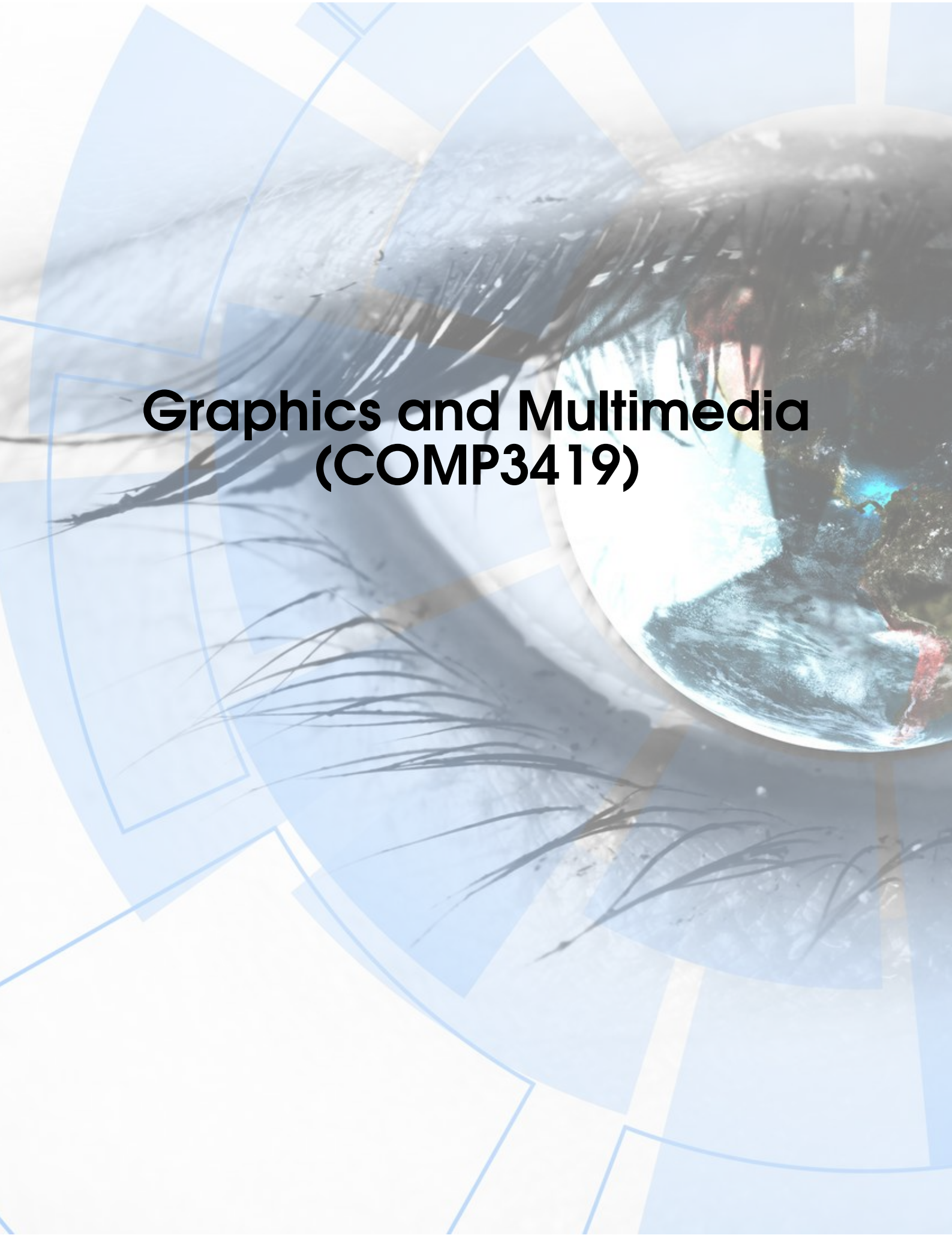


Graphics and Multimedia (COMP3419)

Lab Submission 1 Specification

1 Key Information

- **Deadline**: (1) Submission should be made *before 6am on September 10 (Monday in Week 7)*. (2) Demonstration should be made *in the lab session on Week 7*.

- **Submission**: All source code should be submitted as a zip file via Canvas and counts for 5% of your total assessment. Do not submit your output video file. Please provide a README file within the zip file, including the instructions to run your code.

- **Demonstration**: You are allowed to use any video to demonstrate your code. Please have your corresponding output video ready for the demonstration. You are required to provide a working live demo to show your work to your tutor during the lab session on Week 7.

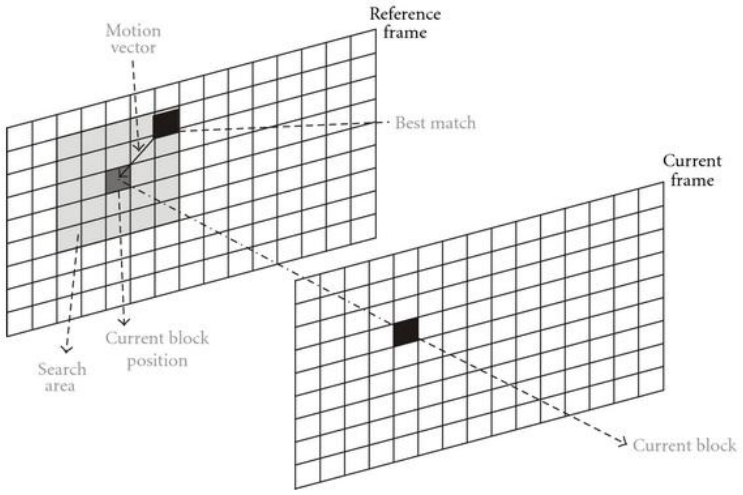
2 Demonstration Rules

- Make sure you can demonstrate your program in the reasonable time. You can either use your laptop or the computer at the lab. Please test your program before coming to the demonstration lab.

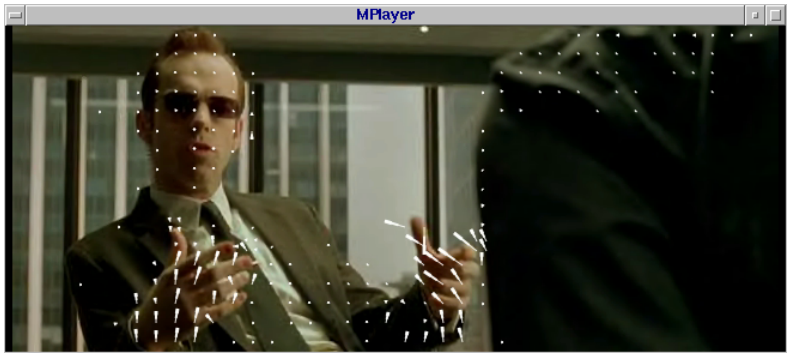
- You will be required to explain the implementation of your code to the tutor.

- If you can not finish all the requirements, you should provide a live demo for the workable parts.

3 Motion estimation with Macroblock matching (Lab Submission 1)



(a) Illustration of the block matching



(b) Example extracted motion vectors. Please note that in your lab submission, you do not need to draw arrows. Highlight the pixels with high magnitude would be sufficient.

Figure 6.1: The illustration of block matching algorithm and the extracted optical flows.

This task is to perform the motion estimation with macroblock matching. The basic premise of motion estimation is that in most cases, consecutive video frames will be similar except for changes induced by objects moving within the frames. The basic idea of motion estimation is to define grids of block regions on two adjacent frames and find the displacement vector (a 2D Cartesian vector in 2D videos) between the matched blocks. To describe the meta-algorithm step by step:

1. Iterate the video frames F_i of size $f_x \times f_y$; Define a grid block size $K \times K$, K is preferred to be odd to make it easier to determine the central coordinate of each grid block. Each frame F_i results in $f_x f_y / K^2$ grid blocks overall.
2. For each grid block B_i at (x, y) in frame F_i, search for the grid block B'_{i+1} at (x', y') in F_{i+1}

with the minimum sum squared distance (SSD) between B_i and B'_{i+1}. SSD can be computed as

$$SSD(B_i, B'_{i+1}) = \sqrt{\sum_{bx}\sum_{by}\sum_{bc}(B_i(bx,by,bc) - B_{i+1}(bx,by,bc))^2} \tag{6.1}$$

The displacement from B_i to B'_{i+1} can be represented as $(x'-x, y'-y)$, a 2-D vector. To speed up, you can search the neighbour blocks only within a certain radius.

3. Save the displacement vectors of frame F_i in a 3D matrix of size $f_x/K \times f_y/K \times 2$ or two 2D matrices with size $f_x/K \times f_y/K$ each.
4. Visualise the displacement fields with a 2D image with the same size as frame F_i. The same magnitude value is assigned to all the pixels within the region covered by a single grid block. Alternatively you can draw arrows to represent the extracted displacement fields.
5. Visualise the boundary of the object with large displacement. Hint : the dilation or erosion might be helpful.
6. Repeat step 1-5 for all frames

Drawing arrow is not required. If you really want to draw arrows, the following code can be your starting point. [Processing]

```
void arrowdraw(int x1, int y1, int x2, int y2) {
  line(x1, y1, x2, y2);
  pushMatrix();
  translate(x2, y2);
  float a = atan2(x1-x2, y2-y1);
  rotate(a);
  line(0, 0, -10, -10);
  line(0, 0, 10, -10);
  popMatrix();
}
```

Drawing arrow is not required. If you really want to draw arrows, the following code can be your starting point. [Python]

```
import math
import cv2

def arrowdraw(img, x1, y1, x2, y2):
    radians = math.atan2(x1-x2, y2-y1)
    x11 = 0
    y11 = 0
    x12 = -10
    y12 = -10

    u11 = 0
    v11 = 0
    u12 = 10
    v12 = -10
```

```
        x11_ = x11*math.cos(radians) - y11*math.sin(radians) + x2
        y11_ = x11*math.sin(radians) + y11*math.cos(radians) + y2

        x12_ = x12 * math.cos(radians) - y12 * math.sin(radians) + x2
        y12_ = x12 * math.sin(radians) + y12 * math.cos(radians) + y2

        u11_ = u11 * math.cos(radians) - v11 * math.sin(radians) + x2
        v11_ = u11 * math.sin(radians) + v11 * math.cos(radians) + y2

        u12_ = u12 * math.cos(radians) - v12 * math.sin(radians) + x2
        v12_ = u12 * math.sin(radians) + v12 * math.cos(radians) + y2
        img = cv2.line(img, (x1, y1), (x2, y2), (255, 0, 0), 5)
        img = cv2.line(img, (int(x11_), int(y11_)), (int(x12_), int(y12_)),
        (255, 0, 0), 5)
        img = cv2.line(img, (int(u11_), int(v11_)), (int(u12_), int(v12_)),
        (255, 0, 0), 5)

        return img
```

For more details of in-depth understanding of the video estimation algorithm, you may refer to `http://web.stanford.edu/class/ee398a/handouts/lectures/EE398a_MotionEstimation_2012.pdf`

For a faster solution of estimating optical flow with Lucas-Kanade method please see `http://research.ijcaonline.org/volume61/number10/pxc3884611.pdf`
and a Matlab solution at `http://www.cs.ucf.edu/~gvaca/REU2013/p4_opticalFlow.pdf`

Please note for the lab submission you are not required to use the Lucas-Kanade method.