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.

---

1. Compute the camera calibration using chessboard images

In [10]:
```
import numpy as np
import cv2
import glob
import matplotlib.pyplot as plt
import os
import pickle
import matplotlib

%matplotlib inline
```

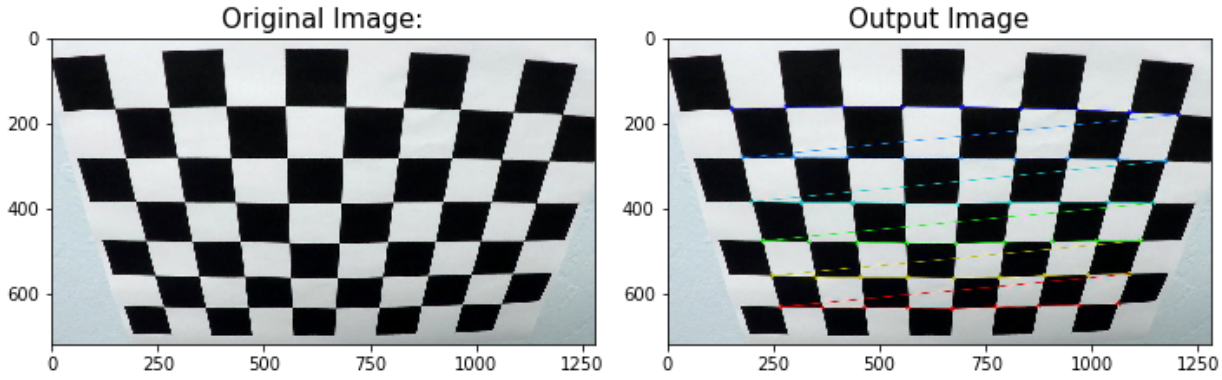
In [11]:
```
def calibration(IMG_PATH, chessboard_shape):
    # Arrays to store object points and image points from all the images.
    objpoints = [] # 3d points in real world space
    imgpoints = [] # 2d points in image plane.

    # Make a list of calibration images
    images = glob.glob(IMG_PATH)

    # prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
    objp = np.zeros((chessboard_shape[1] * chessboard_shape[0], 3), np.floa
    objp[:,:2] = np.mgrid[0:chessboard_shape[0], 0:chessboard_shape[1]].T.r

    # Step through the list and search for chessboard corners
    for fname in images:
        img = cv2.imread(fname)
        gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

        # Find the chessboard corners
        ret, corners = cv2.findChessboardCorners(gray, chessboard_shape, No

        # If found, add object points, image points
        if ret == True:
            objpoints.append(objp)
            imgpoints.append(corners)

        # Save images with drawing corners
        new_fname = fname.split('/')[-1]
        img = cv2.drawChessboardCorners(img, chessboard_shape, corners, ret
        SAVE_PATH = 'output_images/calibration/'
        cv2.imwrite(filename=SAVE_PATH + '/' + new_fname, img=img)

    return cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None
```

In [12]:
```
CAL_IMG_PATH = './camera_cal/calibration*.jpg'
chessboard_shape = (9, 6)

ret, mtx, dist, rvecs, tvecs = calibration(CAL_IMG_PATH, chessboard_shape)
with open('output_images/calibration/calibration.p', 'wb') as file:
    pickle.dump((ret, mtx, dist, rvecs, tvecs), file)
```

Display output example

In [13]:
```
def display(ori_img, new_img, new_title, fname, gray=False):
    f, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
    f.tight_layout()
    matplotlib.rc('xtick', labelsize=15)
    matplotlib.rc('ytick', labelsize=15)
    ax1.imshow(cv2.cvtColor(ori_img, cv2.COLOR_BGR2RGB))
    ax1.set_title('Original Image:', fontsize=15)
    if gray:
        ax2.imshow(new_img, cmap='gray')
    else:
        ax2.imshow(cv2.cvtColor(new_img, cv2.COLOR_BGR2RGB), cmap='gray')
    ax2.set_title(new_title, fontsize=15)
    f.savefig('output_images/comparison/' + fname)
    plt.show()
```

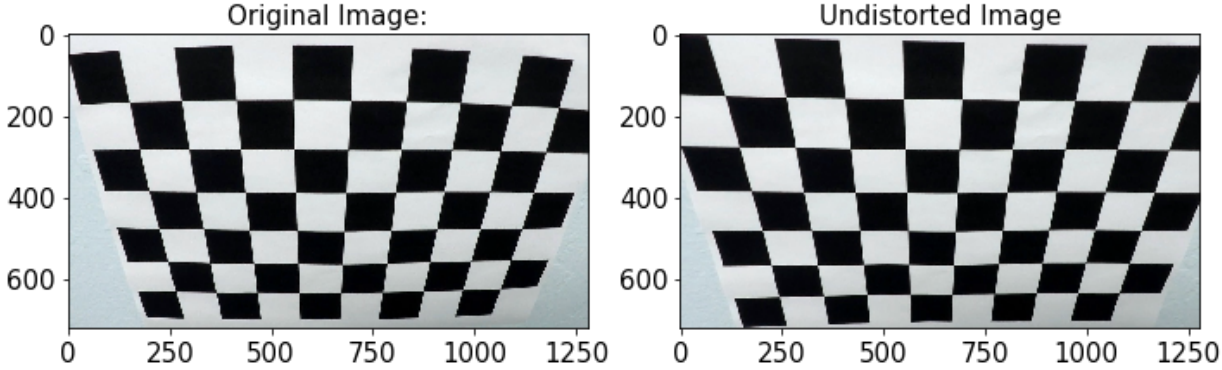
In [14]:
```
test_chess_img = plt.imread(os.path.join('camera_cal', 'calibration2.jpg'))
cal_img = plt.imread(os.path.join('output_images/calibration/', 'calibratic
display(test_chess_img, cal_img, 'Output Image', 'chees_out')
```



2. Apply a distortion correction to raw images.

In [15]:
```
def undistort(img, mtx, dist):
    return cv2.undistort(img, mtx, dist, None, mtx)
```

In [16]:
```
undistorted_chess_img = undistort(test_chess_img, mtx, dist)

display(test_chess_img, undistorted_chess_img, 'Undistorted Image', 'undist
```



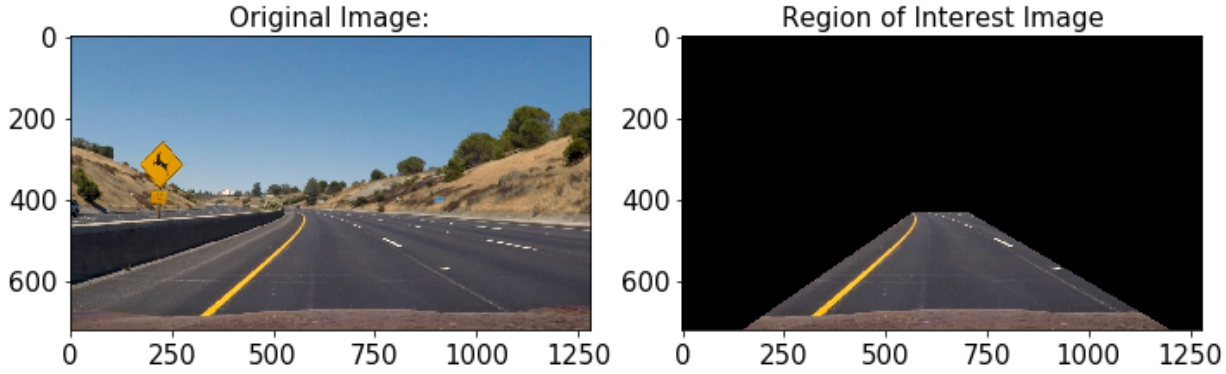
Utility function from Project 1, find the region that we are interested.

