CS 10:
Problem solving via Object Oriented Programming

Video Processing

Agenda

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

Previously we manipulated a single image, video is just a stream of images over time

***n* images form a video**

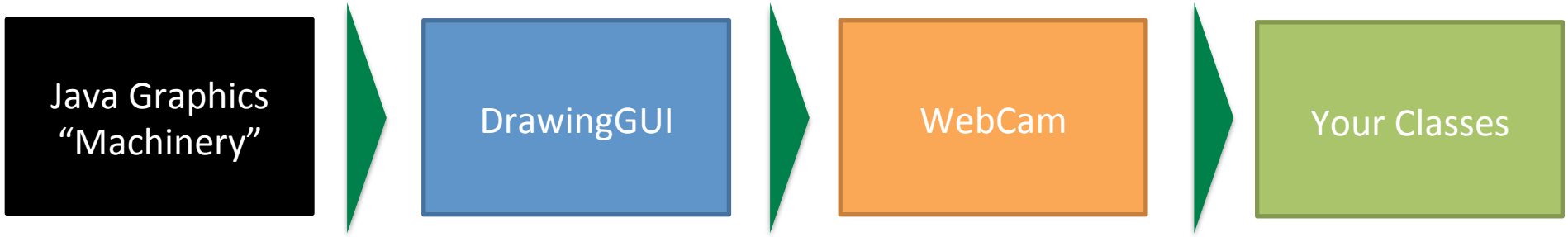
0 1 … … n-2 n-1

- Can individually process each image (called a frame in video)
- Just have to be done processing before the next image arrives!
- Can do some tricks if we realize most of the image is the same from frame to frame

Image: appocoda.com

I've provided a WebCam class to try to make handling video easier

Conceptual



- Java provides GUI code
- Somewhat complicated
- Learning the specifics of Java's GUI "machinery" not really the point of this course

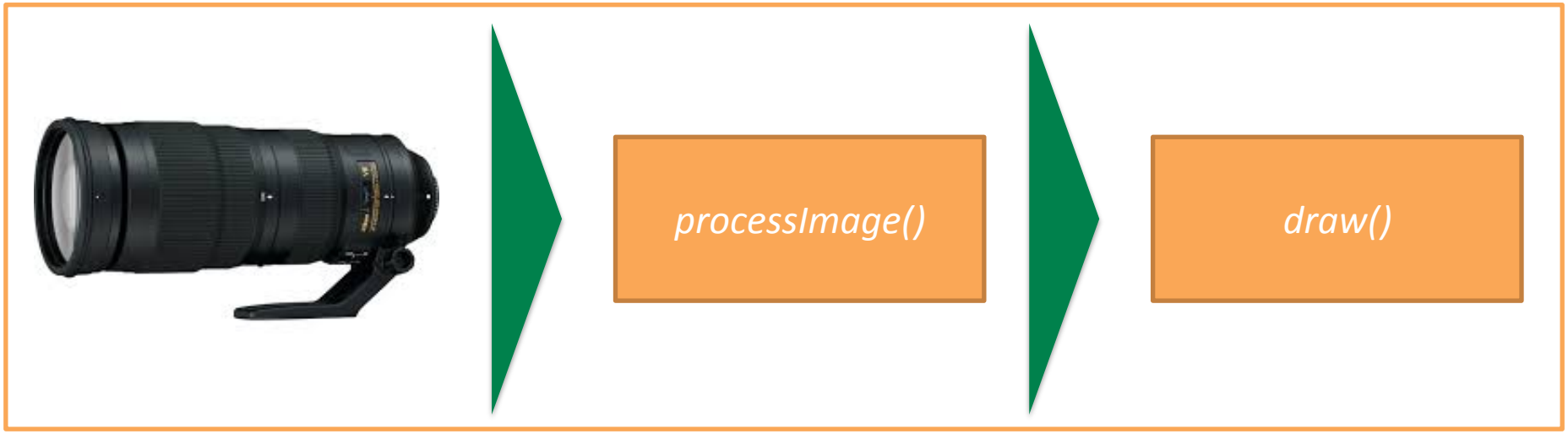
- Wrapper that inherits from Java's *JFrame* machinery
- Sets up GUI
- Provides methods we override:
  - *handleTimer()*
  - *handleMousePress()*
  - *handleKeyPress()*
  - *draw()*

