Data Generation Toolkit for Image Based Rendering Algorithms

V Vamsi Krishna, P J Narayanan
Center for Visual Information Technology
International Institute of Information Technology, Hyderabad, India
{vkrishna@research., pjn@}iiit.ac.in

Keywords: Synthetic Data, IBR, Data Generation Tool

Abstract

Computer Vision algorithms require accurate and high quality data for experimentation, tuning, and testing. Each research group used to have their own test data initially. Some of the test data came to be shared, making comparison of performance of different algorithms possible. In this paper, we present DGTk a tool for generating data sets primarily for Computer Vision and Image Based Rendering algorithms. Researchers can compose a scene of their choice using standard models, place lights and cameras in it and compute a variety of representations about the scene. The data supported include images, depth maps, calibration matrices, correspondences etc. The tool provides an intuitive interface for setting up 3D scenes by importing some standard 3D model files. It provides an easy to use interface for setting up and imaging rich static and dynamic scenes and enables the sharing of data among researchers. DGTk goes beyond being a graphics authoring system and is an independent, lightweight, and extensible tool that is computer vision aware.

1 Introduction

Computer Vision (CV) and Image based rendering (IBR) algorithms are used to analyze the events and structure from videos and images given some additional information like calibration of the camera. IBR algorithms use various intermediate structures computed from the images and other imaging parameters such as depth map, correspondences, etc for testing and tuning. Evaluation and tuning of CV or IBR algorithms require high quality images with accurate ground-truth information. Synthetic data is useful as, qualitative and quantitative analysis of the performance of these algorithms is possible using them. The performance of the algorithms on synthetic data is a good indicator of it's performance on real world images. The most important advantage of using synthetic data is that it would help in cross checking the information reconstructed by an algorithm.

Traditionally data sets (synthetic or real) were created by individual researchers for testing their algorithms. The data sets so generated have become standard test beds for benchmarking the performance of other algorithms e.g, CMU Datasets [8]. Every algorithm makes it's own assumptions

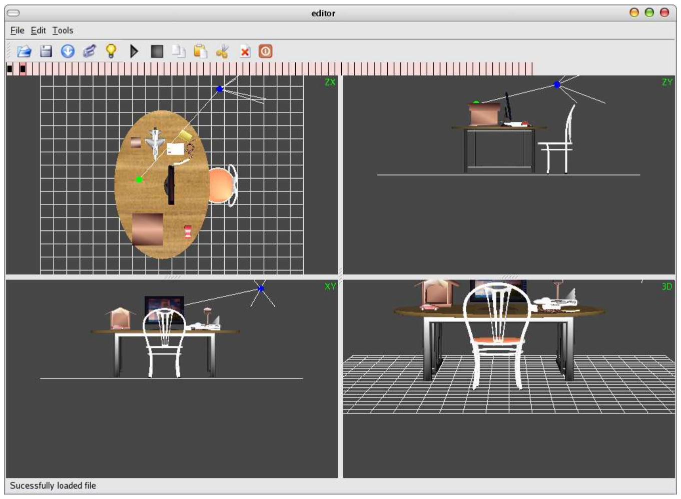


Figure 1: Screen shot of the tool

about the available information. A data set created for testing one particular algorithm may not be valid for testing the performance of a newly developed algorithm. Diversity in the data set becomes a critical issue. For creating diversity, one has to find or build another dataset, which again is a tedious process. This is because the tools developed for data generation are generally very limited and non-scalable. A generalized tool to generate synthetic data would be very useful for various researchers working in these areas.