In [17]:
```
def region_of_interest(img, vertices):
    """
    Applies an image mask.

    Only keeps the region of the image defined by the polygon
    formed from `vertices`. The rest of the image is set to black.
    `vertices` should be a numpy array of integer points.
    """
    #defining a blank mask to start with
    mask = np.zeros_like(img)

    #defining a 3 channel or 1 channel color to fill the mask with dependin
    if len(img.shape) > 2:
        channel_count = img.shape[2]  # i.e. 3 or 4 depending on your image
        ignore_mask_color = (255,) * channel_count
    else:
        ignore_mask_color = 255

    #filling pixels inside the polygon defined by "vertices" with the fill
    cv2.fillPoly(mask, vertices, ignore_mask_color)

    #returning the image only where mask pixels are nonzero
    masked_image = cv2.bitwise_and(img, mask)
    return masked_image
```

In [18]:
```
# Make a list of test images
test_images = glob.glob('./test_images/*.jpg')
ROAD_SAVE_PATH = 'output_images/road/'
UNDISTORTED_SAVE_PATH = 'output_images/undistorted/'
UNDISTORTED_FULL_SAVE_PATH = 'output_images/undistorted/full_image/'

imshape = cv2.imread(test_images[0]).shape

vertices = np.array([[(150, imshape[0]), (570, 430), (700, 430),
                      (imshape[1]-80, imshape[0])]], dtype=np.int32)

undistored_list = []
for fname in test_images:
    img = cv2.imread(fname)
    new_fname = fname.split('/')[-1]
    undist = undistort(img, mtx, dist)
    cv2.imwrite(filename=UNDISTORTED_FULL_SAVE_PATH + '/' + new_fname, img=
    img = region_of_interest(img, vertices)
    cv2.imwrite(filename=ROAD_SAVE_PATH + '/' + new_fname, img=img)
    img = undistort(img, mtx, dist)
    cv2.imwrite(filename=UNDISTORTED_SAVE_PATH + '/' + new_fname, img=img)
```

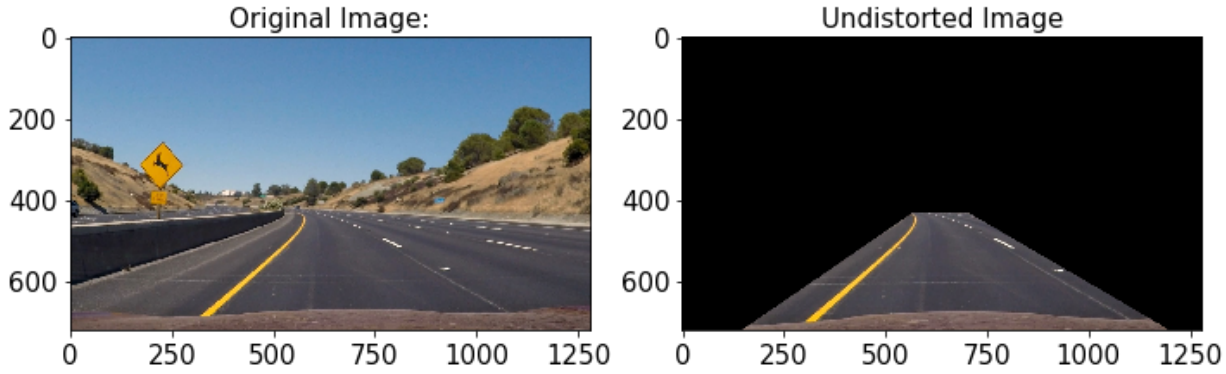
Display region of interest image example

In [19]:
```
test_img = cv2.imread('test_images/test2.jpg')
region_img = region_of_interest(test_img, vertices)

display(test_img, region_img, 'Region of Interest Image', 'region_of_int')
```



Display undistorted image example

In [20]:
```
undistorted_img = undistort(region_img, mtx, dist)

display(test_img, undistorted_img, 'Undistorted Image', 'undistorted_img')
```

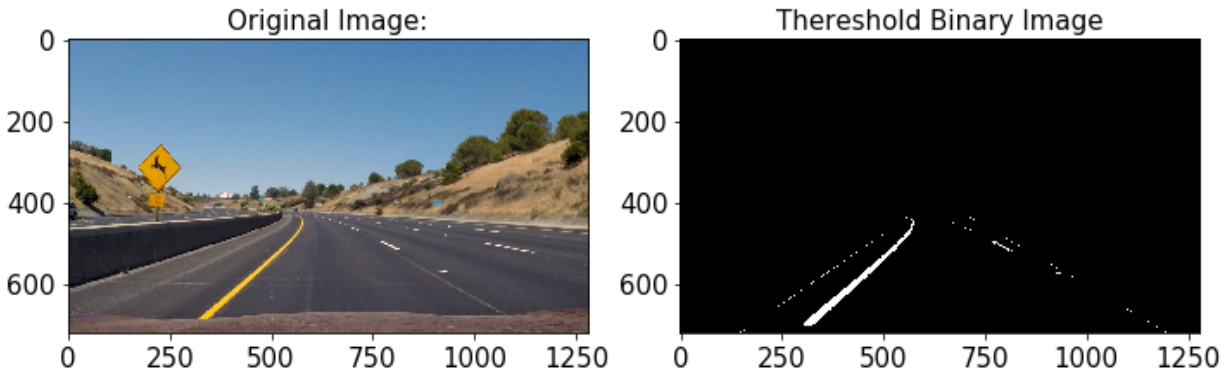


3. Use color transforms, gradients, etc., to create a thresholded binary image.

In [21]:
```
def threshold_binary(img, s_thresh=(170, 255), sx_thresh=(20, 100)):
    img = np.copy(img)
    hls = cv2.cvtColor(img, cv2.COLOR_RGB2HLS)
    l_channel = hls[:,:,1]
    s_channel = hls[:,:,2]

    sobelx =  cv2.Sobel(l_channel, 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 >= sx_thresh[0]) & (scaled_sobel <= sx_thresh[1]

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

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

    return combined_binary
```

In [22]:
```
undistored_images = glob.glob('./output_images/undistorted/*.jpg')
THRESHOLD_SAVE_PATH = 'output_images/threshold_binary/'

threshold_binary_list = []
for fname in undistored_images:
    img = cv2.imread(fname)
    new_fname = fname.split('/')[-1]
    combined_binary = threshold_binary(img,
                                       s_thresh=(100, 255),
                                       sx_thresh=(80, 110))
    threshold_binary_list.append(combined_binary)
    cv2.imwrite(filename=THRESHOLD_SAVE_PATH + '/' + new_fname, img=combine
```

