CS 10:
Problem solving via Object Oriented Programming

Winter 2017

Tim Pierson
260 (255) Sudikoff

Agenda

1. Webcam processing
2. Color tracking
3. Frame differencing
4. Recording a loop
5. Background subtraction

Previously manipulated a single image, video is just multiple images over time

n images form a video

0 1 … … n-2 n-1

We can individually process each image (sometimes called a frame)
We just have to be done before the next image arrives!

Image: appocoda.com

Video brings together many concepts we've previously seen with images

Webcam.java

- Wrapping OpenCV for Java
- Two set up parameters:
  - *scale* -- down-size the image (useful for intensive processing)
  - *mirror* -- flip the image left-right (feels more natural)
- Constructor prints out the native camera size so you can decide what a good scale factor is to yield a sufficiently small image
- You don't really need to be familiar with the Webcam code. It's built off DrawingGUI, so the same methods apply for handling events and drawing
- One new method is *processImage()*, which we define in Webcam subclasses to do something with each separate image as it comes off the webcam
- Image itself is stored in an instance variable called *"image"*, to which processImage has access, and which it can modify

Subclassing Webcam.java allows us to easily process individual image frames

WebcamProcessing.java

- Run – screen colored blue
- Subclass of Webcam
- Each frame grabbed by the camera in Webcam calls *processImage()* in subclass, with image stored in BufferedImage *image* variable
- Since result of camera grab is a BufferedImage, can apply any of our previous image processing methods to it
- Here we have a our previous image processor scale the color of the image and display it in *processImage()*
- Notice we did not have to make changes to the image processor

We can also override *draw()* to change how things are rendered

WebcamRendering.java

- Run
  - Press “m” for mosiac
  - Press “p” for pointillism
  - Press “i” for standard image
- *style* sets how image will be rendered, set on key press
  - mosaic
  - pointillism
- *draw()* overridden to choose how image rendered based on *style* variable
- *mosaic()*
  - *Pick up color at x,y*
  - Draw rectangle of size pixel filled with color
  - Draw black border

Agenda

1. Webcam processing
2. Color tracking
3. Frame differencing
4. Recording a loop
5. Background subtraction

We can track a point as it moves, essentially using the point as a mouse

WebcamColorTracking.java

- Run, press mouse on color to track, move object with color
- On mouse press, save color underneath mouse location in *trackColor*
- *track()* finds the pixel closest in color to the color saved; returns a Point object (has both x and y values)
- *draw()* then draws an oval around the point returned by track, provided a color has been selected
- Assumes the object moves smoothly and doesn't change color too much (e.g., lighting, orientation changes)
- Not too sophisticated, but generally works

Agenda

1. Webcam processing
2. Color tracking
3. Frame differencing
4. Recording a loop
5. Background subtraction

To detect movement, we can subtract one frame from the next

WebcamDiff.java

- Run, move hand in front of camera
- *processImage()* each frame, make a copy of the current image to save what came in from the camera, store in BufferedImage *curr*
- Then simply subtract RGB values for corresponding pixels from the current frame and the previous frame
- Compute the absolute value of the difference
  - Can't have negative color values
  - Direction of the difference doesn't matter anyway for this purpose
- Write difference to current image with *setRGB()*
- Set *prev* frame to *curr*

Agenda

1. Webcam processing
2. Color tracking
3. Frame differencing
4. Recording a loop
5. Background subtraction

If we can keep track of one frame, we can keep track of many frames and play back

WebcamLoop.java

- Run, wave hand, click mouse to play back video in reverse order
- Add instance variables to keep track of:
  - Recording vs. playback state
  - Frame buffer called *"frames"*
  - Current frame number
- *processImage()* – if recording, add this frame to frame buffer
- *draw()* – if recording, just draw image, else play "frame" frame in buffer
- *handleMousePress()*
  - *Toggle recording state*
  - *Erase frame buffer or set current frame number (depending on recording state)*

Agenda

1. Webcam processing
2. Color tracking
3. Frame differencing
4. Recording a loop
5. Background subtraction

We can now add concepts together to do "green screen" like background subtraction

WebcamBg.java

- Run, get out of frame, click mouse, come back in frame in
- Constructor saves image we want as background in "*scenery*" instance variable (e.g., Baker tower)
- *handleMousePress()* sets background image as frame coming from camera (this is what we want to subtract)
- *processImage()*
  - Look at each pixel in current camera image
  - If pixel color close to background image pixel in same location, then replace image pixel with scenery pixel at that location
- Works best under controlled lighting and if your webcam isn't trying to be too fancy itself by adjusting brightness, etc.
- Scale the webcam image down to the size of the background scenery (the setup scale, stored in a static final variable in DrawGUI).