Introduction to Video Processing

HW2

Hong Shiang Lin

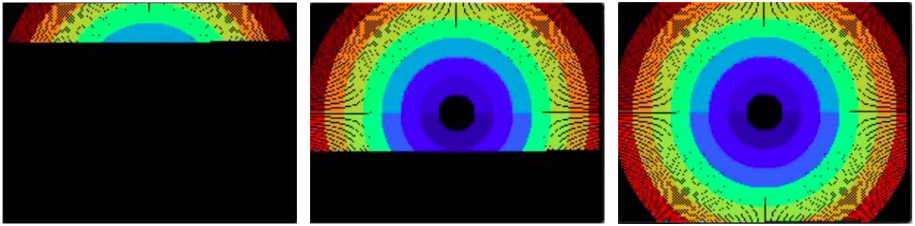
National Taipei University

hongshianglin@gm.ntpu.edu.tw

February 27, 2025

HW2: Raster Scanning

Upon completing the homework, you can generate video images to visualize the raster scanning process.



You should complete the following tasks:

- Complete the provided codes.
- Conduct experiments and analyze the results.
- Prove a property.
- Provide a HW report (see report guidelines).

Code Completion

(Tracing on *revised* **video_formation_simulator.py**)

```
import cv2
from rotated_rainbow import *
from camera import *
from display import *

# Generate lights from a rotated rainbow
scene = RotatedRainbow()
elapsed_time_ms = 0 # in milli seconds
delta_time_ms_light = 10 # in milli seconds
```

- Color converter and human visual system are migrated out in this version.

```python
# Video simulation
fov = 60 # vertical fov in degrees
frames_per_second = 30 #
image_width, image_height = 200, 150 # in pixels
fourcc=cv2.VideoWriter_fourcc(*'mp4v')
scan_video = cv2.VideoWriter('scanning.mp4', fourcc, frames_per_second,
      (image_width, image_height))
camera = Camera(fov, image_width, image_height, frames_per_second)
display = Display(image_width, image_height)
```

- Output image size is reduced in this version.
- The *scan_video* is used for recording the scanning process.

```
# Timing setting
frame = 0
delta_time_ms_frame = 1000 / frames_per_second

# Illustrate the raster scanning processing by snapshoting the image on
    line rate
# Calculate the observation counter duration by setting it as the line
    interval divided by the number of rasters per line
# (i.e., the image width).
elapsed_time_ms_observ = 0
line_interval = camera.get_line_interval()
delta_time_ms_observ = line_interval / image_width
frame_observ = 0
```

- *delta_time_ms_frame*: Frame interval of the camera.
- *delta_time_ms_observe*: Frame interval of the *scan_video*.

```
# TODO #1-1: Create a file handler for the output scanning.txt file for
    recording the scanning indices
# Check the recording format in slides.
```

- You should dump the observed scanning indices for each camera frame into a file named *scanning.txt*
- Scanning indices of different camera frame are separated by line.

(For example, assume two frames of the camera is scanned and its image width is 3×3)

The format inside the *scanning.txt*

1 2 3 5 7 8 (comment: observed scanning indices of the first frame)
1 3 4 5 6 7 (comment: observed scanning indices of the second frame)

```python
# Constant for millisecond to second conversion
ms2sec = 0.001
while frame < 3:
    lights = scene.generate_lights(elapsed_time_ms * ms2sec)
    elapsed_time_ms += delta_time_ms_light

    camera.expose(lights)
```

- This version uses *camera.expose(lights)* to compute the frequency response of lights.

```
if elapsed_time_ms >= frame * delta_time_ms_frame:
    print('Start scanning for frame:', frame)
    elapsed_time_ms_observ = 0
    frame_observ = 0
    scan_idx_prev = 0
    camera.restart_scanning_frame()
    while not camera.frame_completed():
        camera.raster_scan(delta_time_ms_observ)

        # See next slide for observed scanning image generation

        elapsed_time_ms_observ += delta_time_ms_observ
```

- If the *elapsed_time_ms* exceeds the time required to render the latest frame, start scanning the latest frame.
- While the frame is not completely scanned, we keep calling *camera.raster_scan()* to continue the scanning.
- *elapsed_time_ms_observ* is used for determining when to render the observed scanning process into *scan_video*.

```
        if (elapsed_time_ms_observ >= frame_observ * (0.1 * line_
            interval)):
            attenuation_image = camera.output_attenuation_image_
                scanned(scan_idx_prev, camera.get_scan_index())
            gc = camera.get_gamma()

            # See next slide for updating latest scanned segment
                only for quick viz. freshing

            print('Observe scanned lines:', camera.scanned_lines, '
                scan index:', camera.get_scan_index())
            display_image = display.output_buffer()
            scan_video.write(display_image)

            # TODO #1-2: Record current scan index into the output
                .txt file.

            frame_observ += 1
            scan_idx_prev = camera.get_scan_index()
```