In [33]:
```
threshold_binary_img = threshold_binary(undistorted_img,
                                        s_thresh=(100, 255),
                                        sx_thresh=(80, 110))


display(test_img, threshold_binary_img, 'Thereshold Binary Image', 'thresho
```



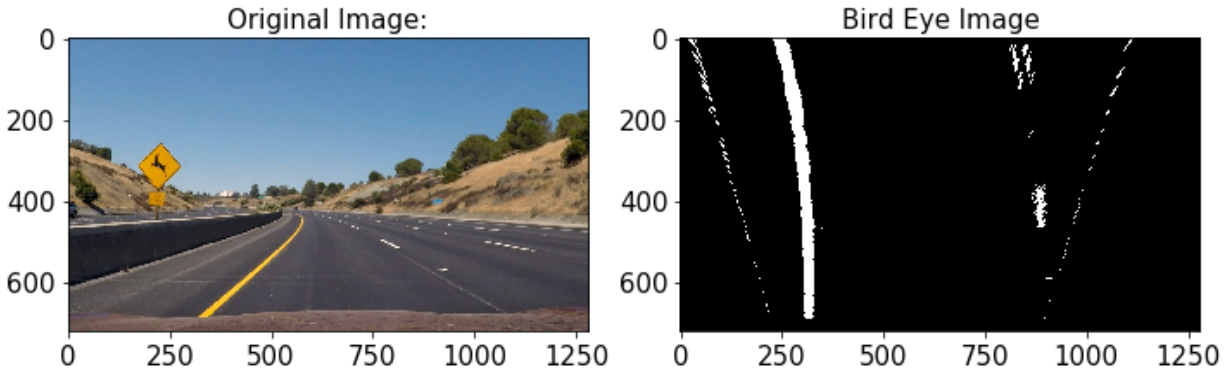
In [ ]:
```
```

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

In [34]:
```
def perspective_transform(img_binary, src, dst):
    """convert the perspective image to bird view"""
    image_shape = (img_binary.shape[1], img_binary.shape[0])
    M = cv2.getPerspectiveTransform(src, dst)
    M_T = cv2.getPerspectiveTransform(dst, src)
    warped = cv2.warpPerspective(img_binary, M, image_shape, flags=cv2.INTE
    return warped, M, M_T
```

In [35]:
```
src = np.float32([[258, 679], [446, 549], [837, 549], [1045, 679]]) # persp
dst = np.float32([[258, 679], [258, 549], [837, 549], [837, 679]]) # perspe


threshold_binary_images = glob.glob('./output_images/threshold_binary/*.jpg
PERSPECTIVE_SAVE_PATH = 'output_images/perspective_transform/'

warped_list = []
for fname in threshold_binary_images:
    img = cv2.imread(fname)
    new_fname = fname.split('/')[-1]
    warped, M, M_T = perspective_transform(img, src, dst)
    warped_list.append(warped)

    cv2.imwrite(filename=PERSPECTIVE_SAVE_PATH + '/' + new_fname, img=warpe
```

In [32]:
```
warped_img, M, M_T = perspective_transform(threshold_binary_img, src, dst)
display(test_img, warped_img, 'Bird Eye Image', 'test', gray=True)
```



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

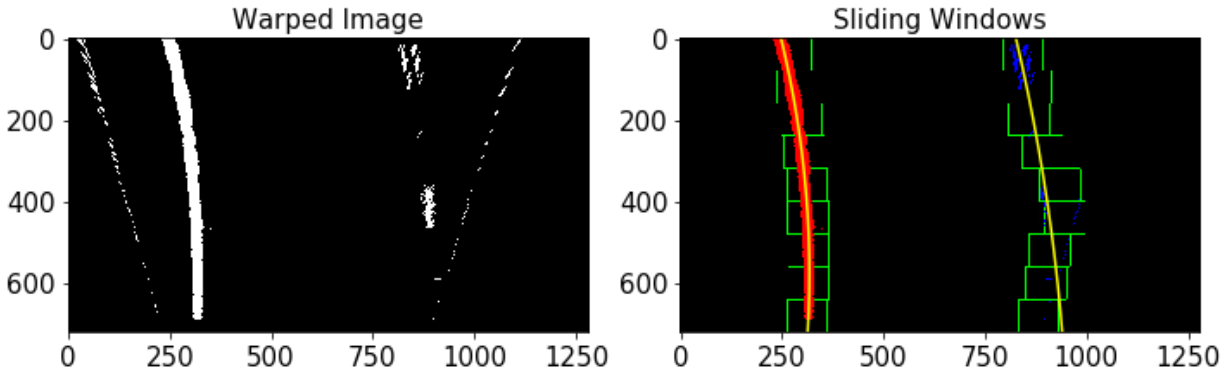
In [36]:
```
# Define a class to receive the characteristics of each line detection
class Line():
    def __init__(self):
        # was the line detected in the last iteration?
        self.detected = False
        # x values of the last n fits of the line
        self.recent_xfitted = []
        #average x values of the fitted line over the last n iterations
        self.bestx = None
        #polynomial coefficients averaged over the last n iterations
        self.best_fit = None
        #polynomial coefficients for the most recent fit
        self.current_fit = [np.array([False])]
        #radius of curvature of the line in some units
        self.radius_of_curvature = None
        #distance in meters of vehicle center from the line
        self.line_base_pos = None
        #difference in fit coefficients between last and new fits
        self.diffs = np.array([0,0,0], dtype='float')
        #x values for detected line pixels
        self.allx = None
        #y values for detected line pixels
        self.ally = None
```

