On the Generation of Ground Truth Data for Depth Reconstruction

Release 0.00

C. Staub, A. Heider, M. Grimm, A. Knoll

June 20, 2012

{staub | heidera | grimmm | knoll } at in.tum.de
Technische Universität München, Robotics and Embedded Systems

Abstract

Acquiring data sets for stereo or multi-view reconstruction with known ground truth is costly and time consuming. Freely available data sets mostly focus on specific scene conditions and might not reflect the intended application scenario. We propose a GPU accelerated ray tracing framework that allows to generate realistic images with adjustable camera properties and projection geometry. Due to the nature of ray tracing, ground truth data can easily be obtained. The system also supports projectors as used e.g. for structured light systems. This facilitates to study interaction effects between different pattern designs, camera and projector properties, as well as the utilized reconstruction algorithm.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/1338]

Contents

1 Introduction

Acquiring data sets with ground truth information is itself already a quite time consuming process, which usually involves additional hardware. If various reconstruction methods, such as stereo, multi-view, or structured light systems are to be considered at the same time, complexity increases considerably. Especially a universal data set for structured light systems is hardly to be realized, since the systems strongly depend on specific pattern designs. Standard resources, such as the Middlebury stereo data sets [6], provide data that allows the comparison of different reconstruction algorithms, but might not reflect the actually intended application domain.

We present a framework that tightly integrates the generation of data sets and ground truth by means of ray tracing, with different reconstruction methods, and patterns for structured light. The recent acceleration of ray tracing by GPU's makes it interesting for interactive applications. For instance, the automotive sector makes use of ray tracing to simulate a PMD camera within a virtual testbed for advanced driver assistance systems [5]. Schmalz [7] used the open-source software renderer Povray [2] to generate static images for structured light. With regard to the medical context, ground truth data is rarely available. The Imperial College provides the reconstruction of a beating heart model, captured with a high speed CT [1]. In combination with images of the original texture such data sets are highly interesting for our ray tracing framework and can e.g. be used within a structured light simulation.

Our framework provides the following main advantages:

- Scenes can be chosen according to domain specific requirements. A variety of different scene models is available from CAD drawings and can easily be created on demand. Scenes can be either dynamic or static.
- Freely adjustable camera and projector properties (intrinsics and extrinsics) and direct comparability with real world setups.
- Simulation of sensor characteristics and sources of interference (e.g., effects of noise, debayering)
- Investigation of interaction effects of the used projector (e.g., resolution and luminosity) and camera in structured light settings.
- Simulation of different lightning conditions for both ambient light and material surfaces.
- Ground truth data is directly available.

2 GPU Accelerated Ray Tracing

Ray tracing can be seen as a general approach of simulating camera systems, imitating the behavior of both sensors and lenses, by following the path that light takes through a virtual scene. Being based on physical principles, ray tracing is capable of producing realistic and high quality images, which easily outperform those produced by traditional rasterization renderers. The introduction of GPU-based ray tracing engines, such as the Nvidia OptiX^{TM} framework [4], allows now the utilization of this computationally expensive technique in interactive applications. As the OptiX SDK comes as a general purpose raytracing library, it is not pre-configured for a specific rendering method, but rather allows the user to implement his own ray tracers.

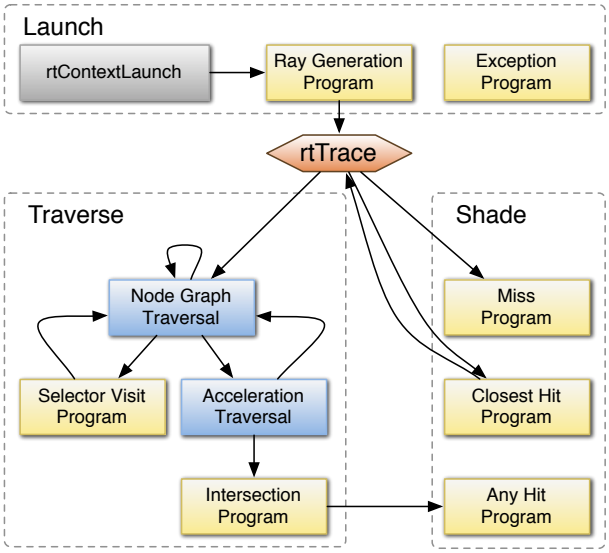

Figure 1: OptiX control flow [4]

To generate realistic images for the purpose of benchmarking depth reconstruction methods with corresponding ground truth data, we are particularly interested in the simulation of camera and projector systems. Ray tracing inverts the typical imaging process of cameras, where light emitted in the scene enters the camera. Instead, "lines of light" are traced backwards from the camera through the scene. This approach reduces the computational cost, since only the parts of the scene which contribute to the image are considered. The control flow of the Nvidia OptiX ray tracer is illustrated in Figure 1. Overall, ray tracing is divided into three main parts. As the host program invokes the ray tracer, it has to decide in which direction rays have to be send into the scene. This step corresponds to the lens of the simulated camera and is further discussed in the next section. After a ray has been sent into the scene, the next step is tracing it throughout the scene. The scene is stored as a node graph that consists of the objects in the scene. As this step is common to all ray tracers it is directly handled by the OptiX framework.

At found intersections the simulation calculates the reflectance of the light ray on basis of the object's surface properties. On one hand the object directly reflects light in the direction of the ray, on the other hand light from the scene is emitted, such as ambient light from different light sources or scattered light from surrounding objects.

The intersections of rays and scene objects is related to the object's distance from the camera. Ground truth depth data is therewith directly available.

2.1 Cameras and Projectors

Our simulation software allows the user to specify an arbitrary number of cameras and projectors. Cameras allow to take images of the scene from any spatial configuration, while projectors emit light through a pattern. The pattern can be static or dynamic over time, illuminating the scene objects with the projection of the pattern. As mentioned above, ray tracing inverts the light transfer process. Thus the used camera model also has to be inverted and therewith specifies in which direction the outgoing rays are send into the scene. This also means that the projector can be handled as a regular camera model, with an additional texture

representing the pattern.

The projection characteristics of a camera mainly depend on the lens and the image sensor. We simulate this hardware by the following groups of parameters:

- **Extrinsic parameters** determine the position and rotation of the camera in relation to the scene. More important, the extrinsics also specify the relationship between multiple cameras and optional projectors.
- **Intrinsic parameters** define how incoming light is manipulated by the lens system before striking the sensor
- **The sensor model** specifies how the sensor reacts to incoming light, e.g. special sensor characteristics can be simulated, such as noise.

The camera model is implemented as a ray generation program. The code is executed in parallel, for each pixel in the desired output image. Light can take a multitude of different ways to end up hitting one single pixel. A ray tracer is limited by computational power and has to approximate the effect using only a small number of rays (sampling factor). How to combine the results that are obtained by different rays in one pixel is task of the sensor model.

A suitable model to describe the projective geometry of a lens is the Bouguet model [3]. Its wide use in camera calibration and the support in many open source vision libraries allows us to directly transfer camera parameters of real camera systems into our simulation framework. In contrast to the original Bouguet model, where all camera parameters are expressed in pixel units, we specify the parameters as a ratio with respect to the sensor size. This is necesarry since measurements given in pixels correspond to a fixed sensor resolution, the simulation however is intended to support user defined sensors.