In this paper, we present DGTk a toolkit to generate data for computer vision and image-based rendering applications. The goals of the toolkit are to enable individual researchers generate high quality, synthetic data to test and tune their algorithms and to help them share the data with others in the community. The DGTk is designed to be a standalone toolkit using which one can create a scene consisting of cameras, lights, geometric objects, etc. For versatility, the tool allows the import of object models in standard formats like blender, ac3d, 3D Studio, etc. Several objects, lights, and cameras can be placed in the scene. Each of the above can move. Their positions can be specified at key frames and interpolated for the intermediate time steps. The scene description can be saved and shared with others. Several representations at many resolutions can be generated for each camera in the scene including images, depth maps, object maps, calibration matrices, pairwise correspondence, etc. The image generation is done using OpenGL API for the sake of speed. Very high-quality imaging is supported through ray tracing. Our tool outputs files for the open-source ray tracing tool POVRay for this purpose. The tool is also designed to be extensible. Users can write their own code to handle another model format or to generate novel representations.

The primary contribution of this work is in the creation of a tool useful for individual researchers to generate high quality, ground truth data and share it with others. Our tool is computer vision aware; it is not just another graphics authoring tool. It therefore generates depth maps, calibration matrices, correspondences, etc., which are very valuable to computer vision researchers. Conventional graphics authoring tools focus on setting up the 3D world and generating views of it. The data generation can be added as additional functionality to a tool like 3D Studio Max or Blender. We chose to create a new tool of our own since the requirements are different. We need to represent additional type of information about the environment which can require non-intuitive extensions to their existing file formats. A simple, open-source tool that can import standard models but has an internal structure that suits computer vision applications is preferable. That is the design philosophy behind the DGTk. The tool along with a few interesting data sets is available for downloading from our site.

Section 2 overviews the prior avenues for high quality data with ground truth. Section 4 presents the design and implementation of DGTk including discussion on its design philosophy and how the goals are met. Section 5 gives a few administrative facts about the tool and its data format and a few concluding remarks. In Section 6 gives the conclusions and future work.

2 Related work

Various data sets have been created to evaluate Computer Vision algorithms recently. For example, Scharstein and Szeliski [3] have designed a collection of data sets for easy evaluation of stereo algorithms. In another related work Szelski [4] describes a method for acquiring high-complexity stereo image pairs with pixel accurate correspondence information using structured light. Steven M. Seitz at al [7] present a method for acquiring and calibrating multiview image datasets with high-accuracy ground truth. Fellenz [1] have developed a low cost vision head for enhancing the dense depth maps. CMU over the years has developed a large number of data sets which have become the standard test data sets e.g, CMU PIE database [8], etc.

Though such enormous amount of test data is available, the data is fixed and there is no way to change the data based on the requirement of the user. A generic tool, if developed for this purpose, would allow the researchers to make new data themselves with ease. Since new type of data may be required by the researchers, for example LDI (Layer Depth Images), point based etc, such a tool should be extend able to include such new file formats.

The 3D authoring tools (both commercial and free) such as 3DSMax, Maya, blender etc are capable of rendering high resolution images using ray tracing, But they do not generate depth maps, Calibration data, Corresponding points, Layer depth images etc readily, which are typically required for testing CV and IBR algorithms.

Our tool provides the flexibility and richness of virtual environment authoring tools in scene description. It is also tuned specifically for CV/IBR and can generate very high resolution images and other representations like camera calibration matrix, depth maps, LDIs etc. It also has the capability to produce image sequences of dynamic scenes along with all assorted information. The tool does not provide functionality for adding vertices or triangles to an existing model. Though such provision could be made available, that is not the focus/purpose of the tool. However there is a large set of freely download able models currently available on the Internet which can be used directly in our tool.

3 Data Generation Tool: Requirements

The requirements for a data generation tool are along three aspects:

Versatility: The tool should allow creation of complicated scenes that mimic the real world scenes. These scenes could involve motion of various objects or camera in the scene etc. Since some of the algorithms rely on videos for acquiring information about a scene, the tool should be able to generate videos or each frame of the videos which help in testing such algorithms. The tool should be capable of generating different kinds of data and representations used by researchers such as images, depth maps, correspondences, etc.

High Quality: The quality of the images and ground truth is very critical for testing it. The high quality data would help us determine the robustness of a particular algorithm (i.e, how much error in the data would be acceptable to it) by generating the low quality versions of the same scene.

