# Advanced Lane Finding Project

## Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.

In the two blocks below I computed the camera calibration matrix and distortion coefficients. I first use cv2.findChessboardCorners to find image points and generate object points. I then use cv2.calibrateCamera to compute the camera calibration matrix and distortion coefficients.

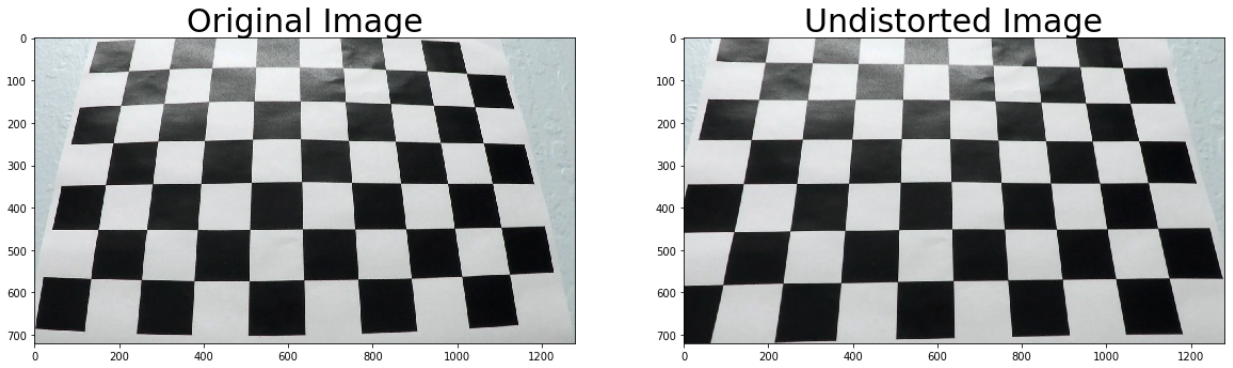
In [1]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import glob
import pickle
from scipy import signal

%matplotlib inline
```

In [2]:
```
nx = 9
ny = 6

images = glob.glob("camera_cal/calibration*.jpg")

objpoints = []
imgpoints = []

objp = np.zeros((nx*ny,3), np.float32)
objp[:,:2] = np.mgrid[0:nx,0:ny].T.reshape(-1,2)

imgs = []

for fname in images:
    img = cv2.imread(fname)
    imgs.append(img)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(gray, (nx, ny), None)

    if ret == True:
        imgpoints.append(corners)
        objpoints.append(objp)

        cv2.drawChessboardCorners(img, (nx, ny), corners, ret)

img_size = (imgs[0].shape[1], imgs[0].shape[0])

# Calibrate camera
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, img

# Save calibration matrix and distortion coefficients
dist_pickle = {}
dist_pickle["mtx"] = mtx
dist_pickle["dist"] = dist
pickle.dump(dist_pickle, open("camera_calibration_result.p", "wb"))
```

## Apply a distortion correction to raw images.

In [3]:
```
img = cv2.imread('camera_cal/calibration3.jpg')

dst = cv2.undistort(img, mtx, dist, None, mtx)
cv2.imwrite('calibration_wide/test_undist.jpg',dst)

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20,10))
ax1.imshow(img)
ax1.set_title('Original Image', fontsize=30)
ax2.imshow(dst)
ax2.set_title('Undistorted Image', fontsize=30)
```

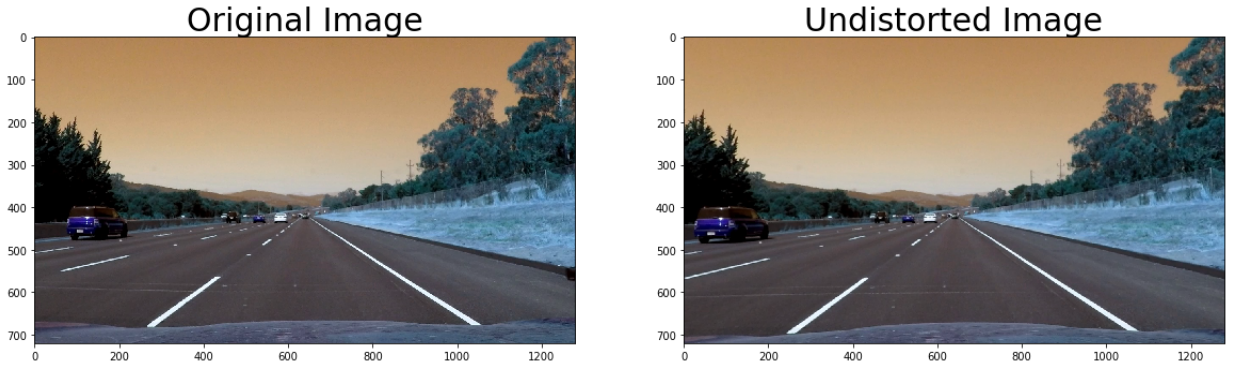
Out[3]: <matplotlib.text.Text at 0x1181c5208>




The two images above demonstrate how the distortion in the original image is corrected.

In [4]:
```
raw = cv2.imread("test_images/straight_lines2.jpg")
imshape = raw.shape

image = cv2.undistort(raw, mtx, dist, None, mtx)

# Visualize undistortion
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20,10))
ax1.imshow(raw)
ax1.set_title('Original Image', fontsize=30)
ax2.imshow(image)
ax2.set_title('Undistorted Image', fontsize=30)
```

Out[4]: <matplotlib.text.Text at 0x11bfc8198>

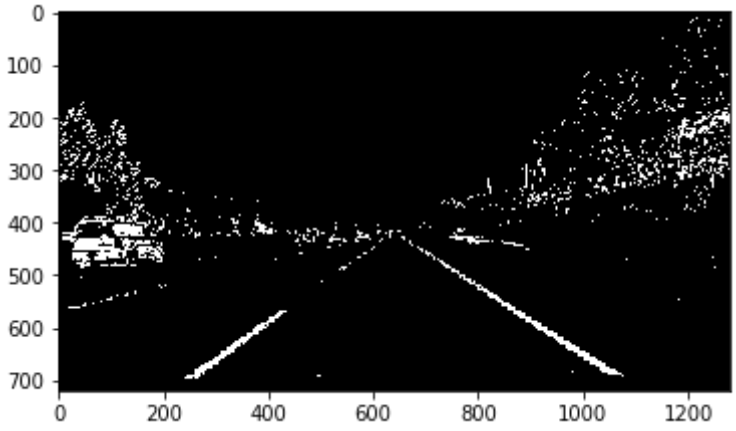



## Use color transforms, gradients, etc., to create a

## thresholded binary image.

In [5]:
```
def apply_threshold_v2(image, xgrad_thresh=(20,100), s_thresh=(170,255)):
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

    sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0)
    abs_sobelx = np.absolute(sobelx)
    scaled_sobel = np.uint8(255*abs_sobelx/np.max(abs_sobelx))

    sxbinary = np.zeros_like(scaled_sobel)
    sxbinary[(scaled_sobel >= xgrad_thresh[0]) & (scaled_sobel <= xgrad_thre

    hls = cv2.cvtColor(image, cv2.COLOR_RGB2HLS)
    s_channel = hls[:,:,2]

    s_binary = np.zeros_like(s_channel)
    s_binary[(s_channel >= s_thresh[0]) & (s_channel <= s_thresh[1])] = 1

    color_binary = np.dstack(( np.zeros_like(sxbinary), sxbinary, s_binary))

    combined_binary = np.zeros_like(sxbinary)
    combined_binary[(s_binary == 1) | (sxbinary == 1)] = 1

    return combined_binary

xgrad_thresh_temp = (40,100)
s_thresh_temp=(150,255)

combined_binary = apply_threshold_v2(image, xgrad_thresh=xgrad_thresh_temp,
plt.imshow(combined_binary, cmap="gray")
```

Out[5]:
```
<matplotlib.image.AxesImage at 0x11d0378d0>
```

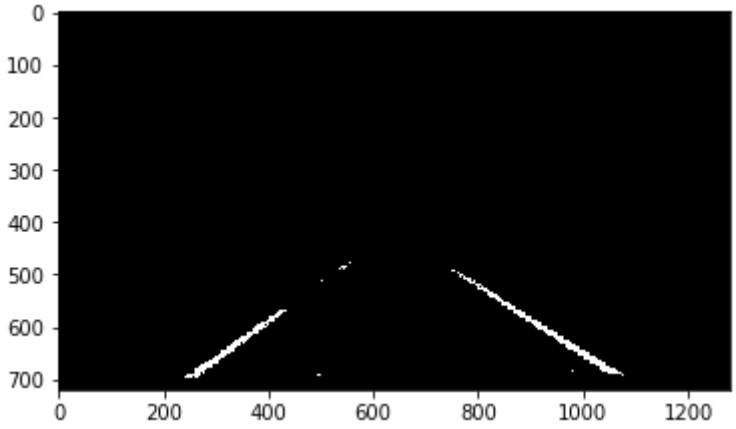



The image above shows a thresholded image. I use a combination of thresholding x gradient on grayscaled image and thresholding S channel in color image. I combine the two binary thresholds to generate a binary image.

## Apply a perspective transform to rectify binary image ("birds-eye view").

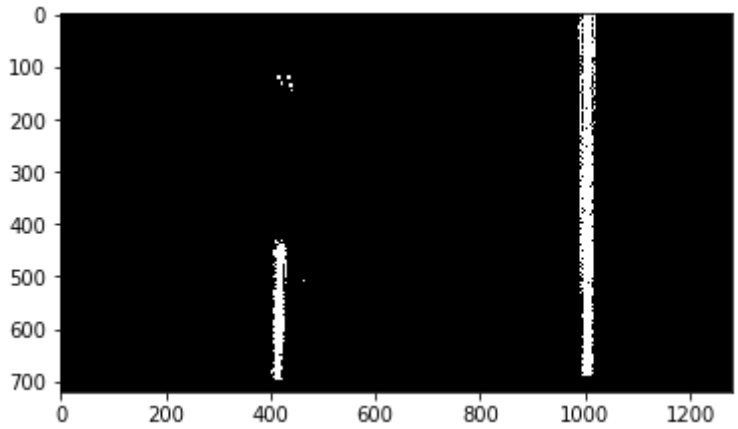
In [6]:
```
def region_of_interest(img, vertices):
    mask = np.zeros_like(img)