Instead of undistorting an image as in image calibration, the Bouguet model is used to generate a distorted image during ray tracing. Due to the approximation of tangential and radial distortion with higher-order polynomials in the Bouguet model, the required inversion is not straight forward. Starting from the image coordinates $x_n = (x_{n_x}, x_{n_y})$ given by a regular pinhole camera model with camera matrix K, the distorted coordinates x_d of x_n are given with

$$x_d = \begin{bmatrix} x_{d_x} \\ x_{d_y} \end{bmatrix} = \left(1 + kc_1 r^2 + kc_2 r^4 + kc_5 r^6\right) x_n + dx \tag{1}$$

with dx being the tangential distortion

$$dx = \begin{bmatrix} 2kc_3 x_{n_x} x_{n_y} + kc_4 (r^2 + 2x_{n_x}^2) \\ kc_3 (r^2 + 2x_{n_y}^2) + 2kc_4 x_{n_x} x_{n_y} \end{bmatrix} \tag{2}$$

where $(kc_1, \cdots, kc_5)$ are the distortion coefficients, and $r^2 = x_{n_x}^2 + x_{n_y}^2$. The final projection is then expressed by

$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = K \begin{bmatrix} x_{d_x} \\ x_{d_y} \\ 1 \end{bmatrix} \tag{3}$$

For inverting the camera model, first the effect of the linear camera calibration matrix is undone by solving the equation 3 for x_d. The second step, the inversion of the terms for radial and tangential distortion is more

challenging. As the radial distortion is described by a polynomial of 6th grade, there is no analytical solution for x_n. Thus an iterative numerical solution is used to calculate x_n. We first solve Eqn. 1 for x_n

$$x_n = \frac{(x_d - dx)}{(1 + kc_1 r^2 + kc_2 r^4 + kc_5 r^6)} \tag{4}$$

and start an optimization with the initial guess $x_n = x_p$ on the left side of the equation and x_n in the expression of dx. The evaluation of Eqn. 4 yields to a new $\hat{x}_n$ which is then used in the next iteration. The resulting ray direction is then be given as $x_{n_x} \cdot u + x_{n_y} \cdot v + w$, where (u, v, w) are homogeneous image plane coordinates.

The sensor model specifies how many rays are used to determine the final value of a pixel. The resulting image quality is thus mainly governed by the chosen sampling rate. Sensor specific characteristics, such as the noise performance of sensors, non linear pixel responses, or vignetting, can easily be integrated in the model.

3 Framework

The above-presented ray tracer is integrated into a flexible framework that currently consists of three more components: (1) a pattern generator, which is used to create the texture consumed by the projectors, (2) a depth reconstruction module that wraps different reconstruction algorithms and (3) an evaluation module which compares the achieved reconstruction result with the generated ground truth of the ray tracer.

The functionality of the framework is organized in *plugins*, which can be loaded dynamically. This reduces the effort to implement new features, algorithms or pattern generators. The plugin system is based on the Qt plugin loader. Each module defines its own Qt interface which must be implemented by the plugin.

Each component of the framework offers a user-interface, enabling interactive experimentation with all system parameters, such as intrinsic and extrinsic camera/projector settings, pattern design, and parameters of the reconstruction algorithm.

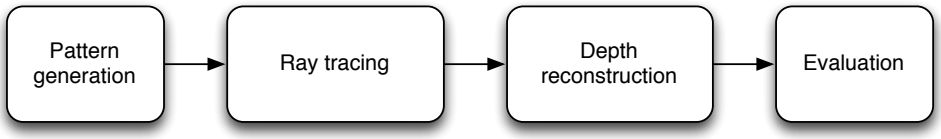

Figure 2: Data flow inside the framework

Figure 2 illustrates the typical processing pipeline of the framework. First, a pattern is generated and passed to the ray tracer, which in turn projects it onto the scene. The resulting color images are then forwarded to the depth reconstruction module. The module processes the incoming images at the highest possible framerate and emits the corresponding depth map. In the fourth phase all data (the ground truth as well as the reconstruction results) is passed to the evaluation module, which evaluates the results on the basis of the metrics proposed by Scharstein and Szeliski [6].

Communication between the modules is realized using the Qt Signals & Slots concept. This reduces dependences between the different phases and enables easy integration of a graphical user interface. Image results are always handed with a descriptive parameter tree to the next module. Handling this kind of information

together with the affected images allows e.g. to transfer knowledge about the utilized pattern structure to the reconstruction module.