Flexibility: Since we are trying to address the more general problem of creating data sets for a wider range of CV and IBR algorithms, the tool should be focused on two major aspects. Firstly, it should have interactive way of setting up the required scene (either static or dynamic) for the data set. Secondly, the GUI of the tool should be easy enough for the user to generate what is required. The user should have the facility to easily add code to generate other kinds of representation.

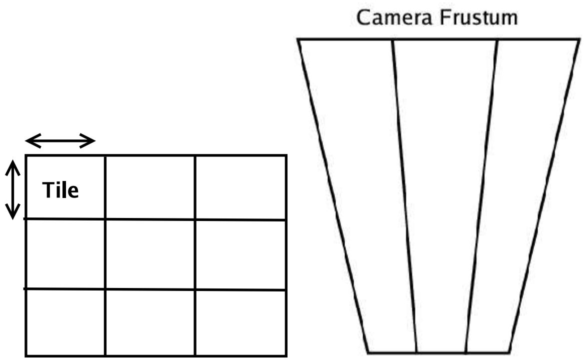


Figure 2: Camera setup for high resolution rendering

4 DGTk: Design and Implementation

Our tool provides the user with the flexibility to import different kinds of 3D model files (ac3d, 3ds, md2 or povray files). This enables the users to create complicated scenes that mimic those in the real world. These scenes could involve motion of various objects or camera in the scene. Since some of the algorithms rely on videos as input, we have provided a facility for the user to render sequence of images with animations (camera or objects). DGTk allows the users to add any number of cameras each with different parameters. If the user wants to add a special configuration of cameras, it can be done by just adding the parameters of the cameras to the scene file (described later). The following sections describe the implementation details of our tool.

4.1 Arbitrarily High resolution

High resolution images are important because they can be used to test the difficult cases of the algorithms. Such high resolution images can generated by our tool based on user specified constraints. As the resolution of the data decreases, the accuracy of the algorithm would go down. So data sets with such varying resolution would help the user to determine the robustness of the algorithm.

Resolution can be thought of in multiple ways:

- Width and height of the image generated.
- The worst case distance in world units in X, Y directions represented by one pixel distance in the image.

For rendering arbitrarily large-sized images Graphics systems allow generating them tile-by-tile with view frustum set asymmetrically (Figure 2). Our system uses a small tile size (640×480) when tiling is necessary because it is the minimum supported resolution on any kinda of hardware. The number of tiles to be rendered depends on the image size and tile size. Assuming that a symmetric frustum is defined using the parameters Left, Right, Top, Bottom, near and far, we can set the asymmetric view frustum for each of the tiles.

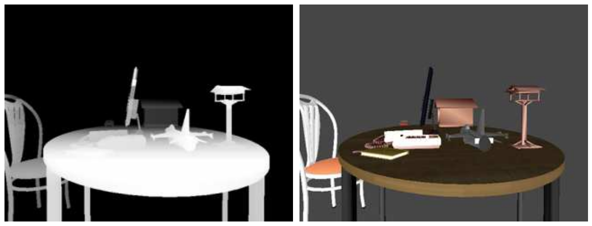


Figure 3: Depth map and Texture map of a scene

4.2 High Quality

When rendering the scene from each camera location, the in-built renderer uses OpenGL features like texture maps, lights, materials etc, to generate high quality (realistic) images. The default renderer may not meet the requirements of the user (due to lack of shadows etc), Hence we provide a facility for the user to export the scene to a Povray [2] scene. This enables the user to use the povray ray-tracer to render the scene.

4.3 Camera Calibration

Calibration information of a camera is very vital for many of the CV algorithms which try to estimate the 3D structure of an object from it's images. Our tool can generate the camera calibration information for each camera. The OpenGL matrix is retrieved and transformed into the form $K[R|t]$, Where K is a 3×3 matrix which refers to the internal parameters of the camera. R and t ($[R|t]$ a 3×4 matrix) give the external parameters of the camera.