- Get the attenuation signal from the camera.
- After updating the latest scanned segment, the signal is transmitted to the display.
- The display image is written into the *scan_video*.
- TODO: Record the newest scan index of current observation.

```
# Update latest scanned segment only for quick viz.
    freshing
for idx in range(scan_idx_prev, camera.get_scan_index()
    ):
    y = int(idx / image_width)
    x = idx - y * image_width
    r, g, b = attenuation_image[y, x, :]
    display_color = [r, g, b]
    # TODO #2: Record brightness after correction (
        already done it on hw1)

    rd, gd, bd = display_color[0], display_color[1],
        display_color[2]
    display.write_buffer([x, y], rd, gd, bd)
```

- Use your implemented gamma correction in HW1.
- Only refreshing latest scanned segment for quick viz.

```
        # Clear the display buffer to prevent the retention of content
            from the previous frame.
        display.clear_buffer()
        frame += 1

    print('Elapsed time in ms:', elapsed_time_ms)

# TODO #1-3: Close the output scanning.txt file.
scan_video.release()
```

- Clear display and the video
- Remember to close the txt file

(Tracing on to *revised* **camera.py**)

```
class Camera():
    def __init__(self, fov, image_width, image_height, frames_per_
        second):
        # Projection matrix
        F = fov * np.pi / 180
        w, h = image_width, image_height
        f = h / (2 * (np.tan(F / 2))) #Focal length
        self.P = np.array([[f, 0, w / 2],
                           [0, f, h / 2],
                           [0, 0, 1]])

        # Exposed image: # stored exposure information, value: 0 - 1
        self.exposed_image = np.zeros([h, w, 3])

        # Scanned image: raster scanning on the latent image
        self.scanned_image = np.zeros([h, w, 3])

        # Output attenuation image: raster scanning on the latent image
        self.attenuation_image = np.zeros([h, w, 3])
```

- Create images for exposed, scanned, and attenuation signals.

```
# Compute line rate and intervals according to fps and fsy
# Line rate (lines per second) = fps * self.h (lines per frame)
self.compute_line_rate()

# Line interval = 1 / (line rate)
self.compute_line_interval_ms()

# Specify horizontal retrace time and vertical retrace time
# Horizontal retrace time should not larger than line interval
self.horizontal_retrace_time = 0.01 * self.line_interval

# Set vertical retrace time in ms
self.vertical_retrace_time = 0.05
```

- Set raster scan parameters.

```
        # Compute actual line scanning time and scanned lines in a
            frame
        # Line scanning time = (line interval) - (horizonal retrace
            time)
        self.compute_line_scanning_time()

        # Compute scanning time per pixel
        self.compute_pixel_interval()

        # Scanned raster index
        self.scan_index = 0
        self.elapsed_time_ms_scan = 0
        self.scanned_lines = 0
        self.completed = False
```

- Compute scanning times.
- Recorded scanning results.

```python
def expose(self, lights):
    # Assume the exposure is simultaneously on an area and the
        exposure time is 0
    # Here we use previous sharpe model for HVS as the camera
        sensor exposure model
    hvs = HumanVisualSystem()
    for light in lights:
        r = g = b = 0 # Initialize r, g, b to zeros
        y = u = v = 0 # Initialize y, u, v to zeros

        # TODO #3: Get receptor responses (already done it on hw1)

        # Capture lights in camera
        x = self.project_to_image_position(light.X)
        self.write_exposed_image(x, r, g, b)
```

- Use human visual system function to represent the camera spectral absorption function.
- Use your implemented receptor responses on hw1.

```
    def raster_scan(self, delta_time_ms):
        # Line interval = self.pixel_interval * self.w  + self.
            horizontal_retrace_time
        t0 = self.elapsed_time_ms_scan
        t1 = self.elapsed_time_ms_scan + delta_time_ms
        sec2ms = 1000
        T_frame_ms = sec2ms / self.fps
        l = self.scanned_lines

        if t1 <= T_frame_ms - self.vertical_retrace_time: # Before
            vertical retracing
            # See next slide for line scanning.
           # Update elapsed time
           self.elapsed_time_ms_scan = t1
        elif t1 <= T_frame_ms: # During vertical retracing
            # Update elapsed time
            self.elapsed_time_ms_scan = t1
        else: # Finish vertical retracing
            self.completed = True
            self.elapsed_time_ms_scan = 0
```