    if len(img.shape) > 2:
        channel_count = img.shape[2]
        ignore_mask_color = (255,) * channel_count
    else:
        ignore_mask_color = 255

    cv2.fillPoly(mask, vertices, ignore_mask_color)

    masked_image = cv2.bitwise_and(img, mask)
    return masked_image


vertices = np.array([[(0,imshape[0]),(550, 470), (700, 470), (imshape[1],ims
masked_image = region_of_interest(combined_binary, vertices)
plt.imshow(masked_image, cmap="gray")
```

Out[6]: <matplotlib.image.AxesImage at 0x11d0a9dd8>



In [7]:
```
src = np.float32(
    [[200, 720],
     [510, 500],
     [765, 500],
     [1100, 720]])

dst = np.float32(
    [[400,720],
     [400,0],
     [1000,0],
     [1000,720]])

M = cv2.getPerspectiveTransform(src, dst)
Minv = cv2.getPerspectiveTransform(dst, src)

warped = cv2.warpPerspective(combined_binary, M, (imshape[1], imshape[0]), f
plt.imshow(warped, cmap="gray")
```

Out[7]: <matplotlib.image.AxesImage at 0x11c8adef0>




The code above and the image following demonstrate how I did my perspective transform and the result of the transform. I assumed that the road is on a plane and the plane does not change throughout the video. I selected four points on the original image, two on the left line and two on the right line and I then map the four points to two straight lines. I use cv2.getPerspectiveTransform to calculate the matrix.

## Detect lane pixels and fit to find the lane boundary.

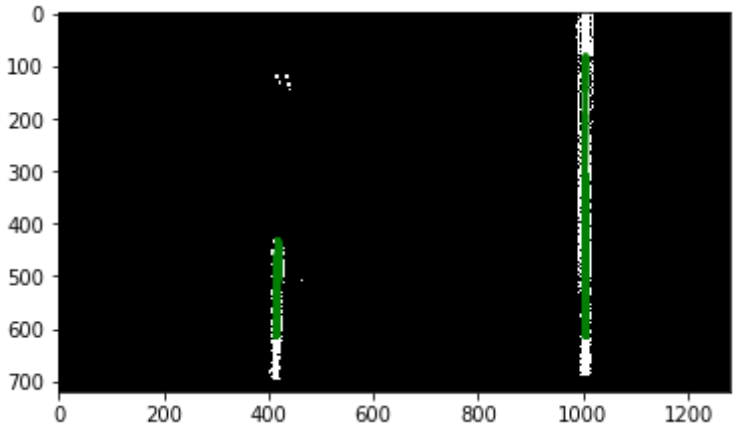
In [8]:
```
horizontal_offset = 40

def histogram_pixels(warped_thresholded_image, offset=50, steps=6,
                     window_radius=200, medianfilt_kernel_size=51,
                     horizontal_offset=50):
    left_x = []
    left_y = []
    right_x = []
    right_y = []

    height = warped_thresholded_image.shape[0]
    offset_height = height - offset
    width = warped_thresholded_image.shape[1]
    half_frame = warped_thresholded_image.shape[1] // 2
    pixels_per_step = offset_height / steps

    for step in range(steps):
        left_x_window_centres = []
        right_x_window_centres = []
        y_window_centres = []

        window_start_y = height - (step * pixels_per_step) + offset
        window_end_y = window_start_y - pixels_per_step + offset

        histogram = np.sum(warped_thresholded_image[int(window_end_y):int(wi

        histogram_smooth = signal.medfilt(histogram, medianfilt_kernel_size)

        left_peaks = np.array(signal.find_peaks_cwt(histogram_smooth[:half_f
        right_peaks = np.array(signal.find_peaks_cwt(histogram_smooth[half_f
        if len(left_peaks) > 0:
            left_peak = max(left_peaks)
            left_x_window_centres.append(left_peak)

        if len(right_peaks) > 0:
            right_peak = max(right_peaks) + half_frame
            right_x_window_centres.append(right_peak)


        if len(left_peaks) > 0 or len(right_peaks) > 0:
            y_window_centres.append((window_start_y + window_end_y) // 2)

        for left_x_centre, y_centre in zip(left_x_window_centres, y_window_c
            left_x_additional, left_y_additional = get_pixel_in_window(warpe
                                                                       y_cen
            left_x.append(left_x_additional)
            left_y.append(left_y_additional)

        for right_x_centre, y_centre in zip(right_x_window_centres, y_windov
            right_x_additional, right_y_additional = get_pixel_in_window(war
                                                                         y_c
            right_x.append(right_x_additional)
            right_y.append(right_y_additional)

    if len(right_x) == 0 or len(left_x) == 0:
        print("Init no peaks for left or right")
```

```
        print("left_x: ", left_x)
        print("right_x: ", right_x)

        horizontal_offset = 0

        left_x = []
        left_y = []
        right_x = []
        right_y = []

        for step in range(steps):
            left_x_window_centres = []
            right_x_window_centres = []
            y_window_centres = []

            window_start_y = height - (step * pixels_per_step) + offset
            window_end_y = window_start_y - pixels_per_step + offset

            histogram = np.sum(warped_thresholded_image[int(window_end_y):in
                               int(horizontal_offset):int(width - horizontal

            histogram_smooth = signal.medfilt(histogram, medianfilt_kernel_s

            left_peaks = np.array(signal.find_peaks_cwt(histogram_smooth[:ha
            right_peaks = np.array(signal.find_peaks_cwt(histogram_smooth[ha
            if len(left_peaks) > 0:
                left_peak = max(left_peaks)
                left_x_window_centres.append(left_peak)

            if len(right_peaks) > 0:
                right_peak = max(right_peaks) + half_frame
                right_x_window_centres.append(right_peak)

            if len(left_peaks) > 0 or len(right_peaks) > 0:
                y_window_centres.append((window_start_y + window_end_y) // 2

            for left_x_centre, y_centre in zip(left_x_window_centres, y_winc
                left_x_additional, left_y_additional = get_pixel_in_window(w
                                                                           y
                left_x.append(left_x_additional)
                left_y.append(left_y_additional)

            for right_x_centre, y_centre in zip(right_x_window_centres, y_wi
                right_x_additional, right_y_additional = get_pixel_in_windov

                right_x.append(right_x_additional)
                right_y.append(right_y_additional)

    return collapse_into_single_arrays(left_x, left_y, right_x, right_y)


def get_pixel_in_window(img, x_center, y_center, size):
    half_size = size // 2
    window = img[int(y_center - half_size):int(y_center + half_size), int(x_

    x, y = (window.T == 1).nonzero()
```

```
    x = x + x_center - half_size
    y = y + y_center - half_size

    return x, y

def collapse_into_single_arrays(leftx, lefty, rightx, righty):
    leftx = [x
             for array in leftx
             for x in array]
    lefty = [x
             for array in lefty
             for x in array]
    rightx = [x
              for array in rightx
              for x in array]
    righty = [x
              for array in righty
              for x in array]

    leftx = np.array(leftx)
    lefty = np.array(lefty)
    rightx = np.array(rightx)
    righty = np.array(righty)

    return leftx, lefty, rightx, righty

def fit_second_order_poly(indep, dep, return_coeffs=False):
    fit = np.polyfit(indep, dep, 2)
    fitdep = fit[0]*indep**2 + fit[1]*indep + fit[2]
    if return_coeffs == True:
        return fitdep, fit
    else:
        return fitdep

leftx, lefty, rightx, righty = histogram_pixels(warped, horizontal_offset=ho

left_fit, left_coeffs = fit_second_order_poly(lefty, leftx, return_coeffs=Tr
print("Left coeffs:", left_coeffs)
print("righty[0]: ,", righty[0], ", rightx[0]: ", rightx[0])
right_fit, right_coeffs = fit_second_order_poly(righty, rightx, return_coeff
print("Right coeffs: ", right_coeffs)

# Plot data

plt.plot(left_fit, lefty, color='green', linewidth=3)
plt.plot(right_fit, righty, color='green', linewidth=3)
plt.imshow(warped, cmap="gray")
```

```
Left coeffs: [ -1.59548390e-04   1.48929110e-01   3.83561816e+02]
righty[0]: , 419.0 , rightx[0]:  989
Right coeffs:  [  2.60912380e-05  -1.79619055e-02   1.00603923e+03]
```

Out[8]: <matplotlib.image.AxesImage at 0x11ca85208>

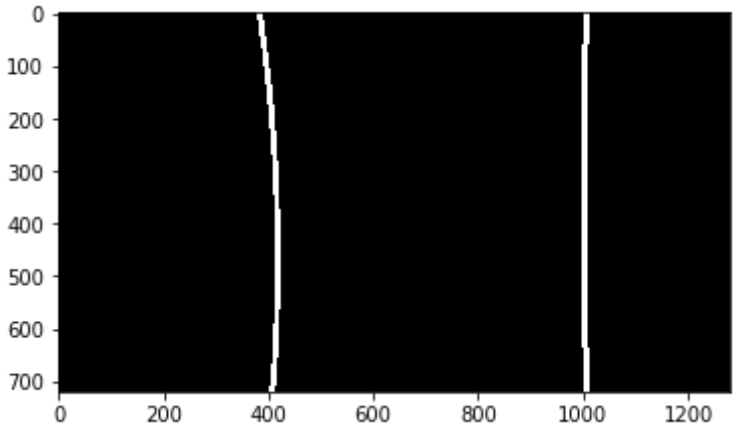



I use the following steps to identify lane-line pixels and fit their positions with a polynomial. I first divide the image into horizontal strips. For each step I count all the pixels for each strip using a histogram generated from np.sum. I then smoothen the histogram and find peaks in the left and right halves. I then get the pixels close to the two peak x coordinates.

In [9]:
```
def draw_poly(img, poly, poly_coeffs, steps, color=[255, 0, 0], thickness=10
    img_height = img.shape[0]
    pixels_per_step = img_height // steps

    for i in range(steps):
        start = i * pixels_per_step
        end = start + pixels_per_step

        start_point = (int(poly(start, poly_coeffs=poly_coeffs)), start)
        end_point = (int(poly(end, poly_coeffs=poly_coeffs)), end)

        if dashed == False or i % 2 == 1:
            img = cv2.line(img, end_point, start_point, color, thickness)

    return img

blank_canvas = np.zeros((720, 1280))


def lane_poly(yval, poly_coeffs):
    return poly_coeffs[0]*yval**2 + poly_coeffs[1]*yval + poly_coeffs[2]


print("Left coeffs: ", left_coeffs)
print("Right fit: ", right_coeffs)
polyfit_left = draw_poly(blank_canvas, lane_poly, left_coeffs, 30)
polyfit_drawn = draw_poly(polyfit_left, lane_poly, right_coeffs, 30)
plt.imshow(polyfit_drawn, cmap="gray")
```

```
Left coeffs:  [ -1.59548390e-04   1.48929110e-01   3.83561816e+02]
Right fit:  [  2.60912380e-05  -1.79619055e-02   1.00603923e+03]
```

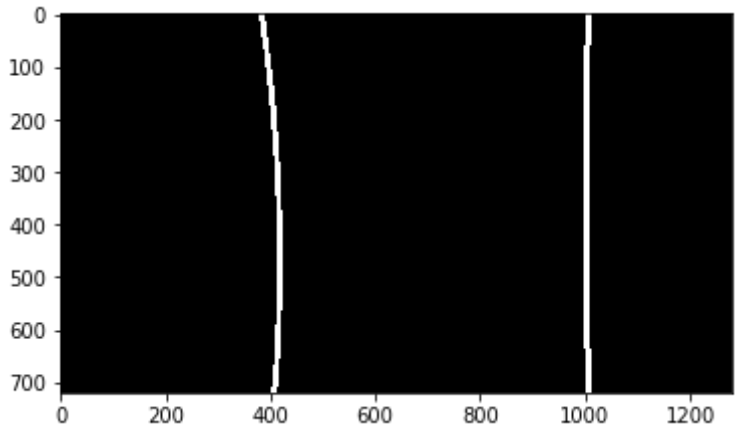
Out[9]: <matplotlib.image.AxesImage at 0x11ca99748>



In [10]:
```
print("Left coeffs: ", left_coeffs)
print("Right fit: ", right_coeffs)
polyfit_left = draw_poly(blank_canvas, lane_poly, left_coeffs, 30)
polyfit_drawn = draw_poly(polyfit_left, lane_poly, right_coeffs, 30)
plt.imshow(polyfit_drawn, cmap="gray")
```

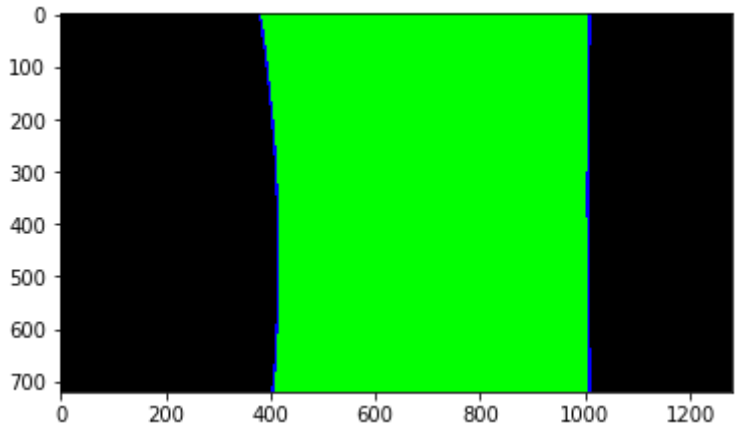
```
Left coeffs:  [ -1.59548390e-04   1.48929110e-01   3.83561816e+02]
Right fit:  [  2.60912380e-05  -1.79619055e-02   1.00603923e+03]
```

Out[10]: <matplotlib.image.AxesImage at 0x11caefe80>



In [11]:
```
color_canvas = cv2.cvtColor(blank_canvas.astype(np.uint8), cv2.COLOR_GRAY2RG

def highlight_lane_line_area(mask_template, left_poly, right_poly, start_y=0
    area_mask = mask_template
    for y in range(start_y, end_y):
        left = evaluate_poly(y, left_poly)
        right = evaluate_poly(y, right_poly)
        area_mask[y][int(left):int(right)] = 1

    return area_mask

def evaluate_poly(indep, poly_coeffs):
    return poly_coeffs[0]*indep**2 + poly_coeffs[1]*indep + poly_coeffs[2]

trace = color_canvas
trace[polyfit_drawn > 1] = [0,0,255]
area = highlight_lane_line_area(blank_canvas, left_coeffs, right_coeffs)
trace[area == 1] = [0,255,0]
plt.imshow(trace)
```

Out[11]: <matplotlib.image.AxesImage at 0x11e0844e0>




## Determine the curvature of the lane and vehicle position with respect to center.

In [12]:
```
def center(y, left_poly, right_poly):
    center = (1.5 * evaluate_poly(y, left_poly)
              - evaluate_poly(y, right_poly)) / 2
    return center

y_eval = 500
left_curverad = np.absolute(((1 + (2 * left_coeffs[0] * y_eval + left_coeffs
                /(2 * left_coeffs[0]))
right_curverad = np.absolute(((1 + (2 * right_coeffs[0] * y_eval + right_coe
                 /(2 * right_coeffs[0]))
print("Left lane curve radius: ", left_curverad, "pixels")
print("Right lane curve radius: ", right_curverad, "pixels")
curvature = (left_curverad + right_curverad) / 2
centre = center(719, left_coeffs, right_coeffs)
min_curvature = min(left_curverad, right_curverad)
```

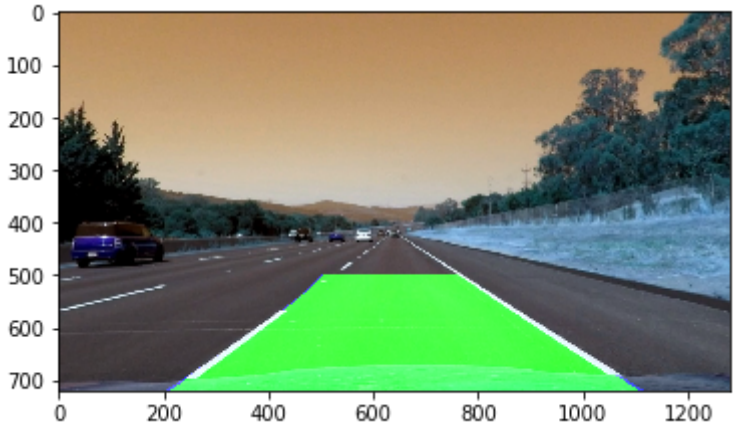
```
Left lane curve radius:  3134.37560023 pixels
Right lane curve radius:  19165.4211523 pixels
```

The above block shows the code that I used to calculate the radius of curvature of the lane and the position of the vehicle with respect to center.

## Warp the detected lane boundaries back onto the original image.

In [13]:
```
lane_lines = cv2.warpPerspective(trace, Minv, (imshape[1], imshape[0]), flag

combined_img = cv2.add(lane_lines, image)
plt.imshow(combined_img)
```

Out[13]: <matplotlib.image.AxesImage at 0x11e4baf28>

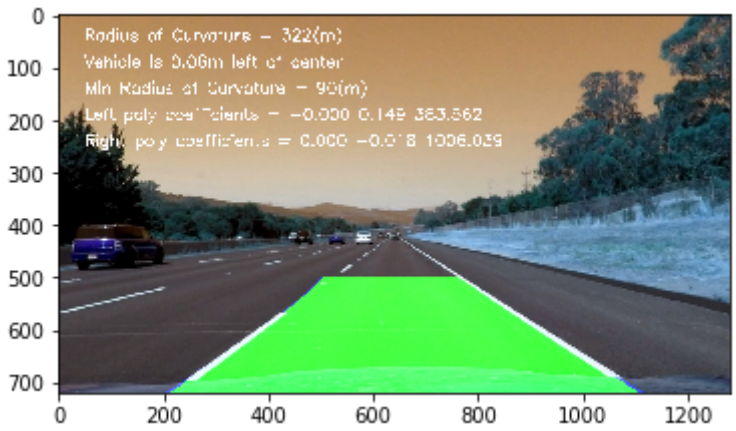



The image above is an example image of my result plotted back down onto the road, showing that the lane area is identified clearly.

## Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

In [14]:
```
def add_figures_to_image(img, curvature, vehicle_position, min_curvature, le
    vehicle_position = vehicle_position / 12800 * 3.7
    curvature = curvature / 128 * 3.7
    min_curvature = min_curvature / 128 * 3.7

    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(img, 'Radius of Curvature = %d(m)' % curvature, (50, 50), fo
    left_or_right = "left" if vehicle_position < 0 else "right"
    cv2.putText(img, 'Vehicle is %.2fm %s of center' % (np.abs(vehicle_posit
                (255, 255, 255), 2)
    cv2.putText(img, 'Min Radius of Curvature = %d(m)' % min_curvature, (50,
    cv2.putText(img, 'Left poly coefficients = %.3f %.3f %.3f' % (left_coeff
    cv2.putText(img, 'Right poly coefficients = %.3f %.3f %.3f' % (right_coe



add_figures_to_image(combined_img, curvature=curvature,
                     vehicle_position=centre,
                     min_curvature=min_curvature,
                     left_coeffs=left_coeffs,
                     right_coeffs=right_coeffs)
plt.imshow(combined_img)
```

Out[14]: <matplotlib.image.AxesImage at 0x11e5ceda0>



In [28]:
```
def image_pipeline(raw):
    global prev_left_coeffs
    global prev_right_coeffs

    imshape = raw.shape
    image = cv2.undistort(raw, mtx, dist, None, mtx)

    xgrad_thresh_temp = (40,100)
    s_thresh_temp=(150,255)

    have_fit = False
    while have_fit == False:
        combined_binary = apply_threshold_v2(image, xgrad_thresh=xgrad_thres

        warped = cv2.warpPerspective(combined_binary, M, (imshape[1], imshap

        vertices = np.array([[(0,imshape[0]),(550, 470), (700, 470), (imshap
        masked_image = region_of_interest(combined_binary, vertices)

        leftx, lefty, rightx, righty = histogram_pixels(warped, horizontal_c

        if len(leftx) > 1 and len(rightx) > 1:
            have_fit = True
        xgrad_thresh_temp = (xgrad_thresh_temp[0] - 2, xgrad_thresh_temp[1]
        s_thresh_temp = (s_thresh_temp[0] - 2, s_thresh_temp[1] + 2)

    left_fit, left_coeffs = fit_second_order_poly(lefty, leftx, return_coeff
    right_fit, right_coeffs = fit_second_order_poly(righty, rightx, return_c


    y_eval = 500
    left_curverad = np.absolute(((1 + (2 * left_coeffs[0] * y_eval + left_co
                /(2 * left_coeffs[0]))
    right_curverad = np.absolute(((1 + (2 * right_coeffs[0] * y_eval + right
                 /(2 * right_coeffs[0]))

    curvature = (left_curverad + right_curverad) / 2
    centre = center(719, left_coeffs, right_coeffs)
    min_curvature = min(left_curverad, right_curverad)

    blank_canvas = np.zeros((720, 1280))
    polyfit_left = draw_poly(blank_canvas, lane_poly, left_coeffs, 30)
    polyfit_drawn = draw_poly(polyfit_left, lane_poly, right_coeffs, 30)

    trace = cv2.cvtColor(blank_canvas.astype(np.uint8), cv2.COLOR_GRAY2RGB)
    trace[polyfit_drawn > 1] = [0,0,255]
    area = highlight_lane_line_area(blank_canvas, left_coeffs, right_coeffs)
    trace[area == 1] = [0,255,0]

    lane_lines = cv2.warpPerspective(trace, Minv, (imshape[1], imshape[0]),

    combined_img = cv2.add(lane_lines, image)

    y_eval = 500
    left_curverad = np.absolute(((1 + (2 * left_coeffs[0] * y_eval + left_co
                /(2 * left_coeffs[0]))
```

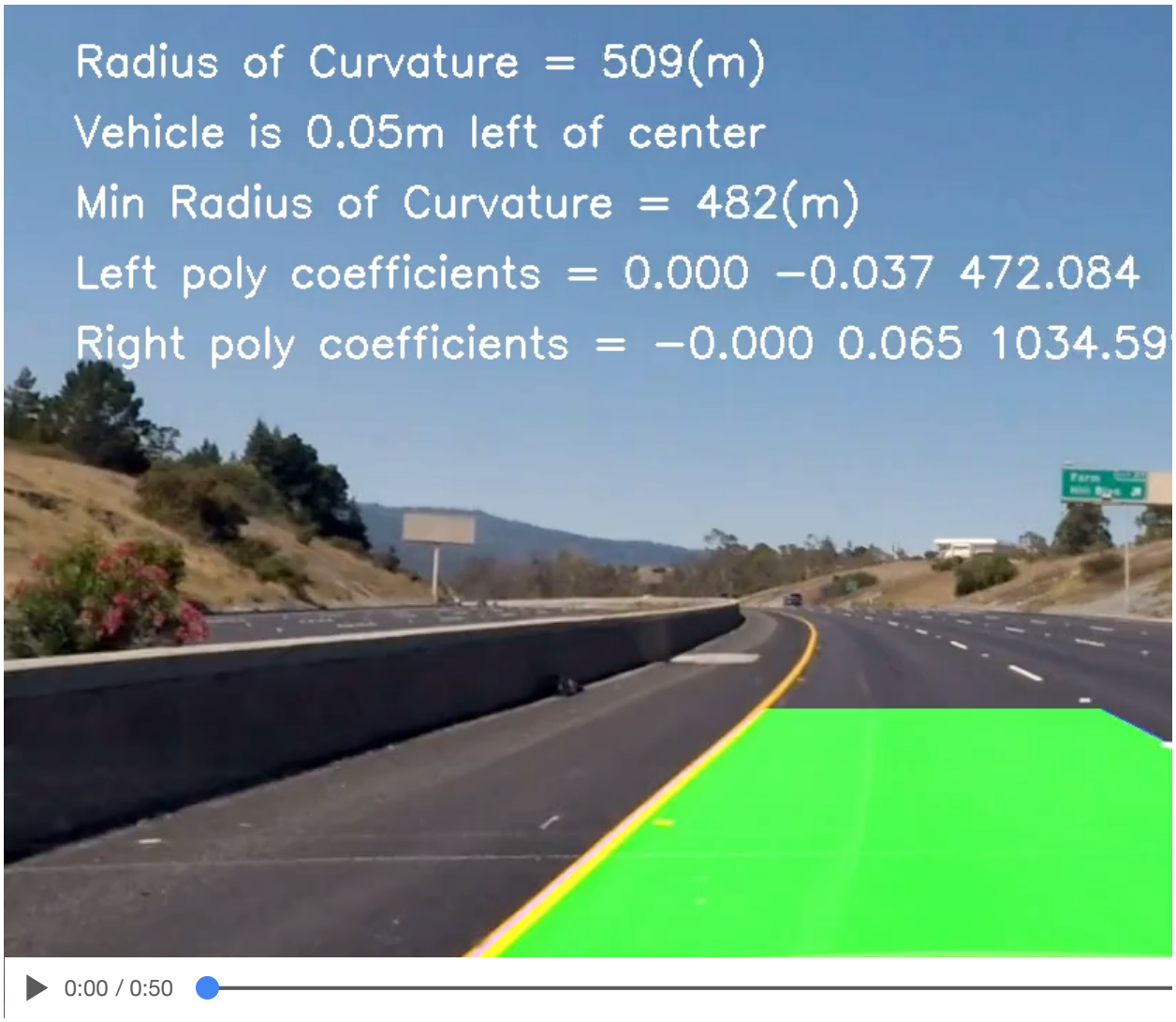
```
    right_curverad = np.absolute(((1 + (2 * right_coeffs[0] * y_eval + right
                 /(2 * right_coeffs[0]))
    curvature = (left_curverad + right_curverad) / 2
    centre = center(719, left_coeffs, right_coeffs)
    min_curvature = min(left_curverad, right_curverad)

    add_figures_to_image(combined_img, curvature=curvature,
                         vehicle_position=centre,
                         min_curvature=min_curvature,
                         left_coeffs=left_coeffs,
                         right_coeffs=right_coeffs)


    return combined_img
```

In [29]:
```
from moviepy.editor import VideoFileClip
from IPython.display import HTML

output = 'project_output_color.mp4'
clip1 = VideoFileClip("project_video.mp4")
output_clip = clip1.fl_image(image_pipeline)
%time output_clip.write_videofile(output, audio=False)
```

```
left_x:  [array([280, 283, 296, ..., 436, 437, 438]), array([267, 296, 29
7, ..., 444, 444, 444]), array([305, 305, 306, ..., 444, 444, 444]), arra
y([397, 397, 397, ..., 437, 437, 437]), array([403, 403, 403, ..., 447, 4
47, 447])]
right_x:  []
Init no peaks for left or right
left_x:  [array([279, 280, 283, ..., 437, 438, 439]), array([267, 296, 29
7, ..., 444, 444, 444]), array([285, 286, 286, ..., 444, 444, 444]), arra
y([397, 397, 397, ..., 437, 437, 437]), array([403, 403, 403, ..., 447, 4
47, 447])]
right_x:  []
Init no peaks for left or right
left_x:  [array([279, 280, 283, ..., 439, 441, 442]), array([267, 273, 27
3, ..., 444, 444, 444]), array([285, 286, 286, ..., 444, 444, 444]), arra
y([287, 287, 397, ..., 437, 437, 437]), array([402, 402, 402, ..., 447, 4
47, 447])]
right_x:  []
 49%|████▉      | 624/1261 [06:15<16:44,  1.58s/it]
```

In [17]:
```
HTML("""
<video width="960" height="540" controls>
  <source src="{0}">
</video>
""".format(output))
```

Out[17]:




## ## Discussion

The main issue I faced was to fit the polynomial correctly. Incorrect result could come from a few different reasons:
1. Thresholding. I tried different threshold values to find the best result. However, the parameter may be good in one frame but not good for another frame. So a potential improvement is to make the thresholding adaptive automatically. In my implementation, I relax the threshold a little bit when there's no line detected.
2. Sometimes even when the lane line is straight, the polynomial is still curvy. The reason can still be traced back to the thresholding. When the thresholding does not output a clean lane line pixels. This can happen. Further reducing noice could improve the result.