

Labb 3

Elias Berglin

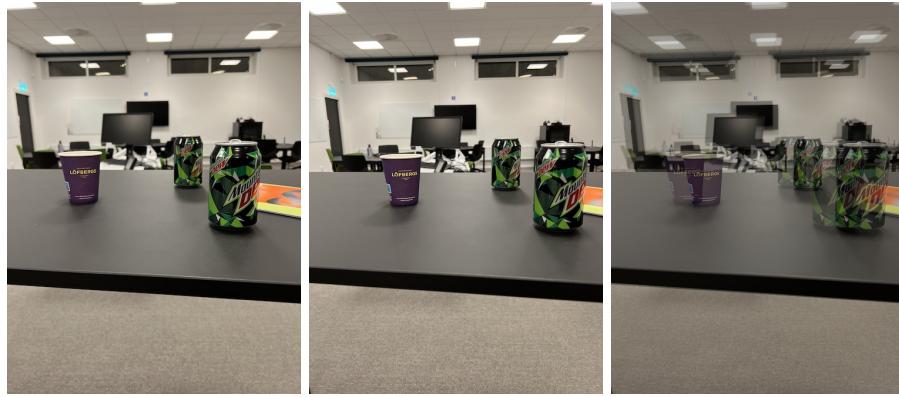
20th December 2021

1 Week 5

1.1 Assignment 5

1.1.1 Optical Flow

I used OpenCV's implementation for Optical Flow. This was originally used for videos but was adopted to be used with images instead. Coarse-to-fine algorithm improves our result on larger deltas by creating different layers of the image in different sizes. The smaller sizes helps the algorithm find features points. The iterations helps the algorithm to improve our results by correcting errors. In figure 1 we can see the input images and in figure 2 we can see the different results. We can see that the arrows correct themselves with more iterations and the points getting more accurate with more layers. The Python code can be found in appendix A



(a) First image

(b) Second image

(c) Images overlaid

Figure 1: The two input images for Optical Flow and the the images overlaid

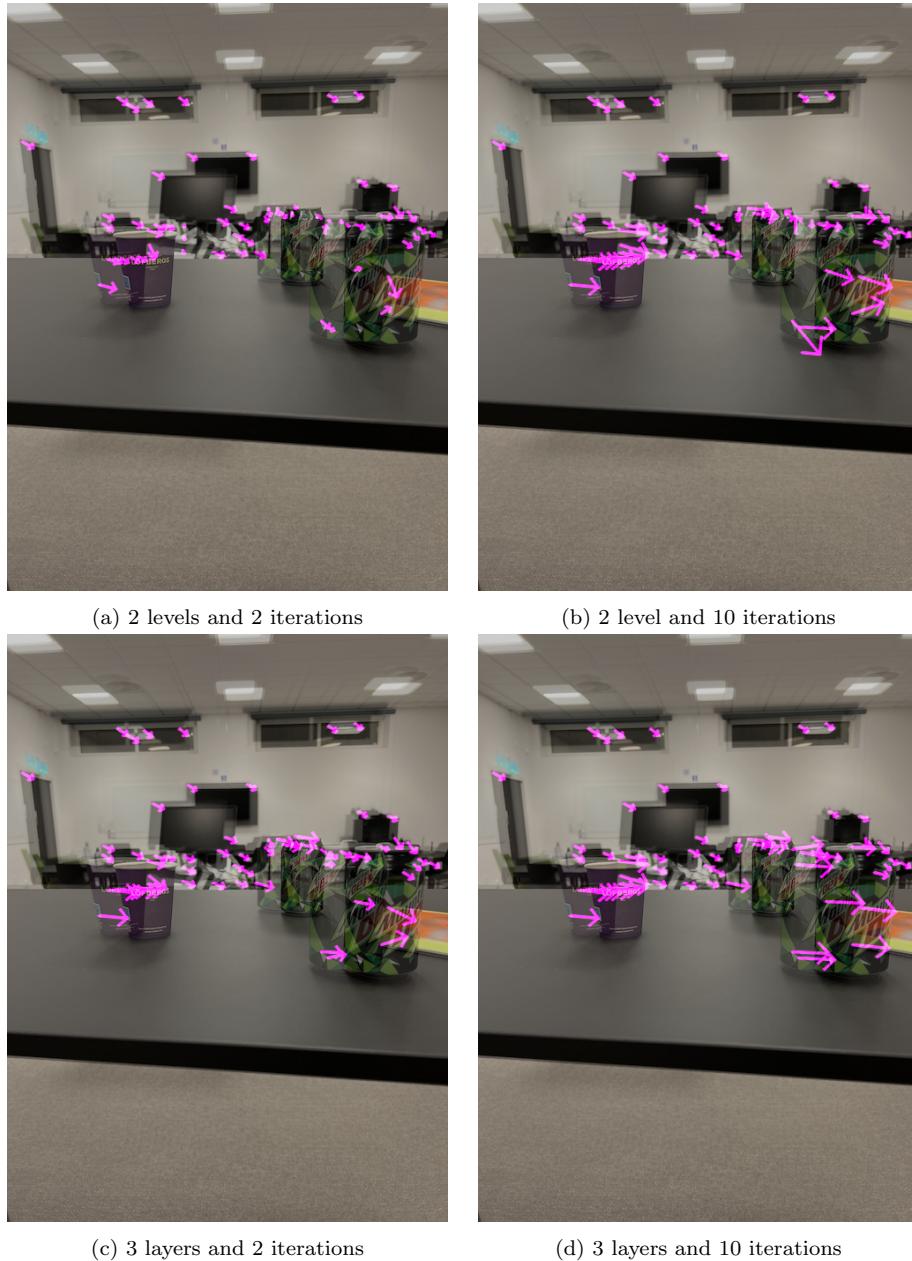


Figure 2: Resulting vectors from Lucas-Kanade method with different amount of pyramids and iterations

1.1.2 K-means

The K-means function can be found in equation 1. This algorithm assumes that all the datasets belong to a category μ .

$$\min_{\mu,y} \sum_i \|x_i - \mu_y\|^2 \quad (1)$$

In figure 3 we can see the different iterations. In figure 3a is the original data categorized. Then first iteration in figure 3b and then when we compare that to 3c we see that nothing changes and thus our final categorized are decided. These images were produced with a python script that can be found in appendix B.