- Before vertical retracing, the image is under line scanning.
- During vertical tracing, there are no new lines.
- The frame scanning is completed when vertical retracing is finished.

```
        # Determine the elapsed time after horizontally retracing on
            current line
        t_line = t1 - l * self.line_interval
        if t_line <= self.line_scanning_time:
            # TODO #4: Finish line scanning by implementing the
                following procedure to replace ’pass’ command:
            # 1. Determine scanning segment based on previous raster
                 index, t_line and raster rate
            # 2. Employ raster_scan_one_pixel to execute scanning
                for each individual pixel
            pass
        elif t_line <= self.line_interval: # Re-tracingg
            pass
        else:
            # TODO #5: Finish horizontal re-tracing to replace ’pass
                ’ command
            # 1. Update scanned_lines and scan_index
            pass
```

- During actual line scanning time, update the scanning by determining new scanning segment.
- Once exceed the line scanning time, the scanning enters a re-tracing stage.
- When finish retracing, the line index and scanning index should be updated

```python
def raster_scan_one_pixel(self):
    # Recover image coordinate from raster index.
    # Let (x[0], x[1]) denote the image coordinate, then scan_index
        = x[1] * w + x[0].
    x = [0, 0]
    x[1] = int(self.scan_index / self.w)
    x[0] = self.scan_index - x[1] * self.w

    # Record current target raster, after scanning it, the target
       raster is moved.
    r, g, b = self.exposed_image[x[1], x[0], :]
    self.write_scanned_image(x, r, g, b)

    # Increase scan index
    self.scan_index += 1
```

- The scan index is composed of 2D image coordinates.
- The scanned image is updated here.

Experiments

(Present additional experiments to showcase your insights.)

- Examine and analyze variations in results by adjusting the fps and image height of the camera.
- Examine and analyze variations in results by adjusting the retrace times of the camera.
- Share any additional noteworthy insights.
- Write them into reports.

Proofs

Prove that the actual scanning lines per frame is given by

$$f'_{s,y} = f_{s,y} - \frac{T_v}{T_l}.$$

Use LaTeX commands or other tools for generating mathematical symbols.

(Demo for Latex Commands)

Commands

```
$f'_{s, y} = f_{s, y} - \frac{T_v}{T_l}$
```

Resulting Symbols

$f'_{s,y} = f_{s,y} - \frac{T_v}{T_l}$

Numerous free online LaTeX math tools are available.

Report: Guidelines

- Describe your implementation. (Do NOT paste code screen shot figures)
- Describe your understanding of theories.
- Describe your experiment settings and results.
- Describe your proofs.
- Write your division of labor.
  - If no report of division of labor, points will be deducted.
  - If you only list the division of labor in percentages, those with a lower percentage will be deducted points accordingly.
  - Every assignment must involve all members (either by distributing TODOs or taking on different roles).
  - No assignment should be completed by only one person. If this happens, those who did not participate will receive a zero.
  - If you only state that you were responsible for debugging but do not explain what issues were resolved, points will be deducted.
  - If you only state that you were responsible for the report, but the report content is minimal, points will also be deducted.
- Only .pdf file format is allowed.

(Continue: Grading criteria of the report)

- Experiment richness.
- Experiment depth.
- Clarity of insights.

Scoring

- Code completion (50%)
    - TODO points in **video_formation_simulato.py** and **camera.py** to generate the necessary videos.
- Experiments (20%)
- Proof (10%)
- Report (20%)

Submission Guidelines

- Put all the codes into a folder *codes*.
- Put all the basic videos (without modifying default setting) into a folder *videos*.
- Put all the experiemented videos (with additional experiment setting) into a folder *supplementary-material*.
- Put **videos**, **codes**, **supplementary-material** (optional) and report.pdf into a folder **hw2**.
- Compress **hw2** and specify the file format as hw2.zip.
- Upload hw2.zip onto Digital Platform 3.
- Deadline: 12:00 a.m. on March 13th.

Origin of The Latex Template

The latex template is downloaded from:
https://www.LaTeXTemplates.com

The Author: Vel (vel@latextemplates.com)