Drawing Line,Rectangle, Circle, Polygon

Sung Soo Hwang

Drawing Function

- Rectangle
  - void rectangle(Mat& img, Point pt1, Point pt2, const Scalar& color, int thickness=1, int lineType=8, int shift=0)
    - img – image
    - pt1: vertex of the rectangle
    - pt2: vertex of the rectangle opposite to pt1
    - color: rectangle color or brightness(for grayscale)
    - thickness: thickness of lines that make up the rectangle
      - Negative values to draw filled rectangle
    - lineType: type of the line. See line() description
    - shift: number of fractional bits in the point coordinates
      - $Point(x, y) \rightarrow Point2f(x \times 2^{-shift}, y \times 2^{-shift})$
    - void rectangle(Mat& img, Rect rect, const Scalar& color, int thickness=1, int lineType=8, int shift=0)
      - rect: alternative specification of the drawn rectangle
        - Rect(x_LT, y_LT, width, height)

Drawing Function

- Rectangle
  - Example code

```
int main(){
	Mat image = imread("lena.png");
	Rect rect = Rect(10, 10, 100, 100); // LT position, width, height
	rectangle(image, rect, Scalar(255, 0, 0), 4, 8, 0);
	imshow("image",image);
	waitKey(0);
	return 0;
}
```

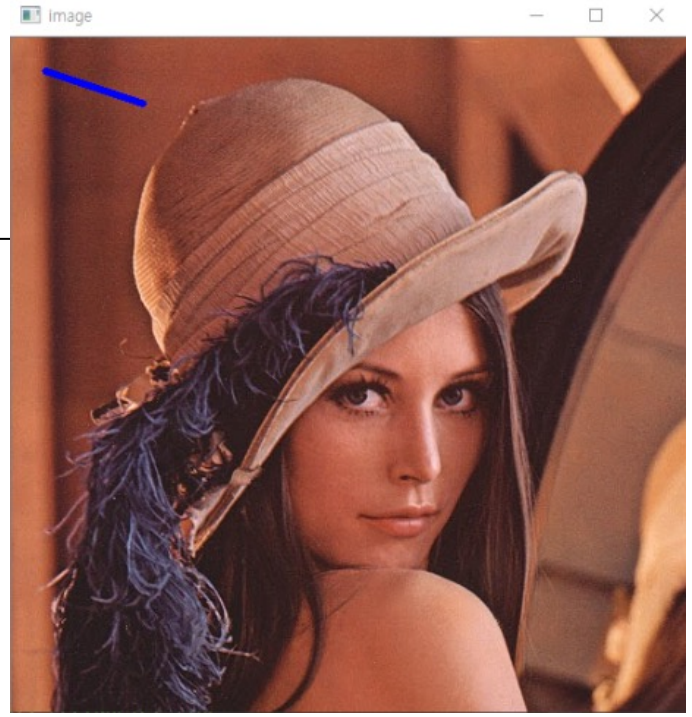
Drawing Function

- Line/Circle
  - void line(Mat& img, Point pt1, Point pt2, const Scalar& color, int thickness=1, int lineType=8, int shift=0)
    - pt1: first point of the line segment
    - pt2: second point of the line segment
    - lineType:
      - 8: 8-connected line.
      - 4: 4-connected line.
      - CV_AA: antialiased line
  - void circle(Mat& img, Point center, int radius, const Scalar& color, int thickness=1, int lineType=8, int shift=0)
    - Center: center of the circle
    - Radius: radius of the circle

Drawing Function

- Line/Circle
  - Example code

```
int main(){
	Mat image = imread("lena.png");
	Point p1(25, 25), p2(100, 50);
	line(image, p1, p2, Scalar(255, 0, 0), 3, 8, 0);
	imshow("image",image);
	waitKey(0);
	return 0;
}
```



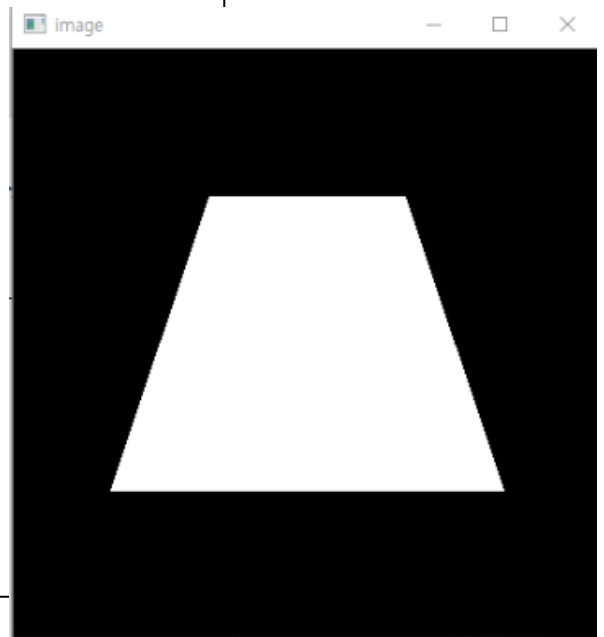
Drawing Function

- Polygon
  - void fillPoly(Mat& img, const Point** pts, const int* npts, int ncontours, const Scalar& color, int lineType=8, int shift=0, Point offset=Point())
    - img – image
    - pts – Array of polygons where each polygon is represented as an array of points
    - npts – Array of polygon vertex counters
    - ncontours – number of contours that bind the filled region
    - color – polygon color
    - lineType – type of the polygon boundaries
    - shift – number of fractional bits in the vertex coordinates
    - offset – optional offset of all points of the contours

Drawing Function

- Polygon
  - Example code

```
int main() {
    Mat image = Mat::zeros(400, 400, CV_8UC3);
    int w = 400;

    Point trapezoid[1][4];
    trapezoid[0][0] = Point(w*2 / 6, w / 4);
    trapezoid[0][1] = Point(w*4 / 6, w / 4);
    trapezoid[0][2] = Point(w*5 / 6, w*3 / 4);
    trapezoid[0][3] = Point(w / 6, w*3 / 4);

    const Point* ppt[1] = { trapezoid[0] };
    int npt[] = { 4 };

    fillPoly(image, ppt, npt, 1, Scalar(255, 255, 255), 8);
    imshow("image", image);

    waitKey(0);
}
```



Writing Text

Sung Soo Hwang

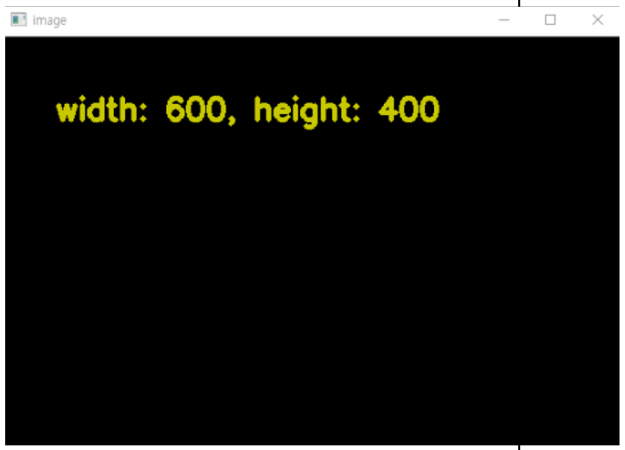
Drawing Function

- Write text
  - void putText(Mat& img, const string& text, Point org, int fontFace, do uble fontScale, Scalar color, int thickness=1, int lineType=8, bool bott omLeftOrigin=false
    - text: text string to be drawn
    - org: bottom-left corner of the text string in the image
    - font: CVFont structure using InitFont()
    - fontFace: FONT_TYPE(FONT_HERSHEY_SIMPLEX, FONT_HERSHEY_PLAIN, FONT_ HERSHEY_DUPLEX, FONT_HERSHEY_COMPLEX, FONT_HERSHEY_TRIPLEX, … can be combined with FONT_HERSHEY_ITALIC)
    - fontScale: Font scale factor that is multiplied by the font-specific base size
    - bottomLeftOrgin: when true, the image data origin is at the bottom-left corner. Otherwise, it is at the top-left corner

Histogram equalization

- Write text
  - String cv::format(const char *fmt, ...)
    - Returns a text string formatted using the printf-like expression.
    - Params: fmt – printf-compatible formatting specifiers.
  - Example code

```
int main() {
	// Create black empty images
	Mat image = Mat::zeros(400, 600, CV_8UC3);
	int w = image.cols;
	int h = image.rows;
	putText(image, format("width: %d, height: %d", w, h),
	Point(50, 80), FONT_HERSHEY_SIMPLEX, 1,
	Scalar(0, 200, 200), 4);
	imshow("image", image);

	waitKey(0);
	return(0);
}
```