The internal parameters ($K_{3\times3}$) of the camera matrix can be obtained from the homogeneous projective matrix and the viewport matrix used in OpenGL. The external parameters ($[R|t]_{3\times4}$) of the camera can be obtained by removing the last row of the homogeneous viewing matrix used in OpenGL. The 3×4 matrix obtained by multiplying K and $[R|t]$ is the camera matrix in the world co-ordinate system.

4.4 Depth Maps

A depth map represents the depth values corresponding to each pixel in a scene (Figure 3). Depth maps of the scene from each camera can be generated by the tool. This would essentially give the shape of the object in the 3D space. Many algorithms

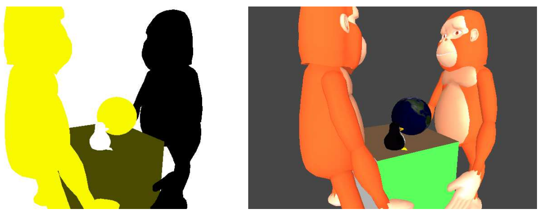


Figure 4: Object map and Texture map of a scene

trying to retrieve 3D information from a scene essentially try to estimate this depth map and there by construct the 3D model. We estimate the 3D world co-ordinates of each pixel in the image by back projecting them into the world co-ordinate system using the camera calibration data. The depth buffer is read back and transformed by the inverse projection matrix to obtain the depth at every pixel. We save the depth maps in a binary format along with the camera calibration data (both at double precision). The first we write the projection matrix, then the model view matrix, followed by two integers specifying width and height of the depth map. This is followed by the depth of each pixel of the image in the world co-ordinate system. The depth maps can be used by algorithms that require them such as depth-image rendering etc. They can also be used as high quality ground truth by structure recovery algorithms.

4.5 Object Maps

Many segmentation, matting and silhouette extraction algorithms try to seperate an object from others. We provide ground truth for this in the form of an object map. The object map is an image where each pixel is labeled with a unique id corresponding to the object that projects to it. Our tool generates the Object Maps as images in R, G, B format (Figure 4) where the object id is stored as a unique R, G, B values.

4.6 Correspondences

Some CV and IBR algorithms require correspondence information between two input images, i.e, which pixel does a particular pixel in first image correspond to in the second image. Our tool can generate dense correspondences (i.e, for each and every pixel in the first image, we find the corresponding pixel in the second image.) between pairs of images, and sparse correspondences (corresponding points for the selected pixel locations in the first image). The correspondences are stored as floating point disparity to have sub pixel precision.

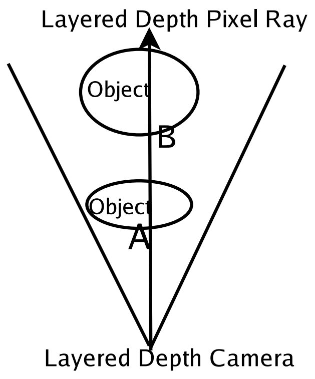


Figure 5: A and B are stored in the same depth pixel

4.7 Layered Depth Images (LDI)

LDIs were first introduced in J.Shade et al [5] and is a special data structure for IBR. This is a more general alternative to sprites and sprites with depth. Unlike the general Depth Maps which just store the depth value of the first hit object in the view direction, an LDI stores multiple depth and color values(depth pixel) per pixel in the view. We used a simple line triangle intersection method to find the LDIs of a scene. We trace the path of each ray from the camera center to the end of the frustum, and keep storing all the depth and color values that the ray comes across before reaching the end of the frustum (Figure 5). The points of intersection and depth information are calculated at double precision. The LDI's are stored in a binary format. First two integers width and height. This is followed by $width \times height$ number of Layered Depth pixels. Since each Layered depth pixel consists of the R, G, B, A and Z and spatial index, The number of depth pixels (an unsigned int) at each layered depth pixel followed by the depth pixels are written.