- Wrapper that inherits from *DrawingGUI*
- Sets up camera
- Provides methods we override:
  - *processImage()* to alter image appearance before display
  - Automatically calls *draw()* after *processImage()*

- Inherit from *WebCam*
- Override *processImage()* to handle frames as captured
- Use *draw()* to display image
- Still get *DrawingGUI*'s methods

WebCam pipeline goes from camera to *processImage()* to *draw()*

WebCam.java



Camera

- Camera captures image every 100 ms
- Updates WebCam *image* instance variable at each new image capture

***processImage()* method**

- Empty in WebCam.java
- Override to change *image* (e.g., dim(), brighten(), etc)
- Changes happen before image is displayed
- Make sure to update WebCam *image* variable after applying changes

draw() method

- By default displays image stored in instance variable *image* on screen
- *image* may have been altered in *processImage()*
- *draw()* an be overridden to give different functionality

Last image from camera is stored in instance variable *image*

WebCam.java

```
24 public class Webcam extends DrawingGUI {
25     protected boolean mac = true;                    // TODO: set to
26     private static final double scale = 0.5;         // to downsize the i
27     private static final boolean mirror = true;      // make true in
28
29     protected BufferedImage image;                   // image grabbed
30
31     private Grabby grabby;                           // handles webca
32     private FrameGrabber grabber;                    // JavaCV
33
34     public Webcam() {
35         super("Webcam");
36
37         try {
38             if (mac) grabber = new OpenCVFrameGrabber(0); // this se
39             else grabber = FrameGrabber.createDefault(0);  // this s
```

Set to true if Mac user, false otherwise

Downsize sample (for faster processing) Here we make image half size

Mirror swaps left and right, makes things "look right"

- Last camera image stored here
- Updated every 100 ms as new images captured

processImage() allows image changes, *draw()* displays *image* unless overridden

WebCam.java

- **processImage() called by WebCam each time a new frame arrives**
- **By default it makes no changes to *image***
- **We can override it to apply our changes**

```
67      public void processImage() {
68          // Default: nothing
69      }
70
71      /**
72       * DrawingGUI method, here showing the current image.
73       */
74      @Override
75      public void draw(Graphics g) {
76          g.drawImage(image, 0, 0, null);
77      }
```

- **Unless overridden, *draw()* will display whatever is stored in instance variable *image* (see last slide for declaration)**
- ***draw()* is called automatically by WebCam after each call to *processImage()***

Demo: WebCamProcessing.java

Notes:
Gives image a blue hue to camera image

WebcamProcessing.java allows us to easily process individual image frames

WebCamProcessing.java

```
11 public class WebcamProcessing extends Webcam {
12     private ImageProcessor proc;// handles image processing, from last clas
13
14     public WebcamProcessing() {
15         proc = new ImageProcessor(null);
16     }
17
18     /**
19      * Use proc to change colors (try something more fun yourself)
20      */
21     @Override
22     public void processImage() {
23         proc.setImage(image); //give current image to ImageProcessor proc
24         proc.scaleColor(0.5, 0.5, 1.5); //change color, emphasize blue, de-
25         image = proc.getImage(); //set image from proc so WebCamProcessing
26     }
27
28     public static void main(String[] args) {
29         SwingUtilities.invokeLater(new Runnable() {
30             public void run() {
31                 new WebcamProcessing();
32             }
33         });
34     }
35 }
```

Inherit from WebCam to get it's methods (which also has DrawingGUI methods)

Add *ImageProcessor* from last lecture to be able to manipulate images

Notice that we don't have to re-write all that code! This is all the new code there is

- Override *processImage()* from WebCam
- Called when image captured by camera
- Last camera frame stored in *image*

- Set *image* in *ImageProcessor proc*
- Use *proc* to scale color
- Here we emphasize blue Color

Don't forget to set WebCam instance variable *image* to the processed image

What will happen if you forget?

