MATLAB 3D Model Renderer

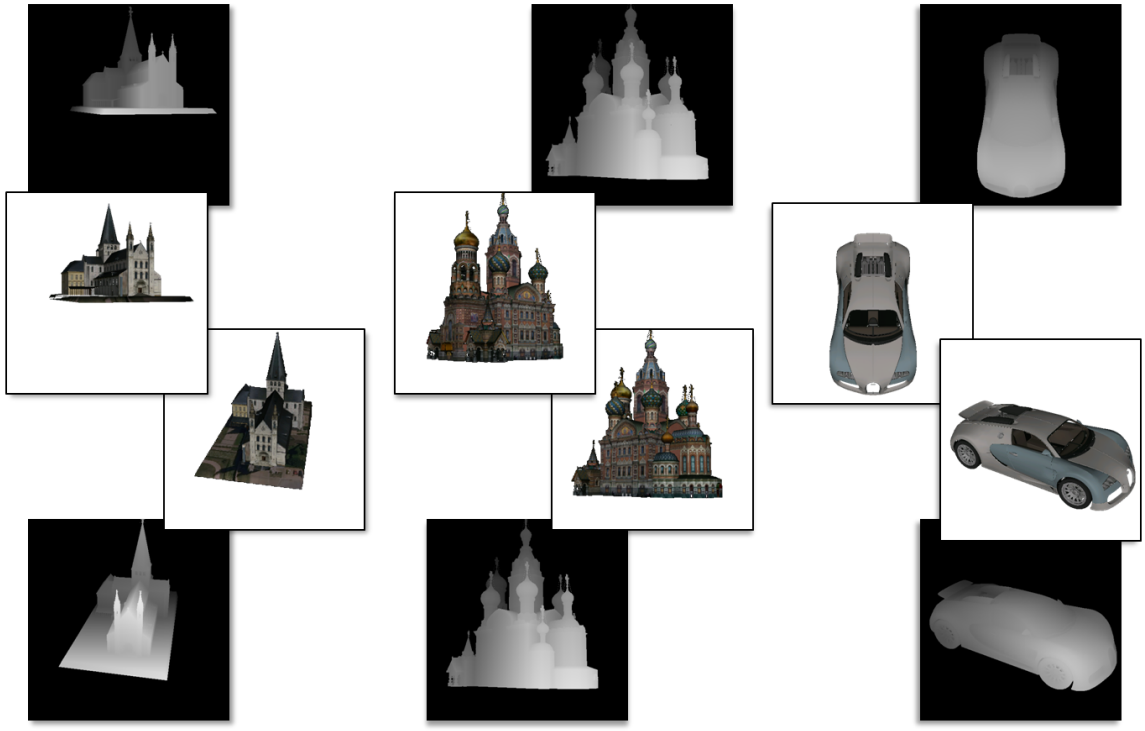


Figure 1. Examples of 3D models from the google 3D warehouse, rendered in MATLAB along with their (inverse) depth-maps, using the MATLAB 3D renderer.

What is this?

MATLAB function for rendering textured 3D models and computing object 6dof pose

MATLAB functions which use OpenSceneGraph and OpenCV for the following:

- Reading standard 3D CG file formats into MATLAB (.wrl, .obj, .3ds, etc.)
- Rendering with camera control using sphere orientation (elevation , azimuth, and yaw) or camera matrix
- Results rendered off-screen and returned as MATLAB matrices. It is therefore suitable for console versions of MATLAB, as well as batch processing of many views / models
- Outputs depth map (inverse depth at each pixel) along with rendered views; can be used with the MATLAB 'surf' function.
- Outputs an 'unproject' matrix, linking each pixel with the coordinates the 3D surface point projected onto that pixel; it is therefore ideal for calibration / pose estimation using a 3D model as reference.
- Along with the 'calib' function, can be used to compute pose from 2D-3D correspondences, and pose-adjust models to images (see below)
- 3D model caching - repeated rendering of the same model automatically use cached data and do not involve re-loading / reading the CG file

This code was developed by Liav Assif and Tal Hassner and was used in the following papers:

- Tal Hassner, Shai Harel*, Eran Paz* and Roee Enbar, *Effective Face Frontalization in Unconstrained Images,* IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Boston, June 2015 (Project and code, arXiv, PDF)
  * These author names are in alphabetical order due to equal contribution.

- T. Hassner, Viewing Real-World Faces in 3D, International Conference on Computer Vision (ICCV), Sydney, Austraila, Dec. 2013 (project, PDF)

- T. Hassner, L. Assif, and L. Wolf, When Standard RANSAC is Not Enough: Cross-Media Visual Matching with Hypothesis Relevancy, Machine Vision and Applications (MVAP), Volume 25, Issue 4, Page 971-983, 2014 (PDF)

If you find this code useful, please add suitable references in your work to any (or all) of these papers.

Code Download

The following files are available for download:

- renderer-1.0.2.tar.gz Linux version of the renderer function (see readme for install details, FAQ and more)
- calib.1.0.1.zip, a MATLAB wrapper for the OpenCV calibration function, used with renderer to adjust the pose of the model rendered using renderer. (see readme for install details, FAQ and more). This is the Linux version, in order to use on Windows, see below for the Windows, OpenCV binaries.
- To help newcomers, we provide pre-compiled versions of the dependent libraries, OpenSceneGraph and Mesa OpenGL in renderer-pre-compiled-libs.0.0.2.tar.gz (~58MB).
- OpenCV binaries for calib on Windows are also available, in opencv_lib.zip and opencv_bin.zip. To use these, unpack to two separate folders and add these folders to the system PATH before running MATLAB.
- An unofficial (and unsupported) OSX port is at bitbucket.org, with special thanks to **Yoav HaCohen**.
- **Elad Shtivi's** code and **Windows binaries** for the real-time demo, including facial animation (see videos below) is now available from github.
- An unofficial (and unsupported) **Windows port for renderer, with special thanks to Chao Yao**: renderer.Windows.1.0.0.zip. See readme.windows.1.0.0.txt for further details. For questions or concerns regarding this Windows version, please contact Chao Yao.

What's new

14th of Apr. 2015
Updated version for the Linux version of renderer; now supports MATLAB 2014b and includes changes to the documentation (previous version available below).

8th of Feb. 2015
Chao Yao has generously provided **pre-compiled Windows binaries for the renderer function**, along with source files, modified for easy compilation on different Windows platforms / MATLAB versions. Code is available in renderer.Windows.1.0.0.zip. Please see readme.windows.1.0.0.txt for additional information.

1st of Dec. 2014
Face frontalization project is now online, including MATLAB code for synthesizing front-facing views of faces in unconstrained images. This project utilizes the calib function as well as data produced by the renderer function (though renderer is not required for frontalization).

30th of Nov. 2014
Added **Windows** pre-compiled **calib** mex file, as well as OpenCV Windows binaries.

15th of Sept. 2014
- **Elad Shtivi's** Windows code and binaries for the real-time demo of renderer and calib, including facial animations, is now online from github. Videos demonstrating some of its capabilities are available on Youtube:

28 Aug. 2014
- Some basic capabilities of the calib+renderer combo, including the addition of facial animation, are demonstrated by **Elad Shtivi** in the following video.

17 Aug. 20144
- calib-1.0.0. Added OpenCV 2.4 support. Documentation and example update.
- renderer-1.0.1. Added demo.m, with many examples.
- FAQ update with solutions to additional issues.
- Added compile_renderer.m and compile_calib.m functions to help build functions on additional machines.

16 July 2014
- Added link to OSX port.

13 July 2014
- Updated renderer code with some bug fixes. New READMErenderer.txt with more information, quick start guide, FAQ and more. Example public domain wrl file now included in the package.

17 March 2014
- Pre-complied dependent libraries file now also contains the Coin3D library, required by OpenSceneGraph to support some 3D formats. This addition is quite large; the previous version of the pre-compiled libraries file can still be downloaded below, in case Coin3D is not required.

1 March 2014
- calib version 0.0.2 containes a few fixes and missing files

23 Dec. 2013
- Renderer code no longer depends on OpenCV
- Pre-compiled dependent libraries are now available.

Previous versions
- 14 Apr. 2015 - renderer-1.0.1.tar.gz
- 30 Nov. 2014 - calib-1.0.0.tar.gz
- 17 August 2014 - calib.0.0.2.zip
- 17 August 2014 - renderer.1.0.0.zip

- 13 July 2014 - renderer.0.0.2.zip
- 17 March 2014 - renderer-pre-compiled-libs.tar.gz
- 23 Dec. 2013 - calib.0.0.1.zip
- 18 Nov. 2013 - renderer.0.0.1.zip

Example usage

See quickstart section in READMErenderer.txt for installation instructions. In case of any problems, please see the FAQ, also in READMErenderer.txt.

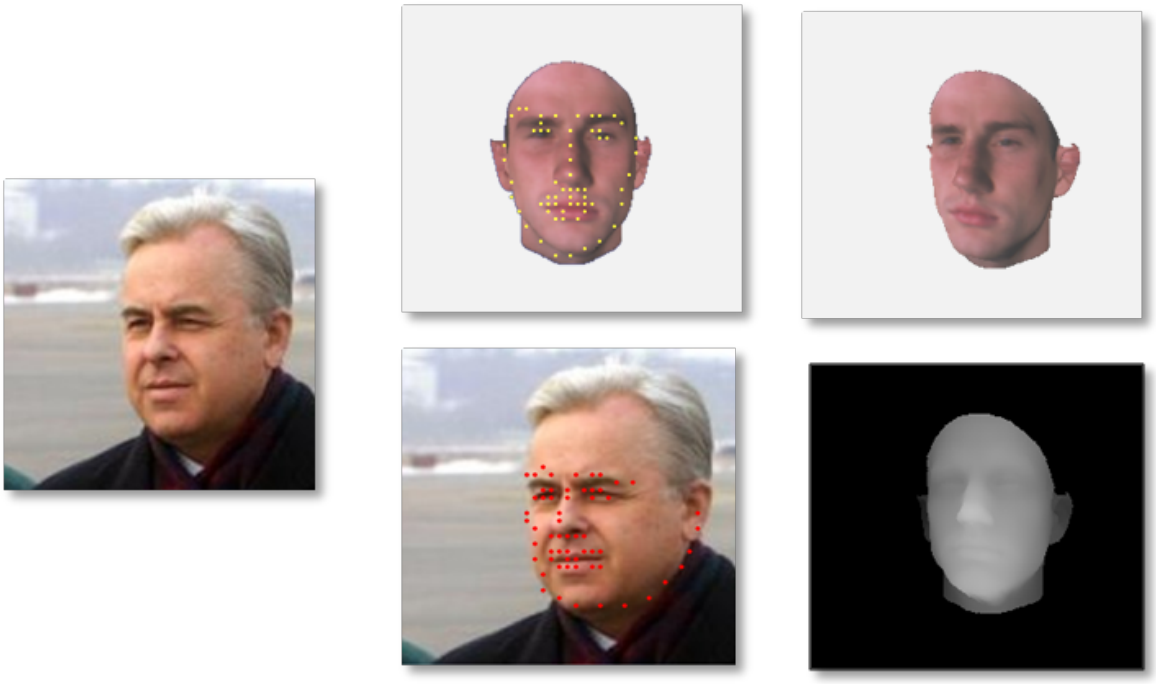
Example taken from Hassner, ICCV'13: Rendering a 3D face model to match the pose of a face appearing in a query photo. Assuming that the code from www.ics.uci.edu/~xzhu/face/ is used for detecting facial features, and a textured 3D model of a face saved as file 'ref_model.obj'.

```
%% STEP 1 - RENDER 3D REFERENCE MODEL, DETECT SPECIFIC FACIAL FEATURES, GET THEIR 3D COORDINATES ON THE 3D MODEL
% render
[refD, refI, refU, outA, outR, outT] = renderer(render_width, render_height, 'ref_model.obj', 0, 1, 1, 0, 0, 0, 'zxy');
% detect facial features on the rendered image, code from www.ics.uci.edu/~xzhu/face/ with modifications to get center of detection window (detect parameters taken from their code)
ref_bs = detect(refI, model, model.thresh);
ref_bs = clipboxes(refI, ref_bs);
ref_bs = nms_face(ref_bs,0.3);
x1 = ref_bs.xy(:,1);
y1 = ref_bs.xy(:,2);
x2 = ref_bs.xy(:,3);
y2 = ref_bs.xy(:,4);
ref_XY = [(x1+x2)/2,(y1+y2)/2];
% get 3D coordinates of facial features, using the refU map from image 2D to model 3D
ind = sub2ind([size(refU,1), size(refU,2)], round(ref_XY(:,2)), round(ref_XY(:,1)));
threedee = zeros(numel(ind),3);
tmp = refU(:,:,1);
threedee(:,1) = tmp(ind);
tmp = refU(:,:,2);
threedee(:,2) = tmp(ind);
tmp = refU(:,:,3);
threedee(:,3) = tmp(ind);
% a bit of sanity
indbad = find(max(threedee,[],2)==0);
threedee(indbad,:)=[];


%% STEP 2 - LOAD QUERY IMAGE, DETECT ITS FACIAL FEATURES
queryI = imread(filen);
img_bs = detect(queryI, model, model.thresh);
img_bs = clipboxes(queryI, img_bs);
img_bs = nms_face(img_bs,0.3);
x1 = img_bs.xy(:,1);
y1 = img_bs.xy(:,2);
x2 = img_bs.xy(:,3);
y2 = img_bs.xy(:,4);
img_XY = [(x1+x2)/2,(y1+y2)/2];
img_XY(indbad,:)=[];


%% STEP 3 - COMPUTE POSE USING REFERENCE 3D POINTS + QUERY 2D POINTS, AND RE-RENDER
% Estimate pose
[est_A,est_R,est_T]=doCalib(size(refU,2),size(refU,1),img_XY,threedee,outA,[],[]);
```

```
% re-render reference to the same pose as query. Note the change in input and output parameters vs. the previous call. Also note that the model file is not reloaded and a cached version is used instead.
[refD, refI, unproject_est] = renderer(render_width, render_height, 'ref_model.obj', 0, 1, est_A, est_R, est_T);
```



(a) A query photo (from the LFW collection)

(b) Facial feature detection on the query and the rendered reference in frontal pose

(c) Output, pose adjusted reference, along with depth-map

Figure 2. Illustrating the example code above. Using a reference 3D model from the USF. DARPA Human-ID 3D Face Database. Detected features on the query image (red points), are matched to the same detections on the reference image (yellow points). The 'unproject' matrix refU is then used to provide each 2D point on the reference image, its corresponding 3D point on the 3D reference model, in the model coordinate system. From these 2D-3D correspondences, a camera matrix is obtained and renderer used again to render a pose adjusted reference image and corresponding depth map.

Copyright and disclaimer

Last update 14th of Apr. 2015