4.8 Dynamic Scenes

The tool has the ability to generate dynamic scenes using the information specified by the user in the form of key frames. The user can with ease setup the scene for every key frame and generate the animated sequence of interpolated images. The interpolation of the key frames is done based on the difference between two consecutive key frames. So if the user wants some animation to be faster he/she just has to make sure that the difference between two key frames is low.

The position vector of the objects in the scene is interpolated using simple linear interpolation between the initial and final positions. Where as we use spherical linear interpolation of quaternions for the smooth interpolation of rotations [6].

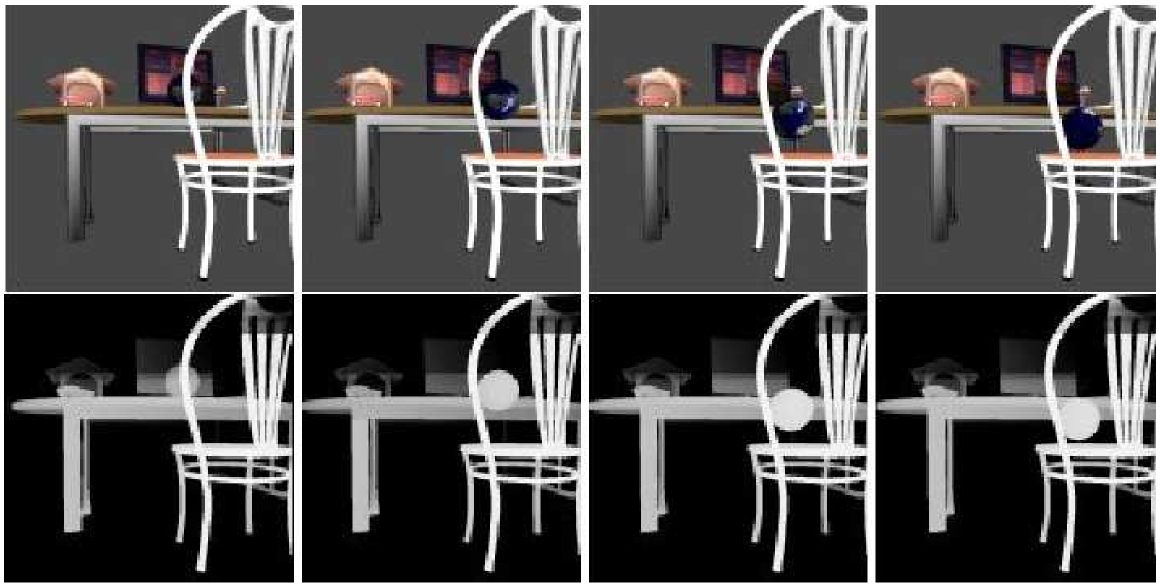
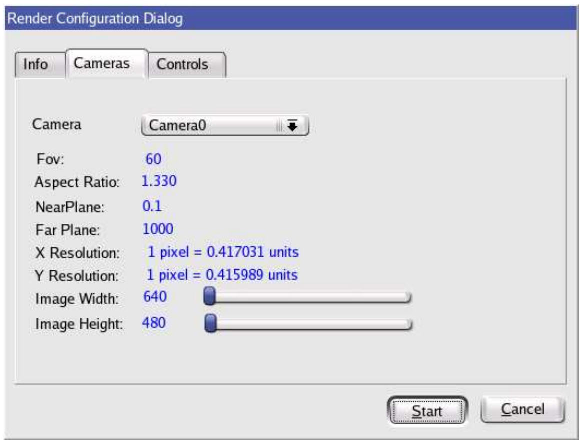