unesco
University Network
Member

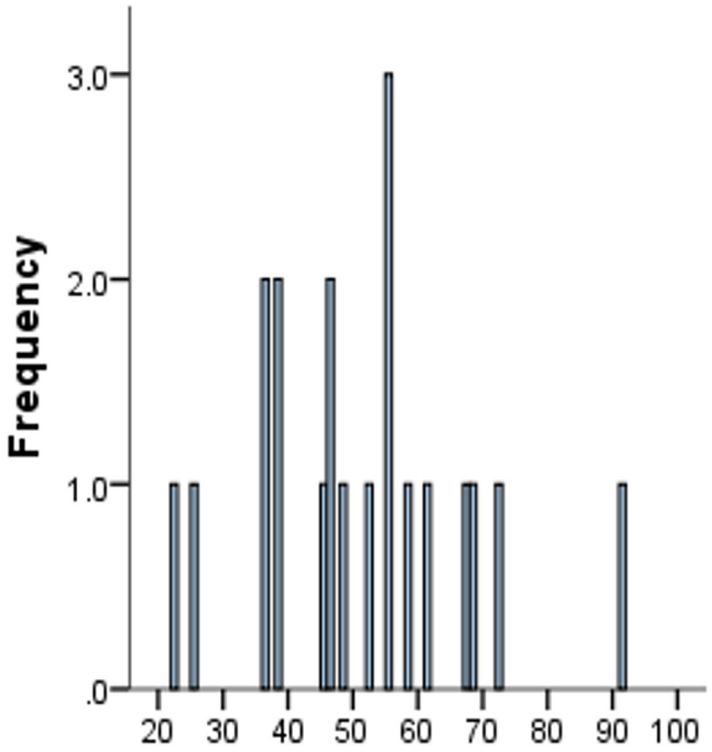
Ministry of
Education

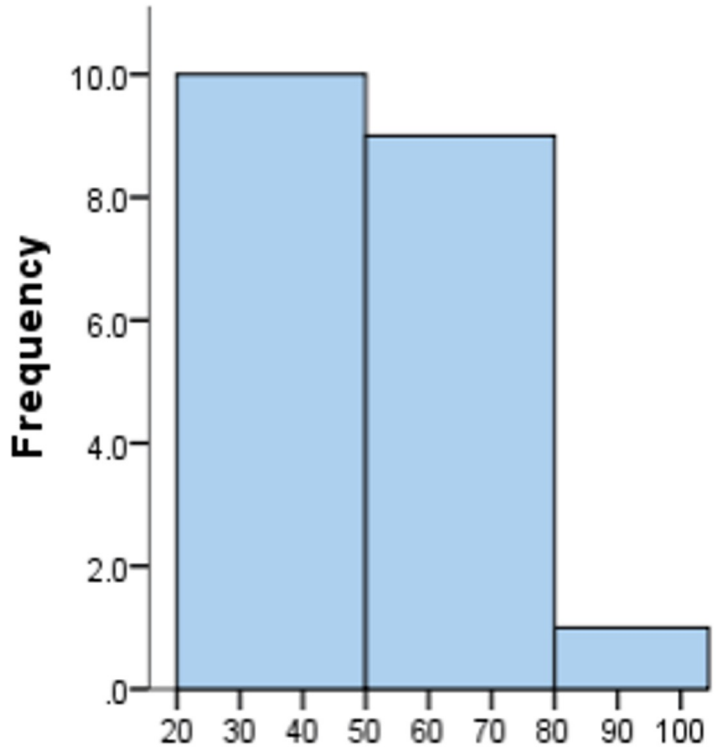
Draw Histogram

Sung Soo Hwang

Introduction

- Histograms are the basis for numerous spatial domain process ing techniques
  - Setting the proper number of bins(or bin width) is important

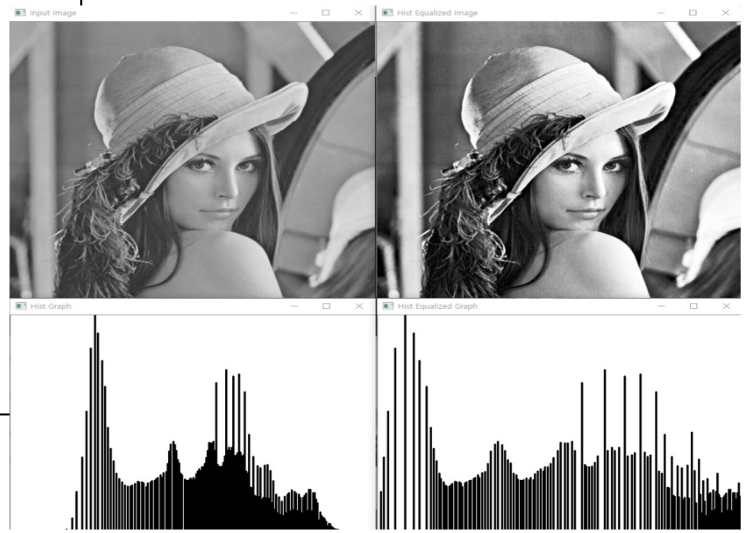




Drawing Function

- Example code

```
int main() {
	Mat image;
	Mat hist_equalized_image;
	Mat hist_graph;
	Mat hist_equalized_graph;

	image = imread("lena.png", 0);
	if (!image.data) exit(1); //Check image

	equalizeHist(image, hist_equalized_image); //histogram equlization

	hist_graph = drawHistogram(image);
	hist_equalized_graph = drawHistogram(hist_equalized_image);

	imshow("Input Image", image);
	imshow("Hist Equalized Image", hist_equalized_image);
	imshow("Hist Graph", hist_graph);
	imshow("Hist Equalized Graph", hist_equalized_graph);

	waitKey(0);
	return 0;
}
```



Histogram equalization

- Example code

```
Mat drawHistogram(Mat src){
	Mat hist, histImage;
	// establish the number of bins
	int i, hist_w, hist_h, bin_w, histSize;
	float range[] = { 0, 256 };
	const float* histRange = { range };

	hist_w = 512;
	hist_h = 400;
	histSize = 256;
	bin_w = cvRound((double)hist_w / histSize);

	//draw the histogram
	histImage = Mat(hist_h, hist_w, CV_8UC3, Scalar(255, 255, 255));

	// compute the histograms
	// &src: input image, 1: #of src image, 0: #of channels numerated from 0 ~ channels()-1, Mat(): optional mask
	// hist: output histogram, 1: histogram dimension, &histSize: array of histogram size,  &histRange: array of histogram's boundaries
	calcHist(&src, 1, 0, Mat(), hist, 1, &histSize, &histRange);

	// Fit the histogram to [0, histImage.rows]
	// hist: input Mat, hist: output Mat, 0: lower range boundary of range normalization, histImage.rows: upper range boundary
	// NORM_MINMAX: normalization type, -1: when negative, the ouput array has the same type as src, Mat(): optional mask
	normalize(hist, hist, 0, histImage.rows, NORM_MINMAX, -1, Mat());

	for (i = 0; i < histSize; i++)
	{
		rectangle(histImage, Point(bin_w * i, hist_h), Point(bin_w * i+hist_w/histSize, hist_h - cvRound(hist.at<float>(i))), Scalar(0, 0, 0), -1);
	}
	return histImage;
}
```

Write an image & a video

Sung Soo Hwang

Image Function

- Write images
  - void imwrite(const String& filename, InputArray img,
    const std::vector<int>& params = std::vector<int>());
    - filename: Name of the file
    - img : (Mat or vector of Mat) Image or Images to be saved.
    - params : Format-specific parameters encoded as pairs (paramId_1, paramValue_1, paramId_2, paramValue_2, ...) see cv::ImwriteFlags

VideoWriter Class

- Constructor
  - VideoWriter::VideoWriter(constr String& filename, int fourcc, double fps, Size frameSize, bool isColor = true
    - filename: Name of the output video file
    - fourcc: 4-character code of codec used to compress the frames.
    - fps: Framerate of the created video stream.
    - frameSize: Size of the video frames.
    - isColor: If it is not zero, the encoder will expect and encode color frames, otherwise it will work with grayscale frames (the flag is currently supported on Windows only).

VideoWriter Class

- Member functions
  - void VideoWriter::write(const Mat& image)
    - image: The written frame. In general, color images are expected in BGR format.
    - The function writes the specified image to video file. It must have the same size as has been specified when opening the video writer.
  - void VideoWriter::release()
    - Closes the video writer.
    - The method is automatically called by subsequent by VideoWriter destructor.

Writing a video

- Example code (writing from a Webcam)

```
int main(){
	VideoCapture cap(0);
	// Check if camera opened successfully
	if(!cap.isOpened()){
		cout << "Error opening video stream" << endl;
		return -1;}
	// Default resolutions of the frame are obtained.The default resolutions are system dependent.
	int frame_width = cap.get(cv::CAP_PROP_FRAME_WIDTH);
	int frame_height = cap.get(cv::CAP_PROP_FRAME_HEIGHT);

	// Define the codec and create VideoWriter object.The output is stored in 'outcpp.avi' file.
	VideoWriter video("outcpp.avi", cv::VideoWriter::fourcc('M','J','P','G'), 10, Size(frame_width,frame_height));

	while(1){
		Mat frame;
		// Capture frame-by-frame
		cap >> frame;

		// If the frame is empty, break immediately
		if (frame.empty()) break;

		// Write the frame into the file 'outcpp.avi'
		video.write(frame);

		// Display the resulting frame
		imshow( "Frame", frame );

		// Press ESC on keyboard to exit
		char c = (char)waitKey(1);
		if( c == 27 ) break; }
	cap.release();
	video.release(); // Closes all the frames
	destroyAllWindows();
	return 0;
}
```