In [37]:
```
LEFT = Line()
RIGHT = Line()

def find_lane_pixels(binary_warped):
    # Take a histogram of the bottom half of the image
    histogram = np.sum(binary_warped[binary_warped.shape[0]//2:,:], axis=0)
    # Create an output image to draw on and visualize the result
    out_img = np.dstack((binary_warped, binary_warped, binary_warped))
    # Find the peak of the left and right halves of the histogram
    # These will be the starting point for the left and right lines
    midpoint = np.int(histogram.shape[0]//2)
    leftx_base = np.argmax(histogram[:midpoint])
    rightx_base = np.argmax(histogram[midpoint:]) + midpoint

    # HYPERPARAMETERS
    # Choose the number of sliding windows
    nwindows = 9
    # Set the width of the windows +/- margin
    margin = 50
    # Set minimum number of pixels found to recenter window
    minpix = 1

    # Set height of windows - based on nwindows above and image shape
    window_height = np.int(binary_warped.shape[0]//nwindows)
    # Identify the x and y positions of all nonzero pixels in the image
    nonzero = binary_warped.nonzero()
    nonzeroy = np.array(nonzero[0])
    nonzerox = np.array(nonzero[1])
    # Current positions to be updated later for each window in nwindows
    leftx_current = leftx_base
    rightx_current = rightx_base

    # Create empty lists to receive left and right lane pixel indices
    left_lane_inds = []
    right_lane_inds = []

    # Step through the windows one by one
    for window in range(nwindows):
        # Identify window boundaries in x and y (and right and left)
        win_y_low = binary_warped.shape[0] - (window+1)*window_height
        win_y_high = binary_warped.shape[0] - window*window_height
        ### TO-DO: Find the four below boundaries of the window ###
        win_xleft_low = leftx_current - margin   # Update this
        win_xleft_high = leftx_current + margin  # Update this
        win_xright_low = rightx_current - margin  # Update this
        win_xright_high = rightx_current + margin # Update this

        # Draw the windows on the visualization image
        cv2.rectangle(out_img,(win_xleft_low,win_y_low), (win_xleft_high,wi
        cv2.rectangle(out_img,(win_xright_low,win_y_low), (win_xright_high,

        ### TO-DO: Identify the nonzero pixels in x and y within the window
        good_left_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high)
        good_right_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high

        # Append these indices to the lists
```

```
        left_lane_inds.append(good_left_inds)
        right_lane_inds.append(good_right_inds)

        ### TO-DO: If you found > minpix pixels, recenter next window ###
        ### (`right` or `leftx_current`) on their mean position ###
        # If you found > minpix pixels, recenter next window on their mean
        if len(good_left_inds) > minpix:
            leftx_current = np.int(np.mean(nonzerox[good_left_inds]))
        if len(good_right_inds) > minpix:
            rightx_current = np.int(np.mean(nonzerox[good_right_inds]))
    # Concatenate the arrays of indices (previously was a list of lists of
    try:
        left_lane_inds = np.concatenate(left_lane_inds)
        right_lane_inds = np.concatenate(right_lane_inds)
    except ValueError:
        # Avoids an error if the above is not implemented fully
        pass

    # Extract left and right line pixel positions
    leftx = nonzerox[left_lane_inds]
    lefty = nonzeroy[left_lane_inds]
    rightx = nonzerox[right_lane_inds]
    righty = nonzeroy[right_lane_inds]

    ### TO-DO: Fit a second order polynomial to each using `np.polyfit` ###
    left_fit = np.polyfit(lefty, leftx, 2)
    right_fit = np.polyfit(righty, rightx, 2)

    # Generate x and y values for plotting
    ploty = np.linspace(0, binary_warped.shape[0]-1, binary_warped.shape[0]
    left_fitx = left_fit[0]*ploty**2 + left_fit[1]*ploty + left_fit[2]
    right_fitx = right_fit[0]*ploty**2 + right_fit[1]*ploty + right_fit[2]

    ## Visualization ##
    # Colors in the left and right lane regions
    out_img[lefty, leftx] = [255, 0, 0]
    out_img[righty, rightx] = [0, 0, 255]

    #For Video:
    LEFT.recent_xfitted = [(left_fitx)]
    LEFT.bestx = np.mean(LEFT.recent_xfitted)
    RIGHT.recent_xfitted = [(right_fitx)]
    RIGHT.bestx = np.mean(RIGHT.recent_xfitted)

    LEFT.current_fit = [(left_fit)]
    LEFT.best_fit = np.mean(LEFT.current_fit)
    RIGHT.current_fit = [(right_fit)]
    RIGHT.best_fit = np.mean(RIGHT.current_fit)

    LEFT.allx = leftx
    LEFT.ally = lefty
    RIGHT.allx = rightx
    RIGHT.ally = righty

    return out_img, left_fitx, right_fitx, ploty, left_fit, right_fit, left
```

In [38]:
```
warped_images = glob.glob('./output_images/perspective_transform/*.jpg')
SLIDING_SAVE_PATH = 'output_images/sliding_windows/'

for fname in warped_images:
    img = cv2.imread(fname)
    new_fname = fname.split('/')[-1]
    out_img, left_fitx, right_fitx, ploty, left_fit, right_fit, leftx, righ
    plt.plot(left_fitx, ploty, color='yellow')
    plt.plot(right_fitx, ploty, color='yellow')
    plt.imshow(out_img)
    plt.savefig(SLIDING_SAVE_PATH + '/' + new_fname)
    plt.close()
```

In [39]:
```
out_img, left_fitx, right_fitx, ploty, left_fit, right_fit, leftx, rightx,

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
f.tight_layout()
matplotlib.rc('xtick', labelsize=15)
matplotlib.rc('ytick', labelsize=15)
ax1.imshow(warped_img, cmap='gray')
ax1.set_title('Warped Image', fontsize=15)

ax2.plot(left_fitx, ploty, color='yellow')
ax2.plot(right_fitx, ploty, color='yellow')
ax2.imshow(out_img, cmap='gray')
ax2.set_title('Sliding Windows', fontsize=15)

f.savefig('output_images/comparison/' + 'sliding')


plt.show()
```

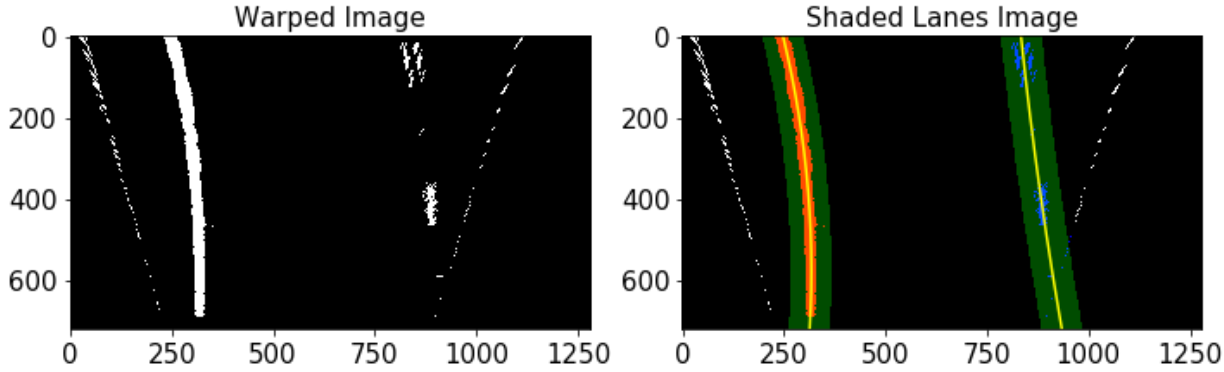


In [40]:
```
def fit_poly(img_shape, leftx, lefty, rightx, righty):
    ### TO-DO: Fit a second order polynomial to each with np.polyfit() ###
    left_fit = np.polyfit(lefty, leftx, 2)
    right_fit = np.polyfit(righty, rightx, 2)
    # Generate x and y values for plotting
    ploty = np.linspace(0, img_shape[0]-1, img_shape[0])
    ### TO-DO: Calc both polynomials using ploty, left_fit and right_fit ##
    left_fitx = left_fit[0] * ploty**2 + left_fit[1] * ploty + left_fit[2]
    right_fitx = right_fit[0] * ploty**2 + right_fit[1] * ploty + right_fit

    return left_fitx, right_fitx, ploty

def search_around_poly(binary_warped):
    # HYPERPARAMETER
    # Choose the width of the margin around the previous polynomial to sear
    # The quiz grader expects 100 here, but feel free to tune on your own!
    margin = 50

    # Grab activated pixels
    nonzero = binary_warped.nonzero()
    nonzeroy = np.array(nonzero[0])
    nonzerox = np.array(nonzero[1])

    # Find our lane pixels first
    out_img, left_fitx, right_fitx, ploty, left_fit, right_fit, leftx, righ

#     leftx, lefty, rightx, righty, out_img = find_lane_pixels(binary_warpe

    ### TO-DO: Fit a second order polynomial to each using `np.polyfit` ###
    left_fit = np.polyfit(lefty, leftx, 2)
    right_fit = np.polyfit(righty, rightx, 2)

    ### TO-DO: Set the area of search based on activated x-values ###
    ### within the +/- margin of our polynomial function ###
    ### Hint: consider the window areas for the similarly named variables #
    ### in the previous quiz, but change the windows to our new search area
    left_lane_inds = ((nonzerox > (left_fit[0]*(nonzeroy**2) + left_fit[1]*
                    + left_fit[2] - margin)) & (nonzerox < (left_fit[0]*(nc
                    + left_fit[1]*nonzeroy + left_fit[2] + margin)))
    right_lane_inds = ((nonzerox > (right_fit[0]*(nonzeroy**2) + right_fit[
                    + right_fit[2] - margin)) & (nonzerox < (right_fit[0]*(
                    + right_fit[1]*nonzeroy + right_fit[2] + margin)))

    # Again, extract left and right line pixel positions
    leftx = nonzerox[left_lane_inds]
    lefty = nonzeroy[left_lane_inds]
    rightx = nonzerox[right_lane_inds]
    righty = nonzeroy[right_lane_inds]

    # Fit new polynomials
    left_fitx, right_fitx, ploty = fit_poly(binary_warped.shape, leftx, lef

    ## Visualization ##
    # Create an image to draw on and an image to show the selection window
    out_img = np.dstack((binary_warped, binary_warped, binary_warped))*255
    window_img = np.zeros_like(out_img)
```

```
    # Color in left and right line pixels
    out_img[nonzeroy[left_lane_inds], nonzerox[left_lane_inds]] = [255, 0,
    out_img[nonzeroy[right_lane_inds], nonzerox[right_lane_inds]] = [0, 0,

    # Generate a polygon to illustrate the search window area
    # And recast the x and y points into usable format for cv2.fillPoly()
    left_line_window1 = np.array([np.transpose(np.vstack([left_fitx-margin,
    left_line_window2 = np.array([np.flipud(np.transpose(np.vstack([left_fi
                                ploty])))])
    left_line_pts = np.hstack((left_line_window1, left_line_window2))
    right_line_window1 = np.array([np.transpose(np.vstack([right_fitx-margi
    right_line_window2 = np.array([np.flipud(np.transpose(np.vstack([right_
                                ploty])))])
    right_line_pts = np.hstack((right_line_window1, right_line_window2))

    # Draw the lane onto the warped blank image
    cv2.fillPoly(window_img, np.int_([left_line_pts]), (0,255, 0))
    cv2.fillPoly(window_img, np.int_([right_line_pts]), (0,255, 0))
    result = cv2.addWeighted(out_img, 1, window_img, 0.3, 0)



    return result, left_fitx, right_fitx, ploty, left_fit, right_fit
```

In [41]:
```
warped_images = glob.glob('./output_images/perspective_transform/*.jpg')
SHADED_SAVE_PATH = 'output_images/shaded_lanes/'

for fname in warped_images:
    img = cv2.imread(fname)
    new_fname = fname.split('/')[-1]
    out_img, left_fitx, right_fitx, ploty, left_fit, right_fit = search_arc
    plt.plot(left_fitx, ploty, color='yellow')
    plt.plot(right_fitx, ploty, color='yellow')
    plt.imshow(out_img)
    plt.savefig(SHADED_SAVE_PATH + '/' + new_fname)
    plt.close()
```

In [42]:
```
out_img, left_fitx, right_fitx, ploty, left_fit, right_fit = search_around_

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
f.tight_layout()
matplotlib.rc('xtick', labelsize=15)
matplotlib.rc('ytick', labelsize=15)
ax1.imshow(warped_img, cmap='gray')
ax1.set_title('Warped Image', fontsize=15)

ax2.plot(left_fitx, ploty, color='yellow')
ax2.plot(right_fitx, ploty, color='yellow')
ax2.imshow(out_img, cmap='gray')
ax2.set_title('Shaded Lanes Image', fontsize=15)

f.savefig('output_images/comparison/' + 'shaded_lane')


plt.show()
```



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

In [43]:
```
def measure_curvature_and_offset(image_shape, ploty, left_fit, right_fit, l
    '''
    Calculates the curvature of polynomial functions in meters.
    '''
    y_eval =image_shape[1]

    # Define conversions in x and y from pixels space to meters
    ym_per_pix = 30/720 # meters per pixel in y dimension
    xm_per_pix = 3.7/700 # meters per pixel in x dimension
    # Start by generating our fake example data
    # Make sure to feed in your real data instead in your project!
    left = left_fit[0] * y_eval ** 2 + left_fit[1] * y_eval + left_fit[2]
    right = right_fit[0] * y_eval ** 2 + right_fit[1] * y_eval + right_fit[

    lane_midpoint_px = (right + left) / 2
    camera_midpoint_px = image_shape[0] / 2

    offset_from_center = np.abs(lane_midpoint_px - camera_midpoint_px) * xm

    leftx = np.array(leftx, dtype=np.float32)
    rightx = np.array(rightx, dtype=np.float32)

    left_fit_cr = np.polyfit(lefty * ym_per_pix, leftx * xm_per_pix, 2)
    right_fit_cr = np.polyfit(righty * ym_per_pix, rightx * xm_per_pix, 2)

    ##### TO-DO: Implement the calculation of R_curve (radius of curvature)
    left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix + left_fit_cr
    right_curverad = ((1 +  (2*right_fit_cr[0]*y_eval*ym_per_pix + right_fi
    curverad = np.min([left_curverad, right_curverad])

    LEFT.line_base_pos = offset_from_center
    RIGHT.line_base_pos = offset_from_center
    LEFT.radius_of_curvature = curverad
    RIGHT.radius_of_curvature = curverad

    return offset_from_center, curverad
```

In [44]:
```
out_img, left_fitx, right_fitx, ploty, left_fit, right_fit, leftx, rightx,

# Calculate the radius of curvature in meters for both lane lines
offset_from_center, curverad = measure_curvature_and_offset(out_img.shape,

print("In test2.jpg:")
print('Offset from center is {} m'.format(offset_from_center))
print('Radius of curvature is {} m'.format(curverad))
```

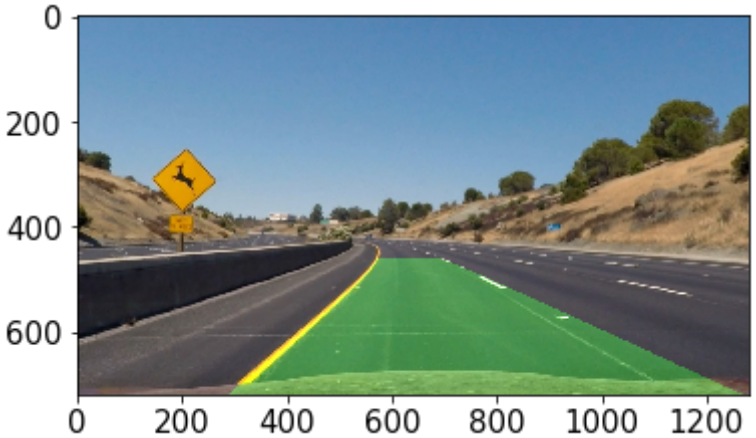
```
In test2.jpg:
Offset from center is 1.1809495135856067 m
Radius of curvature is 786.9896433736152 m
```

7. Warp the detected lane boundaries back onto the

original image.

In [47]:
```
def drawing(warped, Minv, ploty, left_fitx, right_fitx, src_img):
    # Create an image to draw the lines on
    warp_zero = np.zeros_like(warped).astype(np.uint8)
    color_warp = np.dstack((warp_zero, warp_zero, warp_zero))

    # Recast the x and y points into usable format for cv2.fillPoly()
    pts_left = np.array([np.transpose(np.vstack([left_fitx, ploty]))])
    pts_right = np.array([np.flipud(np.transpose(np.vstack([right_fitx, plo
    pts = np.hstack((pts_left, pts_right))

    # Draw the lane onto the warped blank image
    cv2.fillPoly(color_warp, np.int_([pts]), (0,255, 0))

    # Warp the blank back to original image space using inverse perspective
    newwarp = cv2.warpPerspective(color_warp, Minv, (warped.shape[1], warpe
    # Combine the result with the original image
    result = cv2.addWeighted(src_img, 1, newwarp, 0.3, 0)

    return result
```

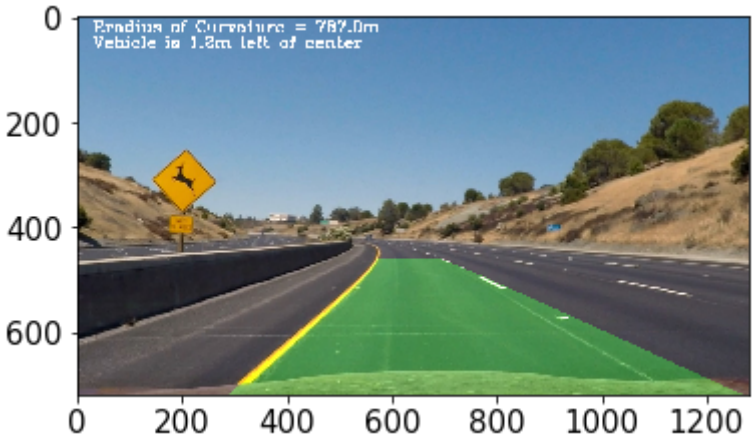
In [51]:
```
undistorted_full_img = undistort(test_img, mtx, dist)

drawing_img = drawing(warped_img,
                      M_T,
                      ploty,
                      left_fitx,
                      right_fitx,
                      undistorted_full_img)

plt.imshow(cv2.cvtColor(drawing_img, cv2.COLOR_BGR2RGB))
plt.show()
```



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

In [52]:
```
def annotate_numbers(image, offset_from_center, curverad):
    font = cv2.FONT_HERSHEY_COMPLEX
    cv2.putText(image, 'Rradius of Curvature = {:.1f}m'.format(curverad), (
                font, 1, (255,255,255), 2)
    cv2.putText(image, 'Vehicle is {:.1f}m left of center '.format(offset_f
                font, 1, (255,255,255), 2)
```

In [55]:
```
annotate_numbers(drawing_img, offset_from_center, curverad)

# plt.imshow(drawing_img)
plt.imshow(cv2.cvtColor(drawing_img, cv2.COLOR_BGR2RGB))

plt.savefig('output_images/comparison/' + 'drawing')
plt.show()
```



In [56]:
```
n_frames = 0

def video_pipeline(img):
    global n_frames
    # For the first frame
    if n_frames == 0:
        # Undistort original image
        undistorted = undistort(img, mtx, dist)
        # Create a thresholded binary image
        binary_thresholded = threshold_binary(undistorted)
        # Apply a perspective transform to rectify binary image
        warped, M, Minv = perspective_transform(binary_thresholded, src, ds
        # Detect lane pixels and fit to find the lane boundary
        windows_img, left_fitx, right_fitx, ploty, left_fit, right_fit, lef
        # Determine the curvature of the lane and vehicle position with res
        offset_from_center, curverad = measure_curvature_and_offset(img.sha
        # Draw shadow on the undistored image
        out_img = drawing(warped, Minv, ploty, left_fitx, right_fitx, undis
        # Annotate curvature and position on the image
        annotate_numbers(out_img, offset_from_center, curverad)
    else:
        # Undistort original image
        undistorted = undistort(img, mtx, dist)
        # Create a thresholded binary image
        binary_thresholded = threshold_binary(undistorted)
        # Only use the region of interest
        region = region_of_interest(binary_thresholded)
        # Apply a perspective transform to rectify binary image
        warped, M, Minv = perspective_transform(binary_thresholded, src, ds
        binary_warped = warped
        # If detect the lane, set parameter
        if LEFT.detected == True:
            left_fit = LEFT.current_fit
        # If didn't detect the lane, use previous best parameter
        else:
            left_fit = LEFT.best_fit
        # If detect the lane, set parameter
        if RIGHT.detected == True:
            right_fit = RIGHT.current_fit
        # If didn't detect the lane, use previous best parameter
        else:
            right_fit = RIGHT.best_fit
        # Detect lane pixels and fit to find the lane boundary
        windows_img, left_fitx, right_fitx, ploty, left_fit, right_fit, lef
        # If detect the lane, update Line() parameters
        if LEFT.detected == True:
            LEFT.recent_xfitted.append((left_fitx))
            LEFT.bestx = np.mean(LEFT.recent_xfitted)
            LEFT.current_fit.append((left_fit))
            LEFT.best_fit = np.mean(LEFT.current_fit)
            LEFT.allx = leftx
            LEFT.ally = lefty
        # If not, use the previous best parameter
        else:
            left_fitx = LEFT.bestx
            left_fit = LEFT.best_fit
```

```
            leftx = LEFT.allx
            lefty = LEFT.ally
        # If detect the lane, update Line() parameters
        if RIGHT.detected == True:
            RIGHT.recent_xfitted.append((right_fitx))
            RIGHT.bestx = np.mean(RIGHT.recent_xfitted)
            RIGHT.current_fit.append((right_fit))
            RIGHT.best_fit_coeffs = np.mean(RIGHT.current_fit)
            RIGHT.allx = rightx
            RIGHT.ally = righty
        # If not, use the previous best parameter
        else:
            right_fitx = RIGHT.bestx
            right_fit = RIGHT.best_fit
            rightx = RIGHT.allx
            righty = RIGHT.ally
        # Determine the curvature of the lane and vehicle position with res
        offset_from_center, curverad = measure_curvature_and_offset(img.sha
        # Draw shadow on the undistored image
        out_img = drawing(warped, Minv, ploty, left_fitx, right_fitx, undis
        # Annotate curvature and position on the image
        annotate_numbers(out_img, offset_from_center, curverad)

    return out_img
```

In [57]:
```
test_img = cv2.imread('test_images/test2.jpg')
out_img = video_pipeline(test_img)

display(test_img, out_img, 'Test Output Image', 'video_test')
```

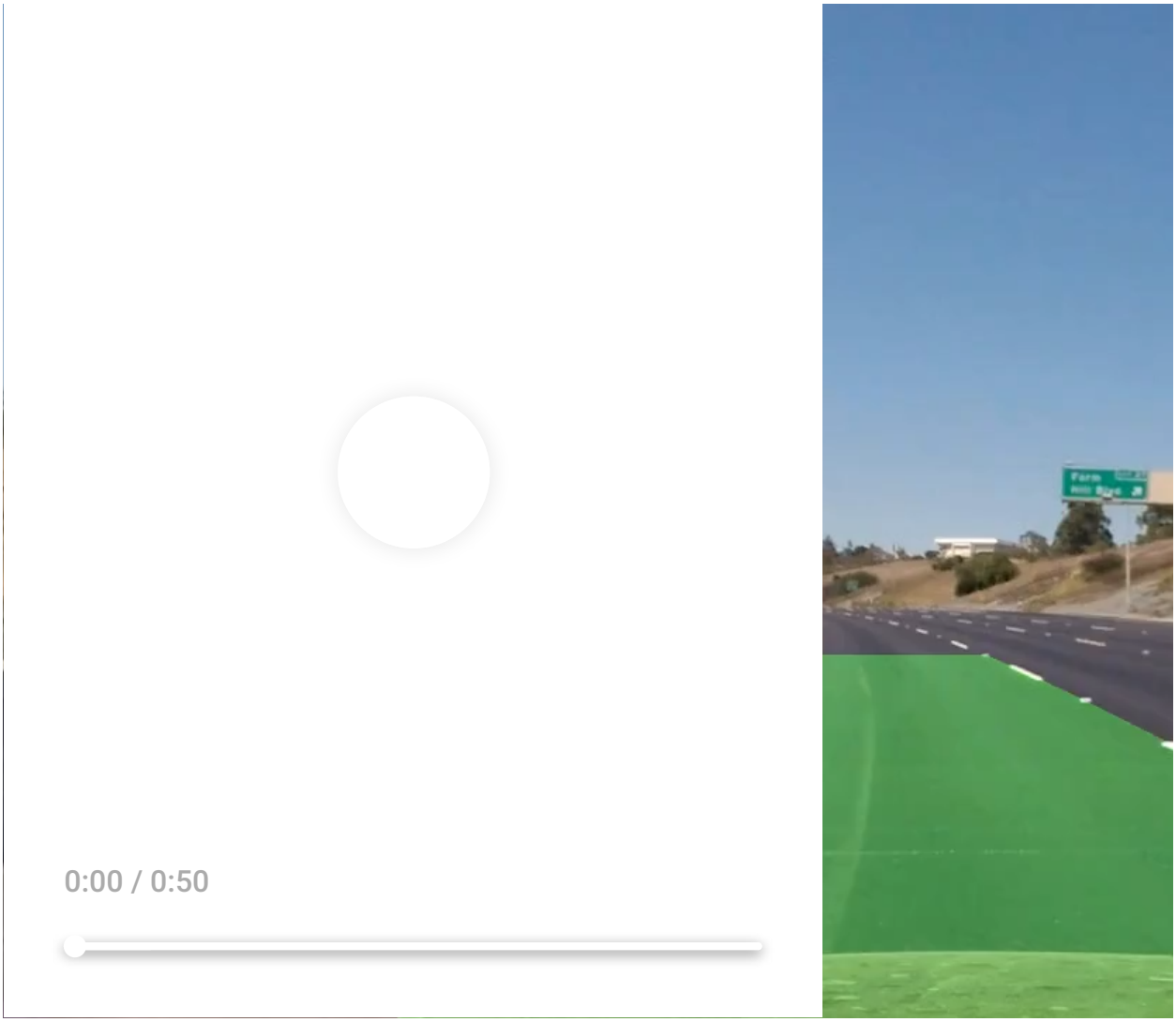
Run the pipeline on the project video and challenge video

In [58]:
```
from moviepy.editor import VideoFileClip

video_output = 'video_output/project_video_output.mp4'
clip = VideoFileClip('project_video.mp4')

output_clip = clip.fl_image(video_pipeline)
%time output_clip.write_videofile(video_output, audio=False, verbose=False,
```

```
CPU times: user 8min 54s, sys: 42.8 s, total: 9min 37s
Wall time: 1min 43s
```

In [59]:
```
from IPython.display import HTML

HTML("""
<video width="960" height="540" controls>
  <source src="{0}">
</video>
""".format(video_output))
```

Out[59]:

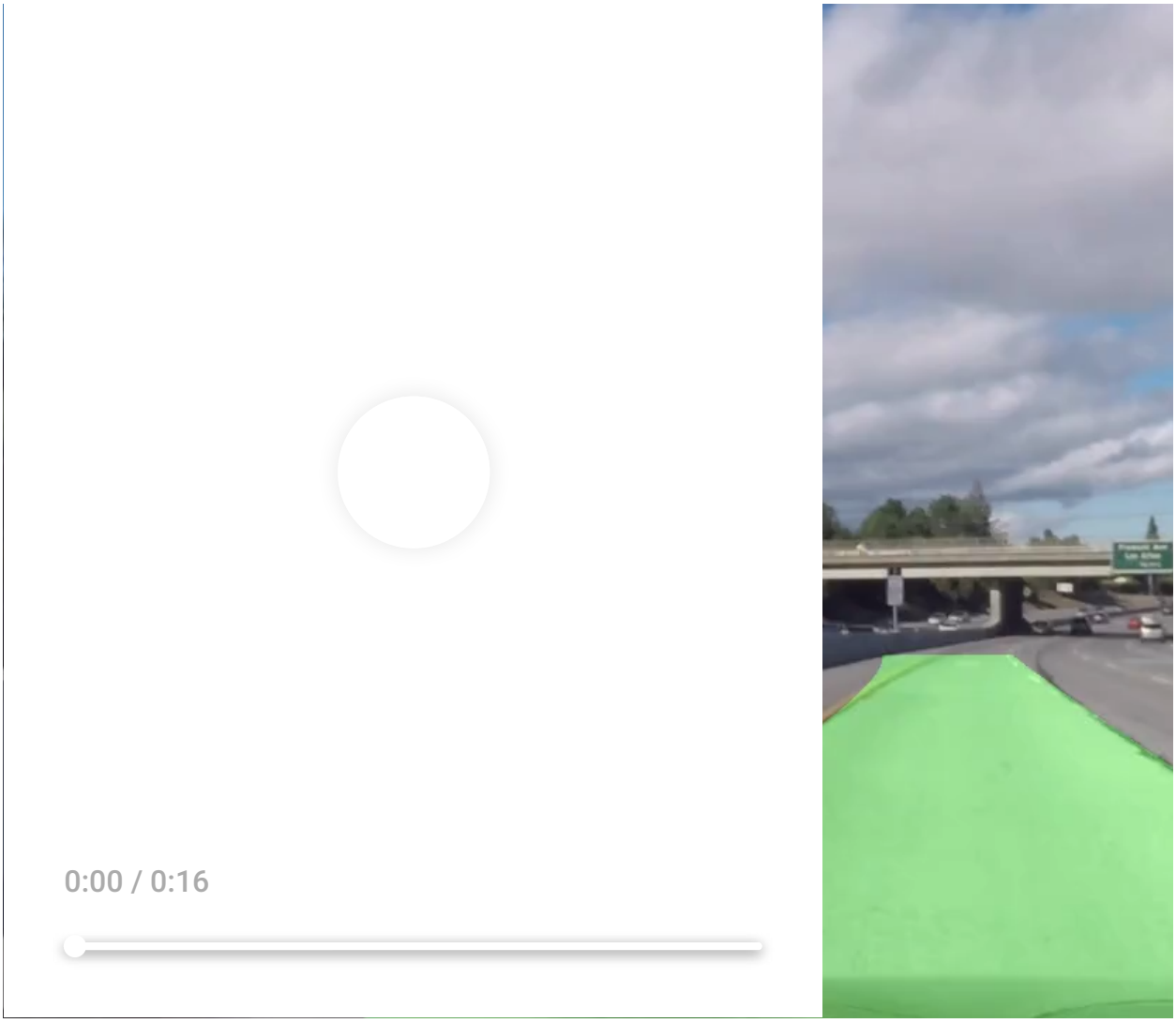


In [60]:
```
challenge_video_output = 'video_output/challenge_video_output.mp4'
challenge_clip = VideoFileClip('challenge_video.mp4')

challenge_output_clip = challenge_clip.fl_image(video_pipeline)
%time challenge_output_clip.write_videofile(challenge_video_output, audio=F
```

```
CPU times: user 2min 35s, sys: 10.2 s, total: 2min 45s
Wall time: 28.3 s
```

In [61]:
```
HTML("""
<video width="960" height="540" controls>
  <source src="{0}">
</video>
""".format(challenge_video_output))
```

Out[61]:

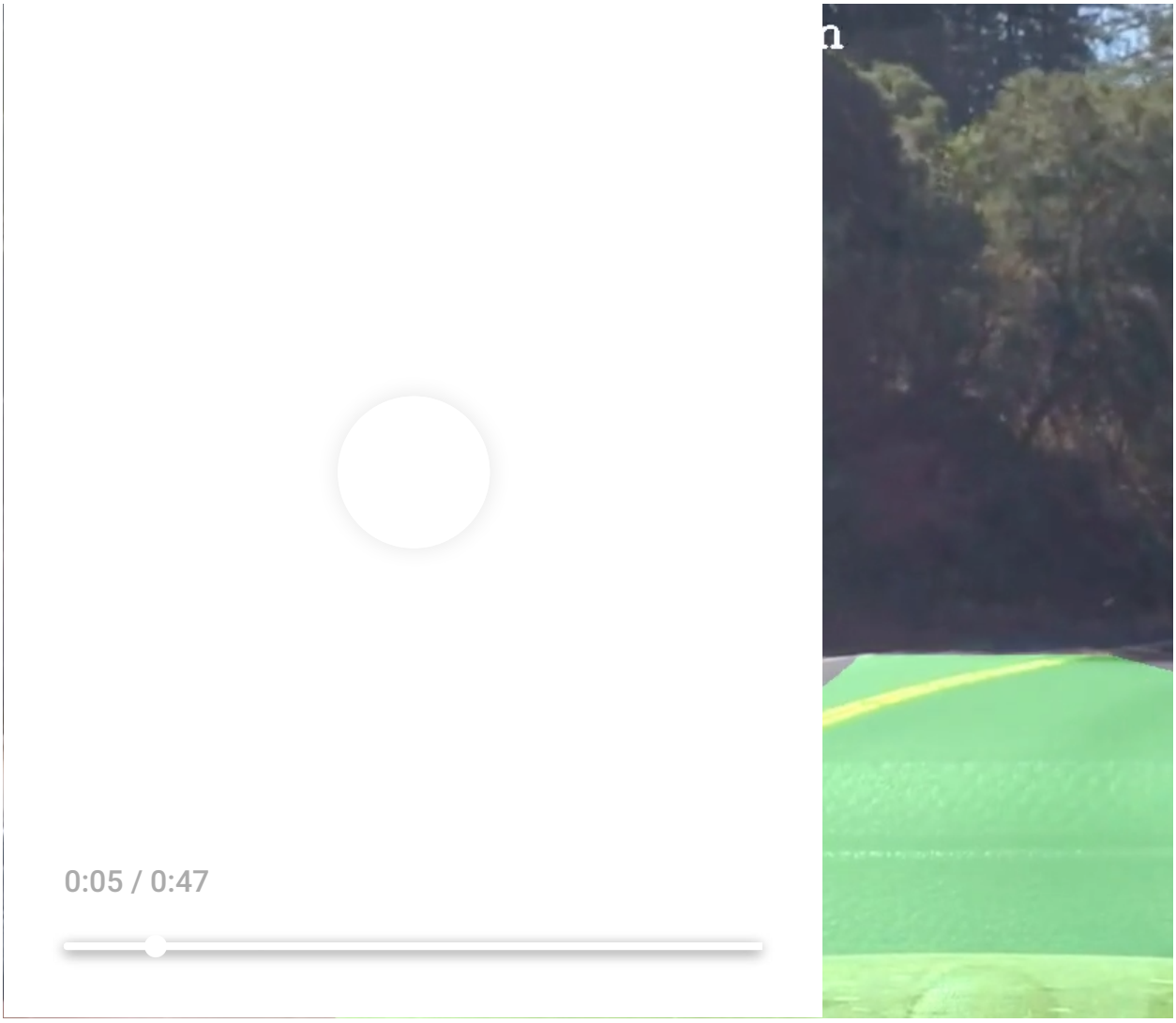


In [62]:
```
harder_challenge_video_output = 'video_output/harder_challenge_video_output
harder_challenge_clip = VideoFileClip('harder_challenge_video.mp4')

harder_challenge_output_clip = harder_challenge_clip.fl_image(video_pipelin
%time harder_challenge_output_clip.write_videofile(harder_challenge_video_c
```

```
CPU times: user 9min 5s, sys: 36.5 s, total: 9min 42s
Wall time: 1min 43s
```

In [63]:
```
HTML("""
<video width="960" height="540" controls>
  <source src="{0}">
</video>
""".format(harder_challenge_video_output))
```

Out[63]:



In [ ]: