# Advanced Lane Finding Project

The goals / steps of this project are the following:

- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- Detect lane pixels and fit to find the lane boundary.
- Determine the curvature of the lane and vehicle position with respect to center.
- Warp the detected lane boundaries back onto the original image.
- Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

# Rubric Points

## Camera Calibration

### 1. Have the camera matrix and distortion coefficients been computed correctly and checked on one of the calibration images as a test?

The code for this step is contained in the **second** code cell of the IPython notebook. The strategy as taught in the lessons is applied here. I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, `objp` is just a replicated array of coordinates, and `objpoints` will be appended with a copy of it every time I successfully detect all chessboard corners in a test image.

`imgpoints` will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.

I then used the output `objpoints` and `imgpoints` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function. I applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result:

## Pipeline (single images)

### 1. Has the distortion correction been correctly applied to each image?

Yes, the output images can be found in the output_images folder.

### 2. Has a binary image been created using color transforms, gradients or other methods?

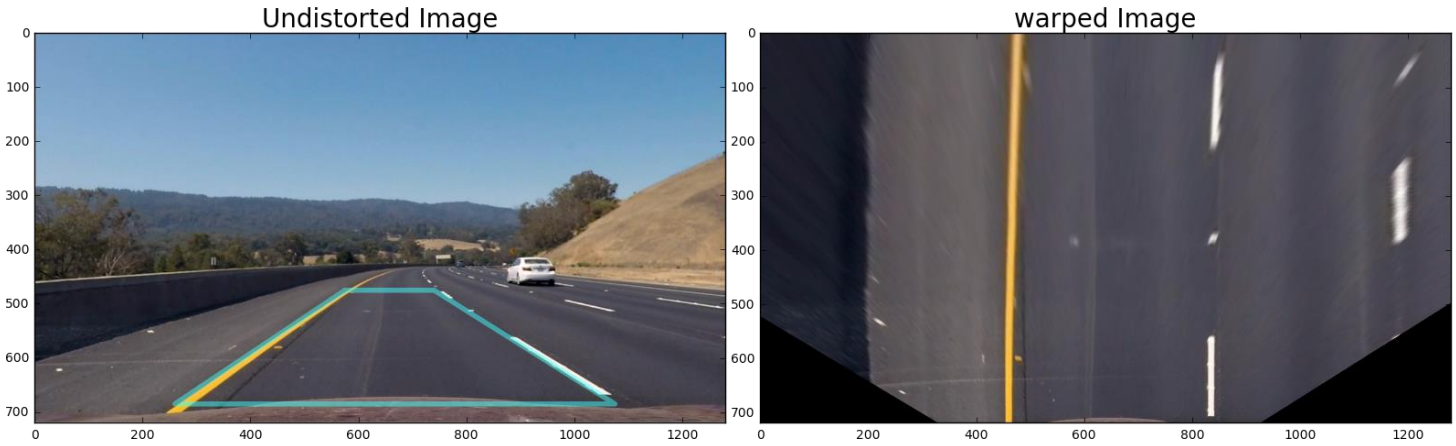
Yes, the output images can be found in the output_images folder.

### 3. Has a perspective transform been applied to rectify the image?

The code for my perspective transform is includes a function called `warp()`, which appears in the 5th code cell of the notebook. It takes input img, ( `src` ) and destination ( `dst` ) points. I chose the hardcode the source and destination points in the following manner:

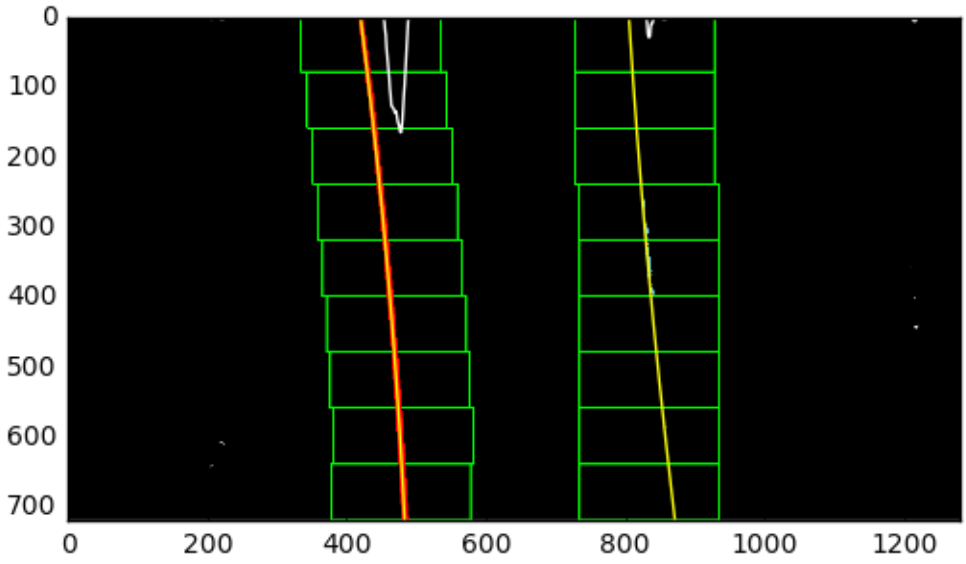
```
src = np.float32([(575,475),
                  (740,475),
                  (260,685),
                  (1075,685)])

dst = np.float32([(450,0),
                  (w-450,0),
                  (450,h),
                  (w-450,h
```

I verified that my perspective transform was working as expected by drawing the `src` and `dst` points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image.




## 4. Have lane line pixels been identified in the rectified image and fit with a polynomial?

Then I did some other stuff and fit my lane lines with a 2nd order polynomial, and the output looked kinda like this:



### 5. Having identified the lane lines, has the radius of curvature of the road been estimated? And the position of the vehicle with respect to center in the lane?

Yep, again, the resulting output images can be found in the output_images folder.

## Pipeline (video)

### 1. Does the pipeline established with the test images work to process the video?

It sure does! The video is attached in the package, but there is one problem in the video. The lane line is followed correctly, the radius of curvature and displacement from the center is also computed and displayed without any problem but I cannot see the subsequent frames in the video.

## README

### 1. Has a README file been included that describes in detail the steps taken to construct the pipeline, techniques used, areas where improvements could be made?

You're reading it!

# Discussion

The approach, I adopted in completing this project, is graphically illustrated below: