Dataset Renderer Introduction

Rafael Radkowski
Jan 22, 2019

The software **DatasetRenderer** renders images (Figure 1, RGB image, 16bit normals, 16 bit depth values) into images and saves the pose into a .txt file. It allows one to prepare an artificial dataset for CNN training.
Location: https://github.com/rafael-radkowski/DNNHelpers.git
Look for a precompiled version in *Box/Software & Models/DNN_Research/DNNHelpers*

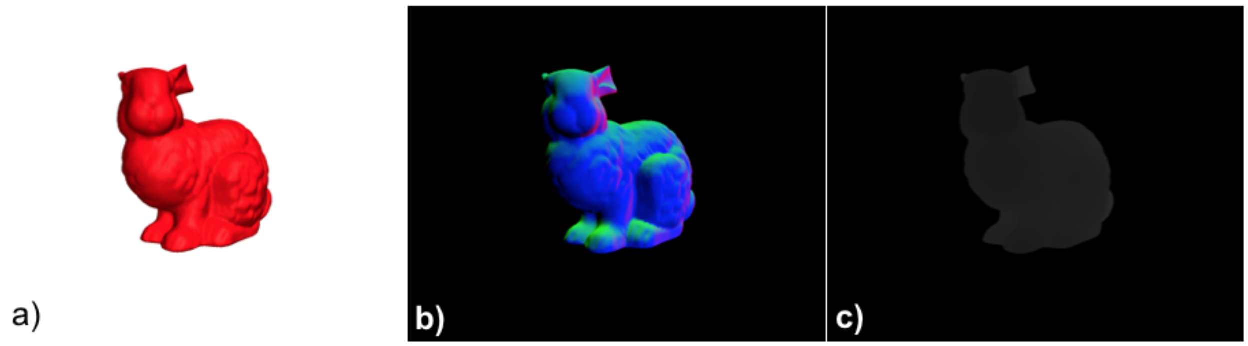

Figure 1: a) a RGB image (RGB8), b) a normal map image (16UC1), c) and a depth map image (16UC1) produced with the dataset renderer.

Features at at glance:
- Load a 3D model (.obj file).
- Set a render model (polyhedron, random)
- Render datasets to image files

1 Prerequisites

The software runs (tested) on Windows 10, Version 1803.
It requires the following 3rd party tools:
- OpenCV v3.4.5, https://opencv.org
- Eigen3 v3.3.7, http://eigen.tuxfamily.org
- GLEW v2.1.0, http://glew.sourceforge.net
- GLFW v3.2.1, https://www.glfw.org
- glm, v0.9.9.3, https://glm.g-truc.net/0.9.9/index.html

Additionally, the software needs a graphics card with OpenGL 3.3, GLSL 3.3.0 core capabilities (tested). Other version may require adaptation.

2 Functionality

The purpose of the software is to render datasets (RGN, normal vector map, depth map) from a loaded 3D model from multiple directions. Thus, to prepare an artificial dataset for CNN testing & debugging. Therefore, it automatically generates camera eye points by using a polyhedron as model-wrapping geometry (see [1], [2]). Figure 2 shows the mechanism. The 3D model is virtually surrounded by a polyhedron (or icosahedron), depending on the selected division-level. Each vertex of the polyhedron acts as a camera eye point. The center of the polyhedron is automatically set as the camera center point, with z, the forward axis of the camera model (note that all our RGB-D images come with RGB data in the xy-frame and depth data along the z-axis).

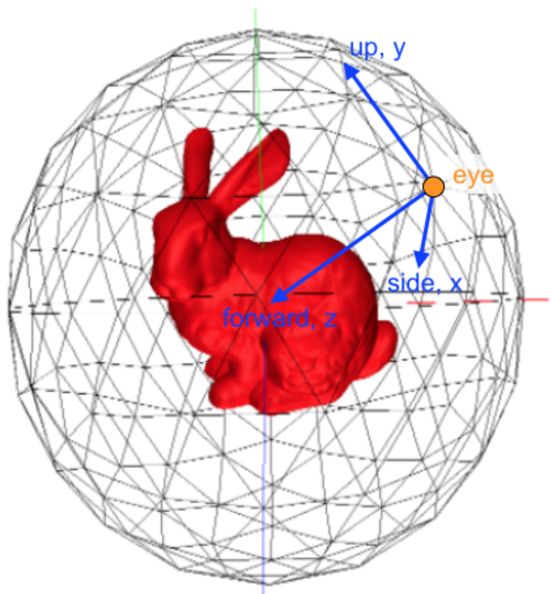

Figure 2: The software generates camera viewpoints by using the vertex points of a polyhedron as camera eye points.

The the user can determine the granularity of the polyhedron (icosahedron) by selecting the number of subdivision for an icosahedron. Figure 3 shows the different results. The software starts with an icosahedron, which is equal to zero subdivisions (div=0). To generate polyhedrons, one needs to divide the edges of the icosahedron (div=1 to N). The first subdivision generates a polyhedron with 42 vertices, thus, 42 eye points, and so on. The software can technically generate an infinite large number of eye points. Subdivision beyond div=8 were not tested. The icosahedron & polyhedron geometry has the advantage that all vertices are equidistantly distributed along the surface.

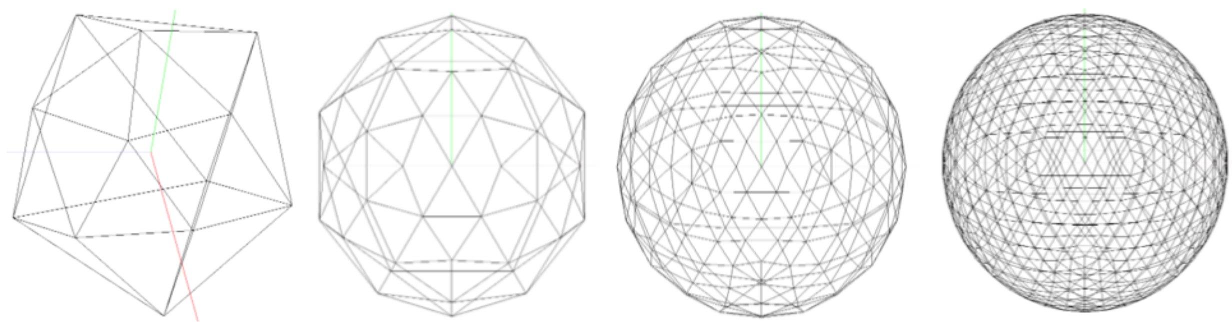

Figure 3: The number of subdivisions (div) of a icosahedron determine the shape of the polyhedron, thus, the number of camera eye ponts.

Note that the camera up-vector is currently fixed towards the general y-direction. Thus, it assumes that each 3D model has a bottom, resting on the xz-plane.

The software uses two render passes from each position to generate the three images: A color + depth map render pass, and a normal map render pass. The first pass generates a color image and a depth image. The color image is of type RGB8 (8 bits/channel), the depth image of GL_DEPTH_COMPONENT32 (32 bit). The depth components are linearized by reversing the non-linear z-component the projection matrix yields using:

$$d' = \frac{2 \cdot far \cdot near}{far \cdot near \cdot d \cdot (far - near)} \quad \text{Eq. 1}$$

with *d'*, the linear depth value, *far*, the far-clipping plane, *near*, the near clipping plane, and *d*, the current non-linear depth value. Note that the value *d* must be normalized with *d/w* to yield a range [-1, 1]. Also note that this equation works for OpenGL only (DirectX uses a slightly different projection matrix). The second render pass generates the normal vector map. It renders them into a color channel of type GL_RGBA32_ARB (32 bits/channel). Note that all render passes store the results in two fbo's. The output window displays the result of a third render pass.

The software creates a white light source. The light source is attached to the eye point to keep the camera-facing model front lit.

3 Usage

The software uses command line options to control its features. Usage (example):

```
DatasetRenderer.exe ../data/stanford_bunny_02_rot.obj -o output
-img_w 1280 -img_h 1024 -m POLY -sub 2  -rad 1.3 -verbose
```

general

```
DatasetRenderer.exe [model_path_and_file]
```

Currently, only .obj models are supported. The 3D model path and file is mandatory.

The following (optional) command line options are supported.

- -o [param] - set the output path. The software writes all output data into this folder. Default is 'output'.
- -img_w [param] - set the width of the output image in pixels (integer), default is 1280.
- -img_h [param] - set the height of the output image in pixels (integer), default is 1024.
- -wnd_w [param] - set the width of the application window in pixels (integer), default is 1280.
- -wnd_h [param] - set the height of the application window in pixels (integer), default is 1024.
- -m [param] - set the camera path models. Can be SPHERE (legacy), POLY (default)
- -seg [param] - for the camera path SPHERE model, set the number of segments (integer).
- -rows [param] -for the camera path SPHERE model, set the number of rows (integer).
- -dist [param] - for the camera path SPHERE and POLY model, set the sphere, polyhedron radius (float). This sets the distance between 3D model and the eye point.
- -sub [param] - for the camera path POLY model, set the number of subdivisions for the *polyhedron* (int), see Figure 3
- -intr [param] - path and filename for the intrinsic camera parameters
- -verbose - displays additional information, no parameter
- -help - displays this help menu

If no parameters are given, *DatasetRenderer* reverts to default parameters. The optional options can be set in any order.

When the software starts correctly, the user should see a window appearing as shown in Figure 4. The main window shows the 3D model and a depth rendering (small window) from the current location. The three additional windows show content of the output images. Note that these windows only appear in verbose-mode (command option -verbose). Also, the window size is 512 x 512 regardless of the output image size set with -img_w, and -img_h.

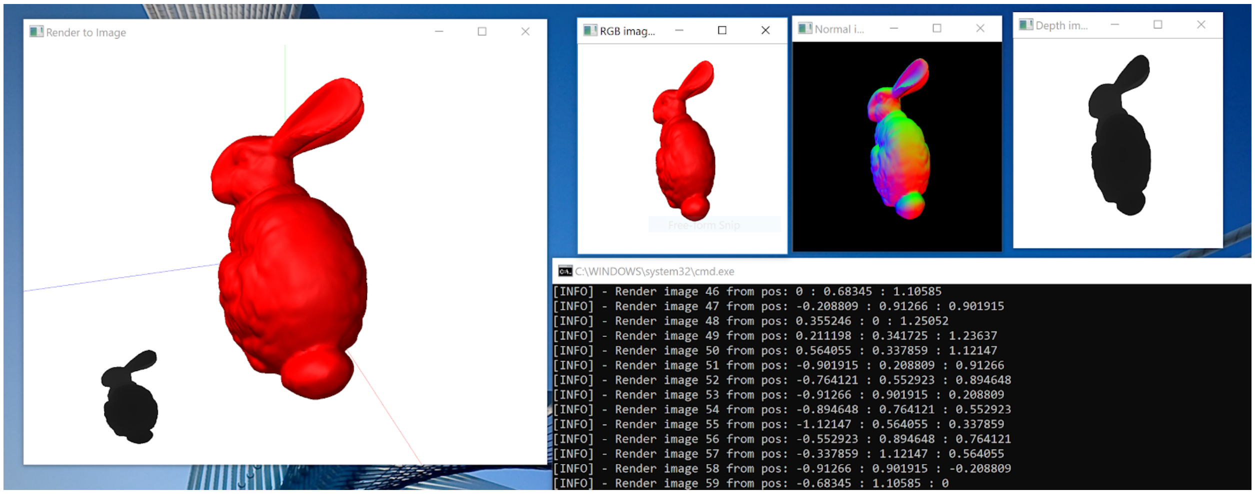

Figure 4: The output window in verbose mode (option -verbose).

DatasetRenderer quits automatically, once all images are rendered. One can find all images in the specified output folder (default is ./output). Figure 5 displays one set of images. All images (.png) are written with opencv support which supports a max. depth of 16 bits per channel. The data types are:

- RGB: CV_8UC3 (int)
- Normal map: CV_16UC1 (short)
- Depth map: CV_16UC1 (short)

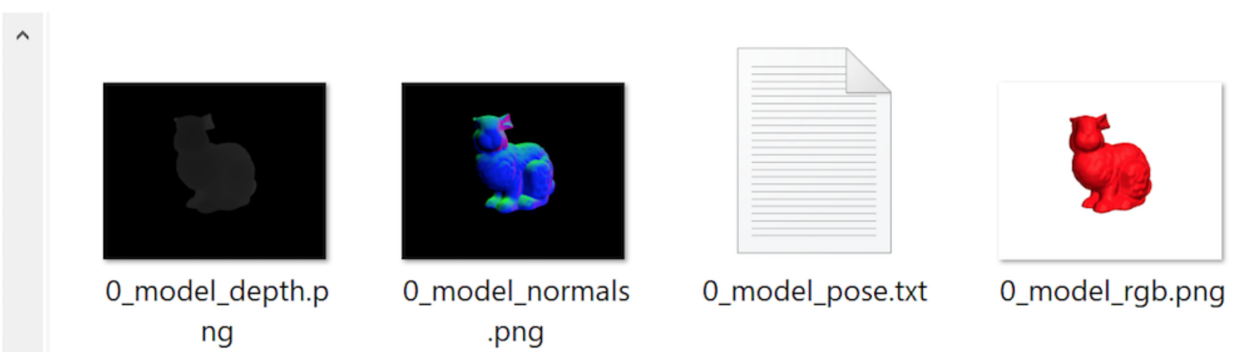

Figure 5: One set of output images

Additionally, the software writes the model pose into a txt. file. The file contains a 4x4 transformation matrix in homogeneous coordinates (colum order).

Also, the software output a summary file *render_log.csv* (Figure 6). Each line represents one image-set. The columns from left to right are:

- index: an integer index for each file. All files from one eye point share the same index.
- rgb_file: path and file of the rgb image
- normals_file: path and file of the normal image
- depth_file: path and file of the depth image
- mat_file: mat and file containing the pose matrix.
- tx, ty, tz: the pose transformation model->camera in Cartesian space (see Figure 2, the rendered coordinate frame).
- qx, qy, qz, qw: the orientation model->camera in Cartesian space as quaternion.

	A	B	C	D	E	F	G	H	I	J	K	L
1	index	rgb_file	normals_file	depth_file	mat_file	tx	ty	tz	qx	qy	qz	qw
2	0	output/0_model_rgb.png	output/0_model_normals.png	output/0_model_depth.png	output/0_model_pose.txt	-0.68345	0	1.10585	0	-0.27327	0	0.961938
3	1	output/1_model_rgb.png	output/1_model_normals.png	output/1_model_depth.png	output/1_model_pose.txt	0.68345	0	1.10585	0	0.273267	0	0.961938
4	2	output/2_model_rgb.png	output/2_model_normals.png	output/2_model_depth.png	output/2_model_pose.txt	-0.68345	0	-1.10585	0	0.961938	0	-0.27327
5	3	output/3_model_rgb.png	output/3_model_normals.png	output/3_model_depth.png	output/3_model_pose.txt	0.68345	0	-1.10585	0	0.961938	0	0.273267
6	4	output/4_model_rgb.png	output/4_model_normals.png	output/4_model_depth.png	output/4_model_pose.txt	0	1.10585	0.68345	-0.48697	0	0	0.873422
7	5	output/5_model_rgb.png	output/5_model_normals.png	output/5_model_depth.png	output/5_model_pose.txt	0	1.10585	-0.68345	0	0.873422	0.486965	0
8	6	output/6_model_rgb.png	output/6_model_normals.png	output/6_model_depth.png	output/6_model_pose.txt	0	-1.10585	0.68345	0.486965	0	0	0.873422
9	7	output/7_model_rgb.png	output/7_model_normals.png	output/7_model_depth.png	output/7_model_pose.txt	0	-1.10585	-0.68345	0	0.873422	-0.48697	0
10	8	output/8_model_rgb.png	output/8_model_normals.png	output/8_model_depth.png	output/8_model_pose.txt	1.10585	0.683451	0	-0.19323	0.680193	0.193229	0.680193
11	9	output/9_model_rgb.png	output/9_model_normals.png	output/9_model_depth.png	output/9_model_pose.txt	-1.10585	0.683451	0	-0.19323	-0.68019	-0.19323	0.680193
12	10	output/10_model_rgb.png	output/10_model_normals.png	output/10_model_depth.png	output/10_model_pose.txt	1.10585	-0.68345	0	0.193229	0.680193	-0.19323	0.680193
13	11	output/11_model_rgb.png	output/11_model_normals.png	output/11_model_depth.png	output/11_model_pose.txt	-1.10585	-0.68345	0	0.193229	-0.68019	0.193229	0.680193
14	12	output/12_model_rgb.png	output/12_model_normals.png	output/12_model_depth.png	output/12_model_pose.txt	-0.40172	0.65	1.05172	-0.25452	-0.17524	-0.04696	0.949897
15	13	output/13_model_rgb.png	output/13_model_normals.png	output/13_model_depth.png	output/13_model_pose.txt	0.401722	0.65	1.05172	-0.25452	0.17524	0.046955	0.949897
16	14	output/14_model_rgb.png	output/14_model_normals.png	output/14_model_depth.png	output/14_model_pose.txt	0	0	1.3	0	0	0	1
17	15	output/15_model_rgb.png	output/15_model_normals.png	output/15_model_depth.png	output/15_model_pose.txt	-1.05172	0.401722	0.65	-0.13663	-0.48097	-0.07618	0.862669
18	16	output/16_model_rgb.png	output/16_model_normals.png	output/16_model_depth.png	output/16_model_pose.txt	-0.65	1.05172	0.401722	-0.39653	-0.43389	-0.22108	0.778225
19	17	output/17_model_rgb.png	output/17_model_normals.png	output/17_model_depth.png	output/17_model_pose.txt	-0.65	1.05172	-0.40172	0.221077	0.778225	0.396525	-0.43389
20	18	output/18_model_rgb.png	output/18_model_normals.png	output/18_model_depth.png	output/18_model_pose.txt	0	1.3	0	-0.70711	0	0	0.707107
21	19	output/19_model_rgb.png	output/19_model_normals.png	output/19_model_depth.png	output/19_model_pose.txt	0.65	1.05172	-0.40172	-0.22108	0.778225	0.396525	0.433889
22	20	output/20_model_rgb.png	output/20_model_normals.png	output/20_model_depth.png	output/20_model_pose.txt	0.65	1.05172	0.401722	-0.39653	0.433889	0.221077	0.778225
23	21	output/21_model_rgb.png	output/21_model_normals.png	output/21_model_depth.png	output/21_model_pose.txt	1.05172	0.401722	0.65	-0.13663	0.480969	0.076178	0.862669
24	22	output/22_model_rgb.png	output/22_model_normals.png	output/22_model_depth.png	output/22_model_pose.txt	1.3	0	0	0	0.707107	0	0.707107
25	23	output/23_model_rgb.png	output/23_model_normals.png	output/23_model_depth.png	output/23_model_pose.txt	1.05172	-0.40172	0.65	0.136633	0.480969	-0.07618	0.862669
26	24	output/24_model_rgb.png	output/24_model_normals.png	output/24_model_depth.png	output/24_model_pose.txt	1.05172	0.401722	-0.65	-0.07618	0.862669	0.136633	0.480969
27	25	output/25_model_rgb.png	output/25_model_normals.png	output/25_model_depth.png	output/25_model_pose.txt	1.05172	-0.40172	-0.65	0.076178	0.862669	-0.13663	0.480969
28	26	output/26_model_rgb.png	output/26_model_normals.png	output/26_model_depth.png	output/26_model_pose.txt	0.401722	0.65	-1.05172	-0.04696	0.949897	0.254524	0.17524
29	27	output/27_model_rgb.png	output/27_model_normals.png	output/27_model_depth.png	output/27_model_pose.txt	-0.40172	0.65	-1.05172	0.046955	0.949897	0.254524	-0.17524
30	28	output/28_model_rgb.png	output/28_model_normals.png	output/28_model_depth.png	output/28_model_pose.txt	0	0	-1.3	0	1	0	0
31	29	output/29_model_rgb.png	output/29_model_normals.png	output/29_model_depth.png	output/29_model_pose.txt	-0.40172	-0.65	-1.05172	-0.04696	0.949897	-0.25452	-0.17524
32	30	output/30_model_rgb.png	output/30_model_normals.png	output/30_model_depth.png	output/30_model_pose.txt	0.401722	-0.65	-1.05172	0.046955	0.949897	-0.25452	0.17524

Figure 6: The log file summarizes all the datasets written into files as well as the 3D model pose transformation (model->camera space).

Known Issues

The software currently uses GLSL shader programs, which can be found in *bin/shaders*. The software needs to find them. Means, start the executable or configure the working directory in Visual Studio accordingly.

References

[1] S. Hinterstoisser, S. Benhimane, V. Lepetit, P. Fua, and N. Navab. Simultaneous recognition and homography ex- traction of local patches with a simple linear classifier. In Proceedings of British Machine Vision Conference (BMVC), 2008[]

[2] Yoshinori Konishi, Kosuke Hattori, Manabu Hashimoto : Real-Time 6D Object Pose Estimation on CPU, arXiv:1811.08588