Image on screen won't change if you don't update *image*

WebCam *draw()* method (called next) displays what is in *image* variable

Demo: WebcamRendering.java

Notes:

Shows camera image in mosaic or pointillism view

Press:

- ‘m’ for mosaic view
- ‘p’ for pointillism view
- ‘+’ to increase pixel size
- ‘-’ to decrease pixel size

WebcamRendering.java: Display ovals or rectangles instead of captured image itself

WebCamRendering.java

```
55      @Override
56      public void draw(Graphics g) {
57          if (image!=null) { //*** notice following if statements
58              if (style=='m') mosaic(g);
59              else if (style=='p') pointillism(g);
60              else super.draw(g);
61          }
62      }
63
64      /*
65       * DrawingGUI method, here just remembering the style for re
66       */
67      @Override
68      public void handleKeyPress(char key) {
69          if (key=='+') {
70              pixelSize++;
71          }
72          else if (key=='-') {
73              if (pixelSize>0) pixelSize--;
74          }
75          else {
76              style = key;
77          }
78      }
```

pixelSize is rectangle or oval size (initially 10)

Set style to m=*mosaic* or p=*pointillism*

- In *draw()* choose what style to use to display image
- If we didn't override *draw()* or didn't press *m or p*, then WebCam's *draw()* method is called – that displays last captured image unaltered
- Here we will display rectangles or ovals instead of the camera image
- Note: we don't need to override *processImage()* because we won't be displaying camera image, we will use image pick up colors
- We will draw on the screen instead of displaying the camera image (see next slide)

WebcamRendering.java: Display ovals or rectangles instead of captured image itself

WebCamRendering.java

```
21    public void mosaic(Graphics g) {
22        // Usual loops, but step by "pixel" size and draw a rectangle of the appropriate color.
23        // Also note <=, to get that last rectangle.
24        // Nested loop over every pixel
25        for (int y = 0; y <= image.getHeight() - pixelSize; y += pixelSize) {
26            for (int x = 0; x <= image.getWidth() - pixelSize; x += pixelSize) {
27                //draw rectangle with interior color picked up from image
28                g.setColor(new Color(image.getRGB(x,y)));
29                g.fillRect(x, y, pixelSize, pixelSize);
30                //draw black border around rectangle
31                g.setColor(Color.black);
32                g.drawRect(x, y, pixelSize, pixelSize);
33            }
34        }
35    }
36
37    /**
38     * Renders the image as a set of ellipses at random positions.
39     */
40    public void pointillism(Graphics g) {
41        // Draw some number of points determined by the image and "pixel" sizes.
42        int numPoints = image.getWidth() * image.getHeight() * 5 / pixelSize;
43        for (int p=0; p<numPoints; p++) {
44            // Pick a random position and size
45            int x = (int) (Math.random() * image.getWidth());
46            int y = (int) (Math.random() * image.getHeight());
47            int s = (int) (Math.random() * pixelSize) + 1;
48
49            // Draw an ellipse there, colored by the pixel's color
50            g.setColor(new Color(image.getRGB(x,y)));
51            g.fillOval(x, y, s, s);
52        }
53    }
```

Loop over x,y locations

Skip forward by *pixelSize* (defaults to 10)

Draw rectangle of pixelSize using color

Pick up color from image at x,y

Draw a black rectangle around color to highlight

- Both mosaic and pointillism loop over image
- Pick up underlying color from camera image for some pixels
- Display as rectangles or ovals of picked up color
- Does not display the camera image itself

- Pointillism works somewhat similarly
- Makes a number of randomly sized ovals (like Blobs)

Agenda

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

Demo: WebcamColorTracking

Notes:

Tracks a color over time

- Click mouse to pick up color from image (use finger tip)
- Will find point with closest color match
- Draws oval around that point as new images arrive (move finger to demonstrate)
- Not too sophisticated, but generally works (Autofocus sometimes causes inaccurate tracking)

WebcamColorTracking.java: Pick up a color on mouse press, then track that color

WebCamColorTracking.java

```
42    /**
43     * DrawingGUI method, here setting trackColor from where mouse was pressed
44     */
45    @Override
46    public void handleMousePress(int x, int y) {
47        System.out.println("Got mouse press");
48        if (image != null) {
49            trackColor = new Color(image.getRGB(x, y));
50            System.out.println("tracking " + trackColor);
51        }
52    }
53
54    /**
55     * Webcam method, here also showing tracked color point (called when WebCam repaints in doInBackground)
56     */
57    @Override
58    public void draw(Graphics g) {
59        super.draw(g);
60        if (trackColor != null) {
61            // Draw circle at point with color closest to trackColor, then draw circle border in the inverse color
62            Point where = track();
63            g.setColor(trackColor);
64            g.fillOval(where.x, where.y, 15, 15);
65
66            //draw circle border in inverse color
67            ((Graphics2D)g).setStroke(new BasicStroke(4)); // thick border
68            g.setColor(new Color(255-trackColor.getRed(), 255-trackColor.getGreen(), 255-trackColor.getBlue()));
69            g.drawOval(where.x, where.y, 15, 15);
70        }
71    }
72
```

- On mouse press, get Color from image at coordinates of press, save in *trackColor*
- draw image on screen
- *track()* finds pixel on image that most closely matches *trackColor* (see next slide)
- Return type is Point
- Draw an oval around Point

WebcamColorTracking.java: Pick up a color on mouse press, then track that color

WebCamColorTracking.java

```
16    /**
17     * Determines which point is closest to the trackColor, returns it
18     */
19    private Point track() {
20        int cx = 0, cy = 0; // coordinates with best matching color
21        int closest = 10000; // start with a too-high number so that everything will be smaller
22        // Nested loop over every pixel
23        for (int y = 0; y < image.getHeight(); y++) {
24            for (int x = 0; x < image.getWidth(); x++) {
25                // Euclidean distance squared between colors
26                Color c = new Color(image.getRGB(x,y));
27                int d = (c.getRed() - trackColor.getRed()) * (c.getRed() - trackColor.getRed())
28                        + (c.getGreen() - trackColor.getGreen()) * (c.getGreen() - trackColor.getGreen())
29                        + (c.getBlue() - trackColor.getBlue()) * (c.getBlue() - trackColor.getBlue());
30
31                //track point with closest color to trackColor (so far)
32                if (d < closest) {
33                    closest = d;
34                    cx = x; cy = y;
35                }
36            }
37        }
38        //return point that had the closest color
39        return new Point(cx,cy);
40    }
```

Loop over all pixels and return x,y location of pixel with closest color match to *trackColor*

- **Get Color for each pixel**
- **Compare with *trackColor***
- **Save *x,y* with closest color**

- **Could we just use *Math.abs(c-trackColor)*?**
- **No, because a color is really a 24-bit number**
- **Red is leftmost 8 bits**
- **A 1 bit change in red color would lead to a large difference in *d***

Return closest point as variable of type Point

Agenda

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

Demo: WebcamDiff.java

Notes:

Subtracts previous image from current image

- Run and wave hand in front of camera
- Subtracts previous image from current image
- If no change in pixel, then subtraction results in zero = black pixel
- All pixels will be black in a static scene

WebcamDiff.java: Subtract the previous frame from the current frame

WebCamDiff.java

```
22      @Override
23      public void processImage() {
24          //Remember, frames are stored in instance variable called image in WebCam
25          //processImage() is called by WebCam.java every time a new image is ready
26
27          //create new image from camera's frame grab
28          BufferedImage curr = new BufferedImage(image.getColorModel(), image.copyD
29          if (prev != null) { // skip first frame
30              // Nested loop over every pixel
31              for (int y = 0; y < image.getHeight(); y++) {
32                  for (int x = 0; x < image.getWidth(); x++) {
33                      //get pixel color from this image (c1), and previous image (c
34                      Color c1 = new Color(image.getRGB(x,y));
35                      Color c2 = new Color(prev.getRGB(x,y));
36                      //calculate difference in color between c1 and c2 (use Math.a
37                      Color c = new Color(Math.abs(c1.getRed()-c2.getRed()),
38                                          Math.abs(c1.getGreen()-c2.getGreen()),
39                                          Math.abs(c1.getBlue()-c2.getBlue()));
40                      //write color difference to image (if c1 == c2, then this col
41                      image.setRGB(x,y,c.getRGB());
42                  }
43              }
44          }
45          //update prev to current and wait for next image
46          prev = curr;
47      }
```