Figure 6: Depth map and texture map of a dynamic scene


Render Configuration Dialog

Info | Cameras | Controls

Camera: Camera0

Fov: 60

Aspect Ratio: 1.330

NearPlane: 0.1

Far Plane: 1000

X Resolution: 1 pixel = 0.417031 units

Y Resolution: 1 pixel = 0.415989 units

Image Width: 640

Image Height: 480

Start | Cancel

Figure 7: GUI for image resolution control

4.9 Ease of use

Since our target audience are researchers in the fields of CV and IBR, we adopted the graphical user interface used by most of the 3D authoring tools (Figure 1). The user can just click and drag the objects to move, rotate or scale them in the 3D space. The tool currently has support for loading AC3D, 3DS, Md2, and a limited polygonal models of Povray scene format. We have support for exporting the 3D scene to povray files (.pov) which can be rendered using a ray tracer like povray. The resolution of the image to be generated from each camera's point of view in the scene can be adjusted using simple scroll bars. The tool automatically shows the resolution that would be achieved using that particular image width for that camera (Figure 7) .

The user can add lights and cameras, move them like any other 3D objects. Provision for preview of the scene from a particular camera's point of view is given, to help the user know exactly how the scene would look in the final images. Our tool has support for generating dynamic scenes. We used simple Key Frame animation to help in creating dynamic scenes. The user just has to move the objects around and specify that those are the positions of the objects in the key frames. This key frame information is used by the tool to interpolate the intermediate frames. The time line is shown on the tool. The user just has to click to select and make some frame as a key frame.

4.10 Extensibility

The user can extend the tool to support new 3d file types as well as new output formats. For doing this the user just has to implement a class which inherits from glObject class and has to over ride the render method of that class and write the rendering function for the new file format. Similarly if the user wants to generate a different output format, the user can access all the 3D objects in the scene and can make OpenGL calls to retrieve required information about the scene.

5 File Format

We have designed an ASCII file format for storing the scene that the users setup using our tool. The information about the objects such as their positions, name of the file are stored.

The first line in the file describes the number of objects present in the scene. The lines following this number are the details about each object. We have tags describing what kind of object follows. glObject_Light refers to light objects, glObject_AC to represent an AC3D file or a povray scene file, glObject_Camera for specifying the camera details, glObject_3DS for 3ds objects and glObject_Md2 for md2 objects. Many objects, cameras, light sources can be present in a single scene file. This is followed by an integer again which tells how many key frames are there in the scene. Then details of each key frame follow. First the frame number of key frame, followed by each object's center and rotation details.

6 Conclusions and Future Work

In this paper, we presented a versatile toolkit to produce high quality data for Computer Vision applications. It can produce various types of data used in Computer Vision. The tool helps individual researchers to generate data and share it with others. A wide range of resolutions, representations, and information are supported by the tool. The tool is currently being enhanced to generate data with given noise properties, to generate point-based representations of solid objects, etc. We are also working on a standard plug-in architecture to enable smooth extension of the tool's functionality.

References

[1] Winfried A. Fellenz, Karsten Schluns, Andreas Koschan, and Matthias Teschner. An active vision system for obtaining high resolution depth information. In *CAIP '97: Proceedings of the 7th International Conference on Computer Analysis of Images and Patterns*, pages 726–733, London, UK, 1997. Springer-Verlag.

[2] http://povray.org. Povray.

[3] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1-3):7–42, 2002.

[4] Daniel Scharstein and Richard Szeliski. High-accuracy stereo depth maps using structured light. In *Computer Vision and Pattern Recognition (CVPR)*, pages 195–202, 2003.

[5] Steven M. Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Rick Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. *Computer Vision and Pattern Recognition (CVPR)*, 2006.

[6] Jonathan Shade, Steven Gortler, Li wei He, and Richard Szeliski. Layered depth images. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 231–242, New York, NY, USA, 1998. ACM Press.

[7] Ken Shoemake. Animating rotation with quaternion curves. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254, New York, NY, USA, 1985. ACM Press.

[8] Carnegie Mellon University (CMU) Computer vision Test Images. http://www.cs.cmu.edu/∼ cil/v-images.html.