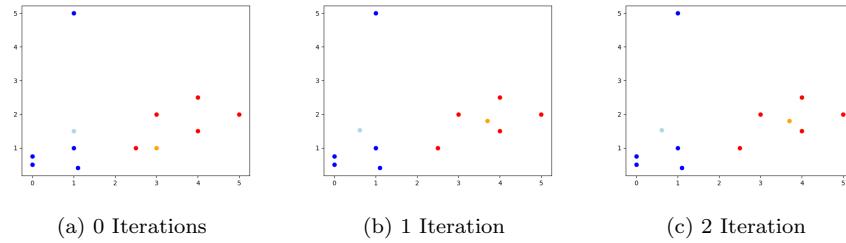


Figure 3: Iterations of K-means

1.1.3 Segmentation

I chose to implement K-means for color segmentation. My code can be found in appendix C. Input image can be found in figure 4 and output can be found in figure 5. The only parameters I can change is the max number of iterations and the number of clusters. The implementation I created stops the process if 70% of the clusters have moved less than 1 unit since the last iteration. So the number of max iterations just gives the algorithm more time to reach this criteria. The number of clusters will define how many colors the resulting image will have.

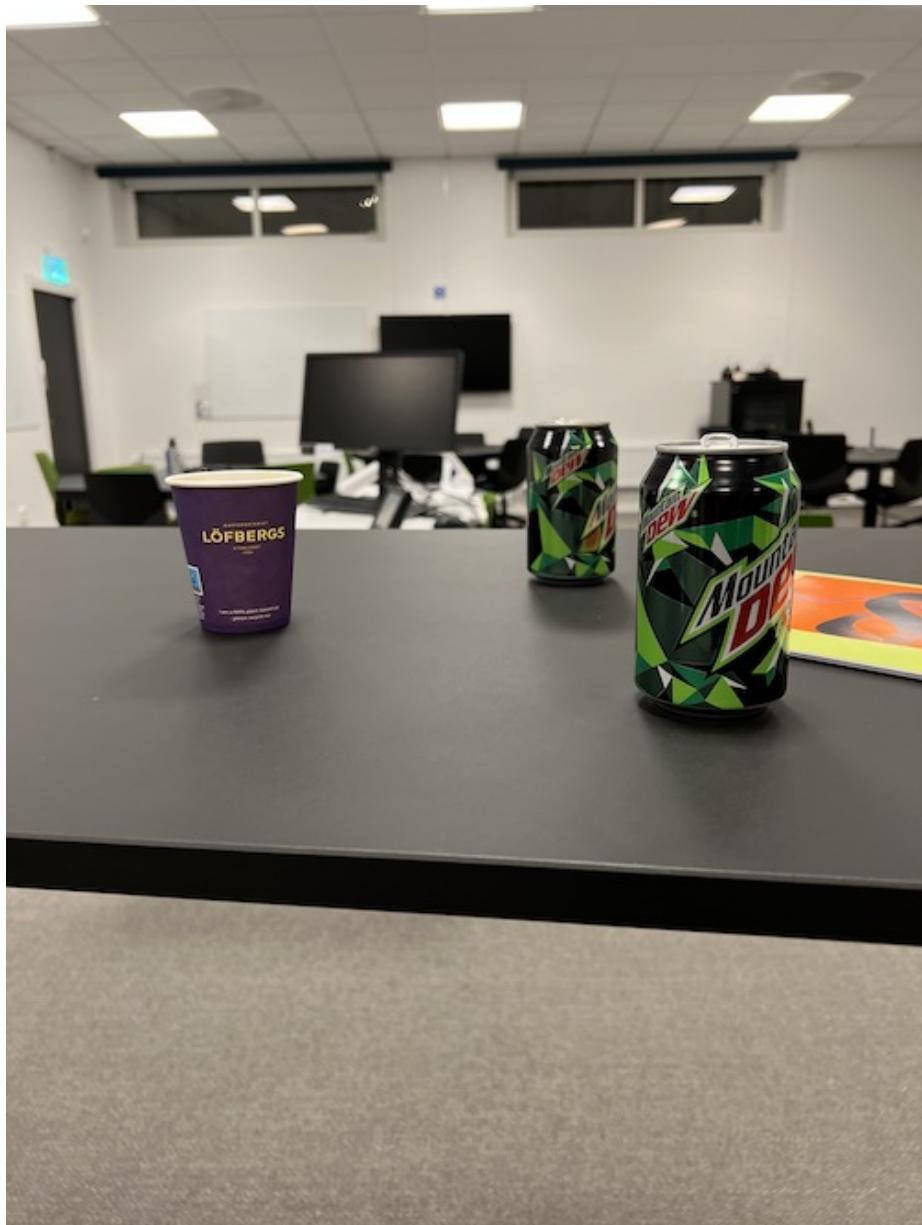


Figure 4: Input for K-means segmentation

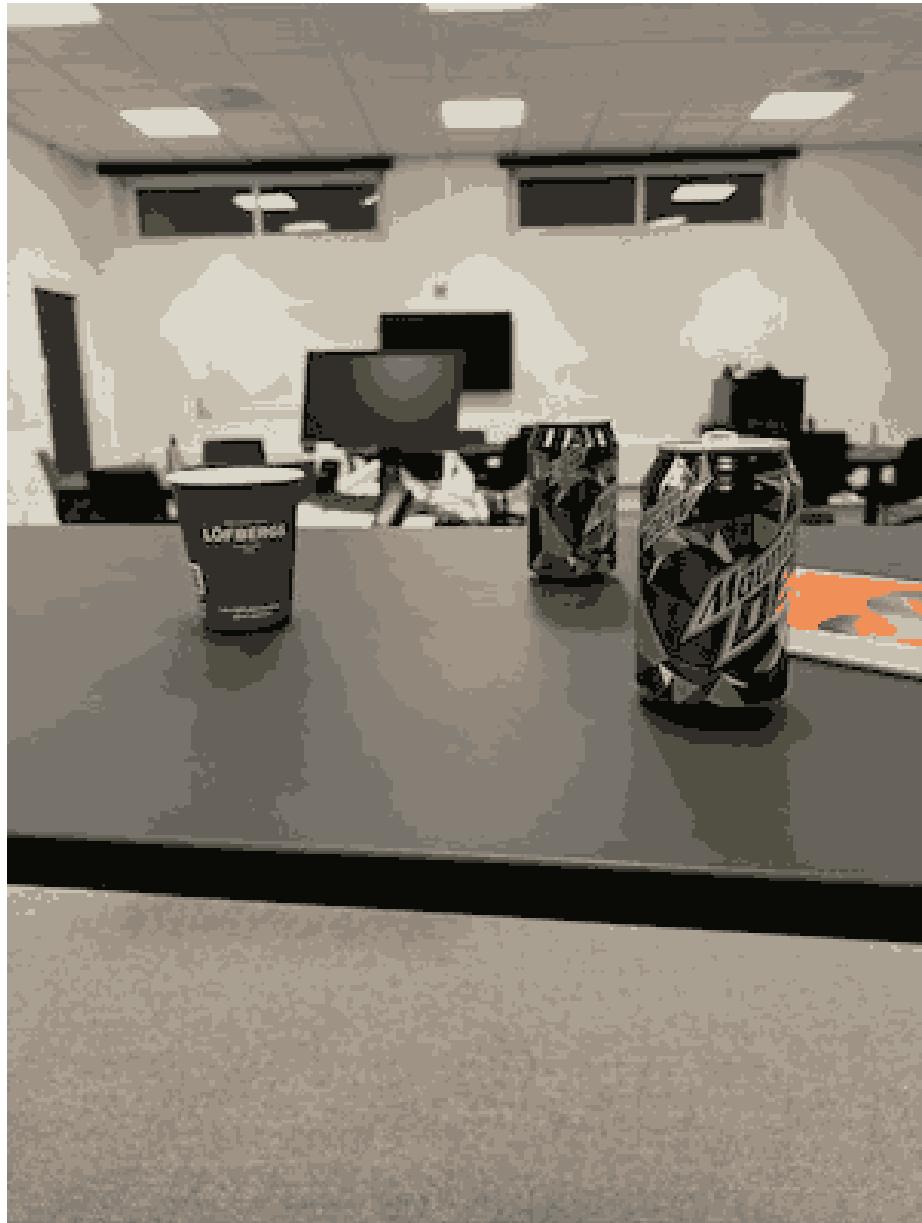


Figure 5: Output from segmentation with 12 clusters and 27 iterations

1.2 Reflections

I do have much to say other than as usual it was cool to see this different methods in action. The hardest part was to figure out the Coarse-to-fine algorithm and

how that improved the optical flow. K-means was just repetition from the data mining course we had. And lastly the coolest thing this week was definitely color segmentation. It was cool to try out different number of clusters to see the different output images.

2 Week 6

2.1 Assignment 6

2.1.1 Theory

The library I choose for Structure from Motion (SFM) was OpenMVG [1]. The library is written in C++ but is run through Python scripts. The Script takes input arguments for both a directory where the input images are located and an output directory. The output consist of a lot of different data. Most importantly it provides a number of .ply files that contain the resulting point cloud. The OpenMVG consists of several submodules for example we have a submodule for running SIFT. To run SFM I used the provided python script, and it goes through the following steps

1. Intrinsic analysis
2. Feature detection
3. Feature Matching
4. Filter Matches
5. Global Reconstruction
6. Coloring

The first step goes through the metadata of the images extracting the camera sensor that took the image and compares that to a pre-defined database. From this it goes through all the images and extracts data and compiles the necessary data to a JSON file to be used by the rest of the operations.

```
1 print ("1. Intrinsic analysis")
2 pIntrinsics = subprocess.Popen( [os.path.join(OPENMVG_SFM_BIN, "
    openMVG_main_SfMInit_ImageListing"), "-i", input_dir, "-o",
    matches_dir, "-d", camera_file_params] )
3 pIntrinsics.wait()
```

Feature detection uses SIFT to extract features from the images and saves the extracted features to files corresponding to the images.

```
1 print ("2. Compute features")
2 pFeatures = subprocess.Popen( [os.path.join(OPENMVG_SFM_BIN, "
    openMVG_main_ComputeFeatures"), "-i", matches_dir+"/sfm_data.
    json", "-o", matches_dir, "-m", "SIFT"] )
3 pFeatures.wait()
```

Next we have feature matching and pair generation. The pair generation generates all the possible pairs of images and puts it in a set. The set is then used to pair images and find matches. The matcher file contains multiple matching algorithm, but it prefers to use Cascade Hashing Matcher.

```

1 print ("3. Compute matching pairs")
2 pPairs = subprocess.Popen( [os.path.join(OPENMVG_SFM_BIN, "
    openMVG_main_PairGenerator"), "-i", matches_dir+"/sfm_data.json"
    ", "-o" , matches_dir + "/pairs.bin" ] )
3 pPairs.wait()
4
5 print ("4. Compute matches")
6 pMatches = subprocess.Popen( [os.path.join(OPENMVG_SFM_BIN, "
    openMVG_main_ComputeMatches"), "-i", matches_dir+"/sfm_data.
    json", "-p", matches_dir+ "/pairs.bin", "-o", matches_dir + "/"
    matches.putative.bin" ] )
7 pMatches.wait()
```