- Keep a copy of the current and previous images
- Loop over x,y locations
- Get color of current image and previous image at x,y
- Subtract current and previous colors at x,y
- Zero if same color (black)
- Update image to be displayed with color difference (black if no difference frame to frame)
- Set previous image to current image
- Why save current image (line 28)
- Because updating *image* as we go but want previous image unaltered

Agenda

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

Demo: WebcamLoop.java

Notes:
Records images into buffer or plays back from buffer (in reverse for “fun”)

- Click mouse to toggle between recording and playback

WebcamLoop.java: Record images into buffer and playback in reverse

WebCamLoop.java

```
18 public class WebcamLoop extends Webcam {
19     private ArrayList<BufferedImage> frames; //will store our recorded
20     private boolean recording = true;    // start by recording
21     private int frame = 0;               // where in the loop we are (f
22
23     public WebcamLoop() {
24         frames = new ArrayList<BufferedImage>(); //we start off record
25         System.out.println("recording:"+recording);
26     }
27
28     /**
29      * Override to start/stop recording
30      * @param event
31      */
32     @Override
33     public void handleMousePress(int x, int y) {
34         // Toggle record/play and go back to frame 0 (with cleared fra
35         recording = !recording;
36         if (recording) {
37             frames = new ArrayList<BufferedImage>();
38         }
39         else {
40             frame = frames.size()-1;     //start at last frame captured
41         }
42         System.out.println("recording:"+recording);
43     }
```

- ArrayList acts as a buffer
- When recording, add each image captured to end of buffer
- When playing back, get each image in buffer and display in order (here in reverse order for “fun”)
- Toggle recording or playback mode on mouse press
- ! means “not”, changes true to false and vice versa
- Create new buffer if starting to record

WebcamLoop.java: Record images into buffer and playback in reverse

WebCamLoop.java

```
48      @Override
49      public void draw(Graphics g) {
50          if (recording) {
51              //recording, so show images as they arrive
52              g.drawImage(image, 0, 0, null);
53          }
54          else {
55              //playing back, frame image at index frame (set to last index on m
56              g.drawImage(frames.get(frame), 0, 0, null);
57              frame--;
58              //careful not to go below frame 0 (first frame)
59              if (frame < 0) frame = frames.size()-1;
60              System.out.println("played:"+frame);
61          }
62      }
63
64      /**
65       * Webcam method, here storing frames.
66       */
67      @Override
68      public void processImage() {
69          //save image in frames ArrayList if recording, otherwise no action to
70          if (recording) {
71              BufferedImage copy = new BufferedImage(image.getColorModel(), imag
72              frames.add(copy);
73              System.out.println("recorded:"+frames.size());
74          }
75      }
76
```

- In *draw()*, if recording, just show latest image
- If not recording, playback in reverse
  - Get last image from buffer
  - Show that image
  - Get previous image
  - Show that image
  - Repeat
- Could we have added image to buffer here instead of in *processImage()*?
- Sure! *draw()* is called right after *processImage()*

- If recording, add each new frame to the ArrayList buffer
- Why did we copy *image*?
- Because *image* will change

Agenda

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

Demo: WebcamBg.java

Notes:

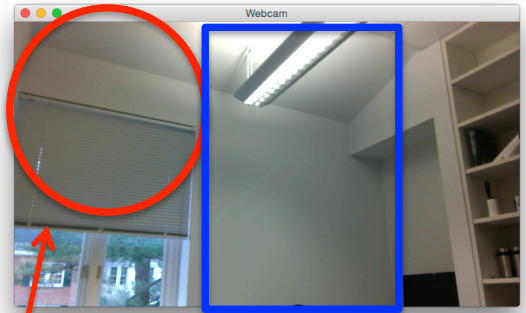
Makes a “green screen” type of effect

- Load a scenery image (Baker tower)
- Click to capture background image from camera
- Now move around
- Compare current and background image color at each x,y location
- If not much color difference, color pixel at x,y with scenery color (e.g., Baker tower)
- Else, color pixel with current image
- Result is you appear to be in front of Baker tower

WebcamBg.java uses three images to make you appear to be somewhere else

scenery

- Static image
- This is where we want you to appear to be located



background

Static snapshot of the camera's view without you in it

image

Live image as it comes from the camera

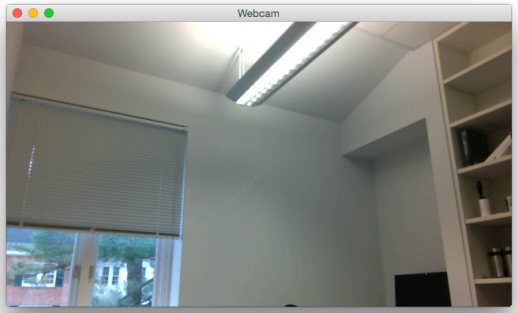
- **This portion of the background and live image are the same (mostly)**
- **Show scenery (Baker tower) there**

- **This portion of the background and live image are the different**
- **Show live camera image there**

WebcamBg.java uses three images to make you appear to be somewhere else

scenery

- Static image
- This is where we want you to appear to be located



background

Static snapshot of the camera's view without you in it

image

Live image as it comes from the camera

- **Why is this part Baker instead of my arm?**
- **Background is close to my shirt color there**

WebcamBg.java: Replace background with image we choose (green screen effect)

WebCamBg.java

```
21 public class WebcamBg extends Webcam {
22     private static final int backgroundDiff=250; // setup: threshold for considering
23
24     private BufferedImage background;        // the stored background grabbed from t
25     private BufferedImage scenery;           // the replacement background (e.g., Ba
26
27     public WebcamBg(BufferedImage scenery) {
28         this.scenery = scenery;
29     }
30
31     /**
32      * DrawingGUI method, here setting background as a copy of the current image.
33      */
34     @Override
35     public void handleMousePress(int x, int y) {
36         if (image != null) {
37             //save background image that we will subtract out
38             background = new BufferedImage(image.getColorModel(), image.copyData(nul
39             System.out.println("background set");
40         }
41     }
```

Define threshold, if color difference less than this, use scenery image, else camera image

Load scenery image (Baker tower)

On mouse press, save current image as background

WebcamBg.java: Replace background with image we choose (green screen effect)

WebCamBg.java

```
48      public void processImage() {
49          if (background != null) {
50              // Nested loop over every pixel
51              for (int y = 0; y < Math.min(image.getHeight(), scenery.getHeight()); y++) {
52                  for (int x = 0; x < Math.min(image.getWidth(), scenery.getWidth()); x++) {
53                      // Euclidean distance squared between colors
54                      Color c1 = new Color(image.getRGB(x,y));
55                      Color c2 = new Color(background.getRGB(x,y));
56                      int d = (c1.getRed() - c2.getRed()) * (c1.getRed() - c2.getRed())
57                              + (c1.getGreen() - c2.getGreen()) * (c1.getGreen() - c2.getGreen())
58                              + (c1.getBlue() - c2.getBlue()) * (c1.getBlue() - c2.getBlue());
59                      //check if distance less than threshold to replace image with scenery, otherw
60                      if (d < backgroundDiff) {
61                          // Close enough to background, so replace
62                          image.setRGB(x,y,scenery.getRGB(x,y));
63                      }
64                  }
65              }
66          }
67      }
```

If background is set, loop over each x,y location

Compare color of camera image with *background* image

If not much color difference compared with *background* image (e.g., no change from background), show *scenery* color for this pixel, else don't change live camera image at this pixel