4 Example

As a proof of concept, we demonstrate the influence of a projected pattern in a stereoscopic setup. In this example, we use a typical scene as encountered in many industrial applications, the inspection of a circuit board. The scene has poorly textured areas on the board as well as sharp edges. Two cameras are used in conjunction with a projector to capture different views of the object which are then used to reconstruct a depth image.

The generated pattern realizes a block-wise Hamming-distance, supporting the windowing technique of the blockmatching method in differentiating different areas. Figure 3 shows the simulation results of the same view with and without a projected pattern.

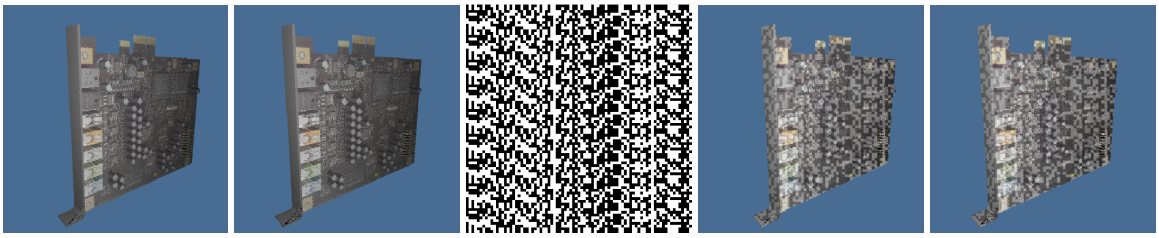

Figure 3: Ray tracing results: without a pattern (left/right), the projected pattern, with a pattern (left/right)

OpenCV stereo blockmatching is used as reconstruction algorithm on the images generated by the ray tracer. Results of the reconstruction together with the obtained ground truth are depicted in Figure 4. As expected, projecting additional texture onto the scene improves the performance of the blockmatching reconstruction algorithm. More interestingly, changes of the setup, e.g. modifying the Hamming distance of the pattern or the used block size, directly affects the output images and can be observed in real-time.

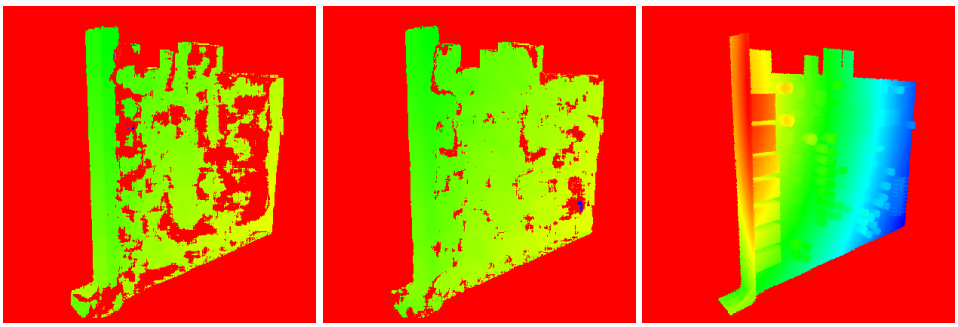

Figure 4: Depth reconstruction results: blockmatching without projector, blockmatching with projector, ground truth generated by the ray tracer

References

[1] http://hamlyn.doc.ic.ac.uk/vision/. 1

[2] http://www.povray.org/. 1

[3] Janne Heikkila. Accurate camera calibration and feature based 3-D reconstruction, 1997. 2.1

[4] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. OptiX: A General Purpose Ray Tracing Engine. *ACM Transactions on Graphics*, August 2010. 2, 1

[5] Erwin Roth. Advanced Driver Assistance System Testing using OptiX. NVIDIA GTC, 2012. 1

[6] Daniel Scharstein and Richard Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *Int. J. Comput. Vision*, 47(1-3):7–42, April 2002. 1, 3

[7] Christoph Schmalz. *Robust Single-Shot Structured Light 3D Scanning*. PhD thesis, Universität Erlangen Nürnberg, 2011. 1