min(y, canvas.shape[0]-1), 0)
        else:
            y = max(min(y, 254), 0)
        lastDx, lastDy = dx, dy
        cv2.circle(canvas, (int(x), int(y)), R, strokeColor, -1)
```

Any additional details?

The one thing that I would like to talk about is an opportunity for improvement. There are certain things that could be handled better in the project such as handling of the white color. Images with white color tend to not be appropriately differenced and end up with black spots, I would really like to fix that.

Also I did run out of time for this so would have liked to implement a painted gif too.

Resources

Painterly Rendering with Curved Brush Strokes of Multiple Sizes Aaron Hertzmann: https://mrl.nyu.edu/publications/painterly98/hertzmann-siggraph98.pdf

StackOverflow: https://stackoverflow.com/questions/9404967/taking-the-floor-of-a-float

OpenCV: https://docs.opencv.org/3.1.0/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11ac52cf1

StackOverflow: https://stackoverflow.com/questions/27527440/opencv-error-assertion-failed-ksize-width-for-gaussianblur

OpenCV: http://answers.opencv.org/question/34314/opencv-error-failed-assertion/

Midterm project: cv.solbel

OpenCV Tutorial: http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_gradients/py_gradients.html

Python Image Search: https://www.pyimagesearch.com/2014/09/15/python-compare-two-images/

Scipy: https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.clip.html

Scipy: https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.split.html

StackOverflow: https://stackoverflow.com/questions/5446522/data-type-not-understood

StackOverflow: https://stackoverflow.com/questions/4971368/numpy-array-conversion-to-pairs

OpenCV: https://docs.opencv.org/3.1.0/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11ac52cf1

StackOverflow: https://stackoverflow.com/questions/27527440/opencv-error-assertion-failed-ksize-width-for-gaussianblur

OpenCV: http://answers.opencv.org/question/34314/opencv-error-failed-assertion/

StackOverflow: https://stackoverflow.com/questions/9404967/taking-the-floor-of-a-float

PixBay: https://pixabay.com/

Appendix: Your Code

Code Language: Python

List of code files:

- main.py
- *paintImage.py*

Credits or Thanks

- My wife for her patience and help looking at the index out of bounds issues!