Next it performs geometric filtering on the matches to filter out bad matches.

```

1 print ("5. Filter matches" )
2 pFiltering = subprocess.Popen( [os.path.join(OPENMVG_SFM_BIN, "
    openMVG_main_GeometricFilter"), "-i", matches_dir+"/sfm_data.
    json", "-m", matches_dir+"/matches.putative.bin" , "-g" , "f" ,
    "-o" , matches_dir+"/matches.f.bin" ] )
3 pFiltering.wait()
```

The last two steps comes down to the actual structure from motion and then coloring the 3D points. The method used here is sequential method. In their library I can see there are three methods for 3D reconstruction. Sequential, Sequential v2 and Global. They provide python script for both sequential and global reconstruction, so that is what I have tried. But I will only look into the sequential part due to the superior result it provided. The codebase for this project is massive, so I will only go through the processing steps.

1. Make sure we have a correct initial pair.
2. Create an essential matrix from these pairs.
3. Loop through possible images for reconstruction.
 - (a) Add possible images to reconstruction.
 - (b) Do bundle adjustment until all points are within a certain precision.
 - (c) Remove unstable poses and observations.
4. Ensure there are no more outliers.
5. Done

```

1 print ("6. Do Sequential/Incremental reconstruction")
2 pRecons = subprocess.Popen( [os.path.join(OPENMVG_SFM_BIN, "
    openMVG_main_SfM"), "--sfm_engine", "INCREMENTAL", "--
    input_file", matches_dir+"/sfm_data.json", "--match_dir",
    matches_dir, "--output_dir", reconstruction_dir] )
```

```
3 pRecons.wait()
4
5 print ("7. Colorize Structure")
6 pRecons = subprocess.Popen( [os.path.join(OPENMVG_SFM_BIN, "openMVG_main_ComputeSfM_DataColor"), "-i", reconstruction_dir + "/sfm_data.bin", "-o", os.path.join(reconstruction_dir, "colorized.ply")] )
7 pRecons.wait()
```

2.1.2 Implementation

I used openMVG [1] and ran both Sequential and Global reconstruction. Due to some errors with camera calibration when using my own images I used a dataset provided by them with images of Échillais church [2]. Figure 6 shows an example of images in the dataset.



Figure 6: Example images from the Échillais church dataset[2]

The first reconstruction I did was through the sequential pipeline. The resulting data points are shown in Figure 7.

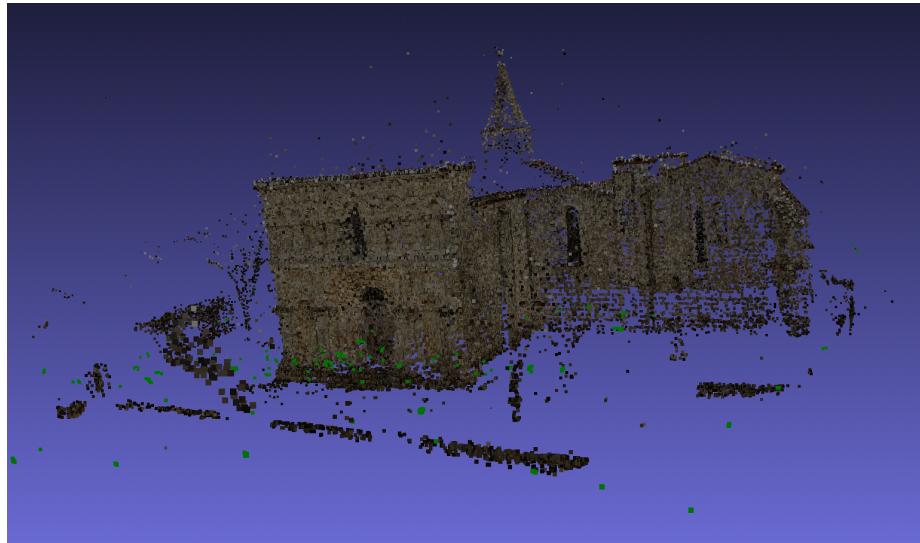


Figure 7: Resulting data points from 3D reconstruction using the sequential SFM pipeline

To compare the different pipelines I also ran the global reconstruction pipeline and the resulting data points can be found in figure 8.

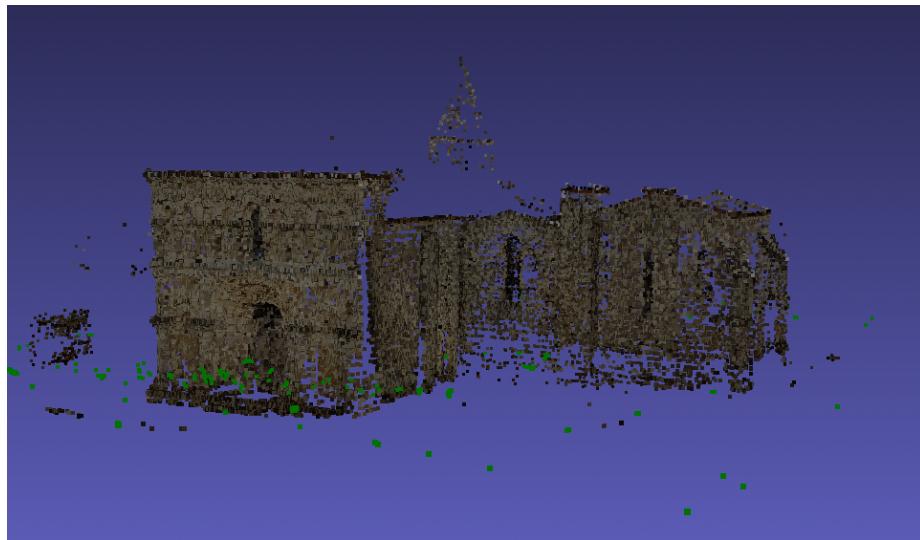


Figure 8: Resulting data points from 3D reconstruction using the sequential SFM pipeline

The global pipeline manages to produce something close to the sequential, but it is not

as detailed as the sequential pipeline. The easiest way to see this is to look at the tower of the church as it is more detailed on the sequential pipeline.

2.2 Reflections

This week I would say was the most interesting week of them all. It was really cool to see how we can use all the previous week's theory into something as cool as Structure from motion. It was hard to find a library that was both easy to understand and could produce a good output. The simplest project I found was a simple python project by Harish Venkataraman [3]. It had problems with the provided dataset and I could not get it working with my own images. I tried another library called openSFM written in both C++ and python, but I could not get it compiling.

So I ended up on OpenMVG written in C++ but can be accessed through python. The problem here is I could not get it working with my own images, but it worked flawlessly with their recommended dataset. I got the error down to the library not being able to derive the focal length for my camera. I tried to provide it myself but with no success. In the end I just used their dataset and I think that is for the better due to the "cool factor" of their data set. From one of the provided datasets I was able to reconstruct a church from France and when loaded I could easily see the structure, and it was really cool.

3 Week 7

3.1 Assignment 7

3.1.1 Triangle

The triangle I choose to do had the following points:

1. (3,2)
2. (2,5)
3. (7,3)

I then chose the inside point (3,2) and the outside point (2,1). I created a python script to calculate my points. The script can be found in appendix D. And the plotted result can be found in figure 9.

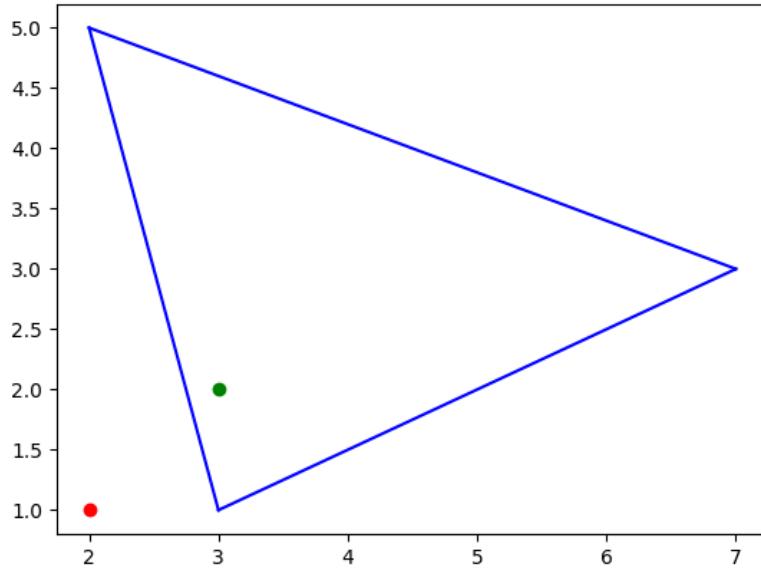


Figure 9: Resulting data points from 3D reconstruction using the sequential SFM pipeline

3.1.2 Phong lighting

I used python to plot the model. The resulting plot can be seen in figure 10 and the code can be found in appendix E. The equation for phong lighting can be found in equation 2.

$$I = k_A I_a + k_D I_D + k_S (\cos(\varphi))^{\alpha} I_L \quad (2)$$

The fixed value used can be found below:

- Light:
 - $I_A = 0.3$
 - $I_D = 0.2$
 - $I_L = 1.2$
- Material:
 - $K_A = 0.1$
 - $K_D = 0.1$

- $K_S = 1.0$
- Variable values
 - $(0 \leq \alpha \leq 20) \vee 3$
 - $(0 \leq \varphi \leq \frac{\pi}{2}) \vee \frac{\pi}{4}$

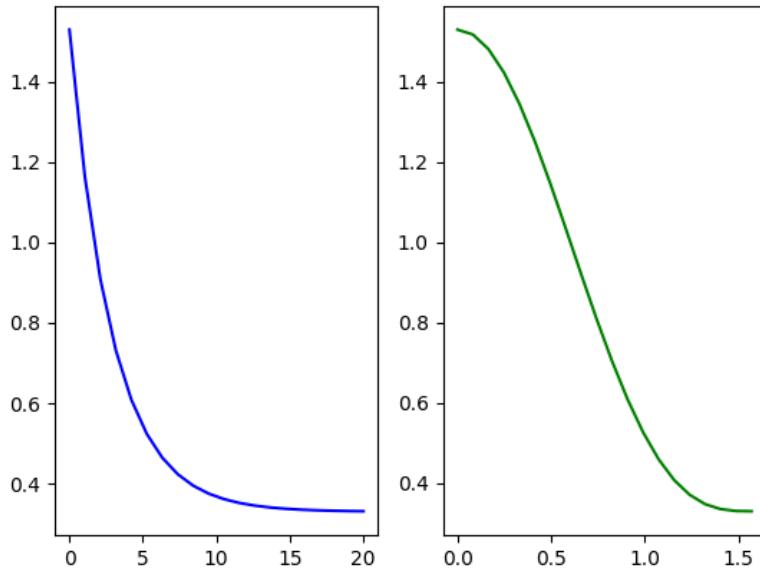


Figure 10: Plot of Phong lighting, blue/left is plot based on α and green/right is plot based on φ

3.1.3 Shading

Python code for creating the 3D image can be found in appendix F. The vertex shader applied can be found in appendix G and the fragment shader can be found in appendix H. I chose to render a sphere and the final render can be found in figure 11.

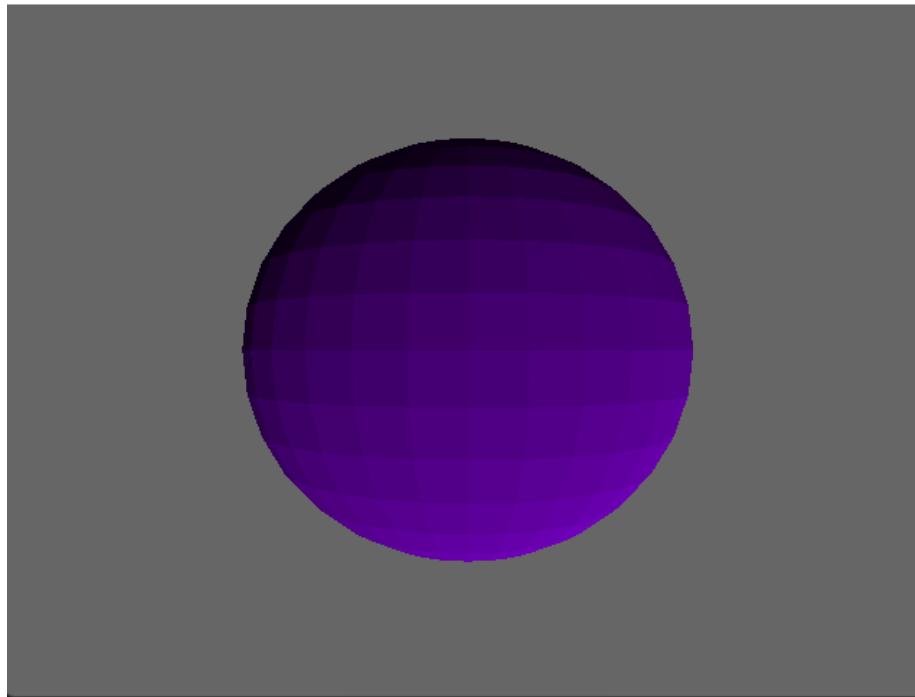


Figure 11: Final rendered 3D image with lambertian shader

3.2 Reflections

As always an interesting week. I did however miss this week's lecture. Thankfully I could easily understand the lecture slides and was able to use them to understand Phong,Lambertian and the formula to check if a pixel is within a triangle. The only real problem I encountered this week was the recommended 3D library for python. The first problem was that OpenGL was deprecated on Mac I solved this by using my Windows desktop pc instead. The next problem was due to the fact that Ratcave was no longer in development it had problems with newer version of Scipy which it depended on. This was solved by downgrading the Scipy version to 1.3.3.

References

- [1] Pierre Moulon, Pascal Monasse, Romuald Perrot, et al. “OpenMVG: Open multiple view geometry”. In: *International Workshop on Reproducible Research in Pattern Recognition*. Springer. 2016, pp. 60–74.
- [2] Romuald Perrot. *Reconstruction Dataset*. 2017. URL: <https://github.com/rperrot/ReconstructionDataSet/tree/master/EchillaisChurch> (visited on 12/18/2021).
- [3] Harish Venkataraman. *Structure From Motion*. 2020. URL: <https://github.com/harish-vnkt/structure-from-motion> (visited on 12/18/2021).

Appendix A Python code for optical flow

```

1 import numpy as np
2 import cv2
3 import os
4
5 frame1 = cv2.imread("im1.jpg")
6 frame2 = cv2.imread("im2.jpg")
7
8 frame1g = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
9 frame2g = cv2.cvtColor(frame2, cv2.COLOR_BGR2GRAY)
10
11 combined = cv2.addWeighted(frame1, 0.3, frame2, 0.5, 0)
12 cv2.imwrite("pyramid/combined.png", combined)
13
14 parameters = dict(maxCorners=100, qualityLevel=0.3, minDistance=7,
15                      blockSize=7)
16
17 p0 = cv2.goodFeaturesToTrack(frame1g, mask=None, **parameters)
18
19 maxLevels = [2,3]
20 maxIterations = [2, 10]
21
22 for level in maxLevels:
23     print("On Max Level: ", level)
24     for iter in maxIterations:
25         OFparams = dict(
26             winSize=(15, 15),
27             maxLevel=level,
28             criteria=(cv2.TERM_CRITERIA_EPS |
29                         cv2.TERM_CRITERIA_COUNT, iter, 0.03),
30         )
31         p1, st, err = cv2.calcOpticalFlowPyrLK(
32             frame1g, frame2g, p0, None, **OFparams)
33         good_new = p1[st == 1]
34         good_prev = p0[st == 1]
35
36         arrows = np.zeros_like(combined)
37
38         for i, (n, p) in enumerate(zip(good_new, good_prev)):
39             nx, ny = n.ravel()
40             px, py = p.ravel()
41
42             nx = int(nx)
43             ny = int(ny)
44             px = int(px)
45             py = int(py)
46
47             arrows = cv2.arrowedLine(arrows, (px, py), (nx, ny), [
48                 255, 0, 255], 2, cv2.LINE_AA,
49                 tipLength=0.4)
50
51             output = cv2.add(combined, arrows)
52             if(not os.path.exists("pyramid/" + str(level))):
53                 os.mkdir("pyramid/" + str(level))
54             outName = "pyramid/" + str(level) + "/" + \
55                     str(level) + "levels-" + str(iter) + "iterations.png"
56             cv2.imwrite(outName, output)

```

Appendix B K-means

```

1  from typing import Tuple
2  from typing import List
3  import matplotlib.pyplot as plt
4  from numpy import sqrt
5  import numpy as np
6
7  def centerToLists(center1, center2):
8      return ([center1[0], center2[0]], [center1[1], center2[0]])
9
10
11 def drawPlot(center1, center2, gc1, gc2):
12     plt.scatter([i[0] for i in gc1], [i[1] for i in gc1], c='blue')
13     plt.scatter([i[0] for i in gc2], [i[1] for i in gc2], c='red')
14     plt.scatter(center1[0], center1[1], c='lightblue')
15     plt.scatter(center2[0], center2[1], c='orange')
16
17 def calcDistance(p1, p2):
18     return sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)
19
20 def categorizePoints(points, center1, center2) -> Tuple[List, List]:
21     gc1 = []
22     gc2 = []
23     for point in points:
24         d1 = calcDistance(point, center1)
25         d2 = calcDistance(point, center2)
26         if d1 < d2:
27             gc1.append(point)
28         else:
29             gc2.append(point)
30     return (gc1, gc2)
31
32 def calculateNewCenter(gc1: List, gc2: List) -> Tuple[Tuple, Tuple]:
33     g1x = np.asarray([i[0] for i in gc1]).sum()/len(gc1)
34     g1y = np.asarray([i[1] for i in gc1]).sum()/len(gc1)
35
36     g2x = np.asarray([i[0] for i in gc2]).sum()/len(gc2)
37     g2y = np.asarray([i[1] for i in gc2]).sum()/len(gc2)
38
39     return ((g1x, g1y), (g2x, g2y))
40
41
42
43 points = [
44     (0, 0.5),
45     (0, 0.75),
46     (1, 1),
47     (1.1, 0.4),
48     (1, 5, 0.75),
49     (2.5, 1),
50     (3, 2),
51     (4, 1.5),
52     (4, 2.5),
53     (5, 2)
54 ]

```

```

55
56 center1 = (1,1.5)
57 center2 = (3,1)
58
59 (gc1, gc2) = categorizePoints(points, center1, center2)
60 drawPlot(center1, center2, gc1, gc2)
61 plt.show()
62 (center1, center2) = calculateNewCenter(gc1, gc2)
63 (gc1, gc2) = categorizePoints(points, center1, center2)
64 drawPlot(center1, center2, gc1, gc2)
65 plt.show()
66 (center1, center2) = calculateNewCenter(gc1, gc2)
67 (gc1, gc2) = categorizePoints(points, center1, center2)
68 drawPlot(center1, center2, gc1, gc2)
69 plt.show()

```

Appendix C Color segmentation

```

1 import numpy as np
2 import cv2
3 from typing import List, Tuple
4 import collections
5 import random
6 import math
7 from PIL import Image
8
9 class Point:
10     position = (0,0)
11     color = [0,0,0]
12     category = [0,0,0]
13
14
15 def convImForShow(im):
16     return cv2.cvtColor(im, cv2.COLOR_RGB2BGR)
17
18 def calcDistance(p1, p2) -> float:
19     x = np.square(p1[0] - p2[0])
20     y = np.square(p1[1] - p2[1])
21     z = np.square(p1[2] - p2[2])
22     sum = x+y+z
23     return math.sqrt(sum)
24
25 def categorizePoints(points: List[Point], centers: List) -> Tuple[
26     List[List], List[Point]]:
27     reee = set()
28     ret = [[] for i in range(len(centers))]
29     dist = np.zeros(len(centers))
30     l = []
31     for p in points:
32         for i, c in enumerate(centers):
33             dist[i] = calcDistance(p.color, c)
34         index = np.argmin(dist)
35         reee.add(index)
36         ret[index].append(p.color)
37         p.category = centers[index]
38         l.append(p)

```

```

38     return (ret, 1)
39
40 def calcCenter(category: List) -> List:
41     x = np.asarray([i[0] for i in category]).sum()/len(category)
42     y = np.asarray([i[1] for i in category]).sum()/len(category)
43     z = np.asarray([i[2] for i in category]).sum()/len(category)
44     return [x,y,z]
45
46
47 def calcNewCenters(categories: List[List]) -> List[List]:
48     newCenters = [[] for i in range(len(categories))]
49     for i, cat in enumerate(categories):
50         newCenters[i] = calcCenter(cat)
51     return newCenters
52
53 def colorPoints(points: List[List], categories: List[List], centers
54 : List[List]) -> List[List]:
55     compare = lambda x, y: collections.Counter(x) == collections.
56     Counter(y)
57     for i, cat in enumerate(categories):
58         for p in cat:
59             indexes = [i for i,x in enumerate(points) if compare(x,
60                         p)]
61             for j in indexes:
62                 points[j] = centers[i]
63     return points
64
65 def imageToPoints(img) -> List[Point]:
66     ret = []
67     for y in range(len(img)):
68         for x in range(len(img[0])):
69             temp = Point()
70             temp.position = (x, y)
71             temp.color = img[y][x]
72             ret.append(temp)
73     return ret
74
75 def pointsToImage(points: List[Point], shape):
76     p = [x.category for x in points]
77     return np.asarray(p, np.uint8).reshape(shape)
78
79 def compareCenters(oldCenter, newCenter):
80     underLimit = 0
81     for i, _ in enumerate(oldCenter):
82         distance = calcDistance(oldCenter[i], newCenter[i])
83         if distance < 1:
84             underLimit += 1
85
86     if underLimit > (len(oldCenter)*0.7):
87         return True
88     return False
89
90
91 im = Image.open("im1.jpg")
92 im = np.asarray(im)
93 number_of_clusters = 4

```

```

92  numberIterations = 50
93
94  shape = im.shape
95  points = imageToPoints(im)
96  randPoints = random.sample(points, numberClusters)
97  centers = [p.color for p in randPoints]
98
99  cat, points = categorizePoints(points, centers)
100
101 iters = 0
102 for i in range(numberIterations):
103     cat, points = categorizePoints(points, centers)
104     newCenters = calcNewCenters(cat)
105     iters = i
106     if compareCenters(centers, newCenters):
107         break
108     centers = newCenters
109
110 print("Did ", iters, " iterations")
111
112 temp = pointsToImage(points, shape)
113 temp = Image.fromarray(temp)
114 temp.show("Iter 0")
115 temp.save("segmented.png")

```

Appendix D Triangle code

```

1 import matplotlib.pyplot as plt
2
3 def subtract(p0, p1):
4     return (p0[0] - p1[0], p0[1], p1[1])
5
6 def dotProduct(p0, p1):
7     return (p0[0] * p1[0]) + (p0[1] * p1[1])
8
9 def isInside(triangle, point):
10    p0, p1, p2 = triangle[0], triangle[1], triangle[2]
11    dxdy = subtract(p1, p0)
12    normal = (dxdy[1], -dxdy[0])
13    line1_check = dotProduct(subtract(point, p0), normal) > 0
14
15    dxdy = subtract(p2, p1)
16    normal = (dxdy[1], -dxdy[0])
17    line2_check = dotProduct(subtract(point, p1), normal) < 0
18
19    dxdy = subtract(p0, p2)
20    normal = (dxdy[1], -dxdy[0])
21    line3_check = dotProduct(subtract(point, p2), normal) > 0
22
23    return line1_check and line2_check and line3_check
24
25 triangle = [
26     (3, 1),
27     (2, 5),
28     (7, 3),
29 ]

```

```

30
31 inside_point = (3,2)
32 outside_point = (2,1)
33
34 points = [inside_point, outside_point]
35
36 for point in points:
37     color = 'go' if isInside(triangle, point) else 'ro'
38     plt.plot([point[0]], [point[1]], color)
39
40 plt.plot([triangle[0][0], triangle[1][0]], [triangle[0][1],
41           triangle[1][1]], 'b')
41 plt.plot([triangle[1][0], triangle[2][0]], [triangle[1][1],
42           triangle[2][1]], 'b')
42 plt.plot([triangle[0][0], triangle[2][0]], [triangle[0][1],
43           triangle[2][1]], 'b')
43 plt.show()

```

Appendix E Phong plot

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4
5 def phong(kA, IA, kD, ID, kS, phi, alpha, IL):
6     return (kA*IA) + (kD+ID) + (kS * (np.cos(phi)**alpha)*IL)
7
8
9 IA = 0.3 # Ambient in the room
10 ID = 0.2 # How much light we add on the whole pov
11 IL = 1.2 # Intesity
12
13 kA = 0.1 # How much ambient light it reflects
14 kD = 0.1 # How the defuse lighting it reflects
15 kS = 1.0 #
16
17 alpha = np.linspace(0.0, 20, num=20)
18 phi = np.linspace(0.0, np.pi/2, num=20)
19
20 ya = [phong(kA, IA, kD, ID, kS, np.pi/4, x, IL) for x in alpha]
21 yp = [phong(kA, IA, kD, ID, kS, x, 3, IL) for x in phi]
22
23 fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2)
24
25 ax1.plot(alpha, ya, 'b')
26 ax2.plot(phi, yp, 'g')
27 plt.show()

```

Appendix F Python 3D render code

```

1 import pyglet
2 import ratcave as rc
3

```

```

4 window = pyglet.window.Window()
5
6 vert = open('vert.GLSL', 'r').read()
7
8 frag = open('frag.GLSL', 'r').read()
9
10 def update(dt):
11     pass
12 pyglet.clock.schedule(update)
13
14 obj_filename = rc.resources.obj_primitives
15 obj_reader = rc.WavefrontReader(obj_filename)
16
17 shader = rc.Shader(vert, frag)
18 print(obj_reader)
19 monkey = obj_reader.get_mesh("Sphere")
20 monkey.position.xyz = 0, 0, -3
21
22 #Uncomment to rotate sphere
23 """ def rotate_meshes(dt):
24     monkey.rotation.y += 45 * dt
25 pyglet.clock.schedule(rotate_meshes) """
26
27 monkey.uniforms['ambientLightStrength'] = 0.15
28 monkey.uniforms['color'] = [0.3, 0.0, 0.5]
29 monkey.uniforms['sunPosition'] = [1.0, -2.0, 3.0]
30
31
32 scene = rc.Scene(meshes=[monkey])
33 @window.event
34 def on_draw():
35     with shader, rc.default_states:
36         scene.draw()
37
38 pyglet.app.run()

```

Appendix G GLSL Vertex shader

```

1 #ifdef GL_ES
2     precision highp float;
3 #endif
4
5 attribute vec4 vertexPosition;
6 attribute vec4 normalPosition;
7 uniform mat4 model_matrix, view_matrix, projection_matrix,
8     normal_matrix;
8 varying vec3 normal;
9
10 void main() {
11     normal = normalize(normal_matrix * (normalPosition)).xyz;
12     gl_Position = projection_matrix * view_matrix * model_matrix *
13     vertexPosition;
13 }

```

Appendix H GLSL Fragment shader

```
1 #ifdef GL_ES
2     precision highp float;
3 #endif
4
5 uniform vec3 color = vec3(0.5, 0.5, 0.5);
6 uniform vec3 sunPosition = vec3(0, 2, 0);
7 uniform float ambientLightStrength = 0.2;
8 uniform float lambertianLightStrength = 0.7;
9 varying vec3 normal;
10
11 void main(void){
12     vec3 lambertian = max(dot(normal, sunPosition), 0.0);
13     vec3 finalIntensity = (lambertian + ambientLightStrength) *
14         color;
15     gl_FragColor = vec4(finalIntensity, 1.0);
16 }
```