SYNTHESIS OF THREE-DIMENSIONAL VIRTUAL WORLDS FROM MONOCULAR IMAGES OF URBAN ROAD TRAFFIC SCENES

Ankita Christine Victor

Master of Technology Thesis

June 2019

International Institute of Information Technology, Bangalore

SYNTHESIS OF THREE-DIMENSIONAL VIRTUAL WORLDS FROM MONOCULAR IMAGES OF URBAN ROAD TRAFFIC SCENES

Submitted to International Institute of Information Technology,
Bangalore
in Partial Fulfillment of
the Requirements for the Award of
Master of Technology

by

Ankita Christine Victor
IMT2014005

International Institute of Information Technology, Bangalore
June 2019

Dedicated to

mom and dad.

Thesis Certificate

This is to certify that the thesis titled **SYNTHESIS OF THREE-DIMENSIONAL VIRTUAL WORLDS FROM MONOCULAR IMAGES OF URBAN ROAD TRAFFIC SCENES** submitted to the International Institute of Information Technology, Bangalore, for the award of the degree of **Master of Technology** is a bona fide record of the research work done by **Ankita Christine Victor**, **IMT2014005**, under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

_____

Professor Jaya Sreevalsan Nair

Bangalore,
The 3rd of June, 2019.

SYNTHESIS OF THREE-DIMENSIONAL VIRTUAL WORLDS FROM MONOCULAR IMAGES OF URBAN ROAD TRAFFIC SCENES

Abstract

Three-dimensional (3D) modeling of urban scenes has warranted well deserved interest in entertainment, navigation, urban planning and simulation. These fields require realistic looking, water-tight models. While automated processes for 3D reconstruction exist, the output of these that are typically either sparse point clouds or blobby models, which lack the detail and finesse required. Modeling realistic scenes is still predominantly a manual process, relying largely on 3D artists.

In this thesis, our motivation is to generate 3D constructions of urban road traffic scenes from monocular images. Cameras mounted on vehicles are capable of providing sequences of images of the road scene, which are datasets of our interest. The challenges in generating 3D models from monocular images stem from the incompleteness of the information of the 3D space from the image, which is a two-dimensional (2D) projection of the scene in the projection plane of the camera. Our approach in utilizing the available pixel information from an image to synthesize the 3D scene is to arrive at a workflow/pipeline which combines solutions from machine learning, computer vision, and computer graphics. Given a monocular image, our proposed pipeline uses deep learning to generate a dense depth map, an inverse projection to correct for perspective distortion in the image, comparisons of positions to correct errors in depth and a rendering engine to load and display 3D models belonging to a particular type at the right position in world space. Thus, our proposed pipeline largely eliminates the need for human-in-the-loop for 3D modeling of scenes. Our proposed method, if integrated into

a modeling software, could be used to significantly speed up the process of modeling virtual environments,

Acknowledgements

I would like to express my sincere appreciation and gratitude to my advisor, Professor Jaya Nair. Her encouragement and guidance has been invaluable to me over the last two years and has helped me become a passionate researcher. Her energy and creativity in identifying unexplored problems and devising solutions is a constant source of inspiration to me. This thesis would not be possible without her mentorship.

I thank both Professor Jaya Nair and Professor T K Srikanth for starting me on this journey into the world of computer graphics and being on my thesis examination committee, and Professor Dinesh Babu Jayagopi for being on my thesis examination committee and for his insight into machine learning. I thank all the professors I've learned under at IIITB. They have all made me grow as a student and a researcher. I also thank the Machine Intelligence and Robotics Center (MINRO) for funding my thesis.

I thank my brother Ashish, my colleagues Tarun Dutt and Nihal Kudligi for being my sounding boards and allowing to me articulate my thoughts, and ideas, and my friends who have been there with me through these five years of college.

Finally, I wish to thank my parents. For all the lessons you took me to, the unconditional love, faith and support, for being my rocks in this world. You nurtured my creativity and helped me become the person I am today. Thank you for everything.

Contents

List of Figures

List of Tables

List of Abbreviations

3D . . . . . Three Dimensional

2D . . . . . Two Dimensional

SfM . . . . . Structure from Motion

AR . . . . . Augmented Reality

VR . . . . . Virtual Reality

CRF . . . . . Conditional Random Field

CAD . . . . . Computer Aided Design

GPS . . . . . Global Positioning System

INS . . . . . Inertial Navigation System

NURBS . . . . . Non-uniform Rational Basis Spline

GUI . . . . . Graphical User Interface

SSIM . . . . . Single Scale Structural Similarity

DLT . . . . . Direct Linear Transform

SVD . . . . . Singular Value Decomposition

HDR . . . . . High Dynamic Range

"All problems in Computer Graphics can be solved with a matrix inversion."

Jim Blinn

CHAPTER 1

INTRODUCTION

Constructing a three dimensional (3D) model of a scene from a two dimensional (2D) image is a fundamental problem in computer vision and graphics. The technology has applications in entertainment, digital mapping, urban planning and training simulations for autonomous vehicle software. In the entertainment industry, many animation features and video games are set in 3D worlds either inspired by or directly modeled on real cities. Digital mapping applications such as Google Maps have entered the third dimension and allow users to explore around extruded cities, as shown in Figure FC1.1. Urban planning looks to 3D reconstruction of the urban environment to form a basis for surveying and future development. Simulation of autonomous vehicle software uses 3D scenes to generate realistic, urban environments for testing. Simply put, there is a demand for high-quality 3D content.

Attempts to automate the process of 3D construction use a combination of sensors and modalities. The data capture process for accurate 3D reconstruction requires an elaborate and expensive set up with multiple cameras and/or sensors. High precision 3D scanners are capable of generating accurate 3D models of single objects, but the technology to scan entire environments is still limited to noisy methods like Light Detection and Ranging (LiDAR). With the onslaught of digital camera devices and the Internet, the Web has become a source of billions of images. Using camera images as

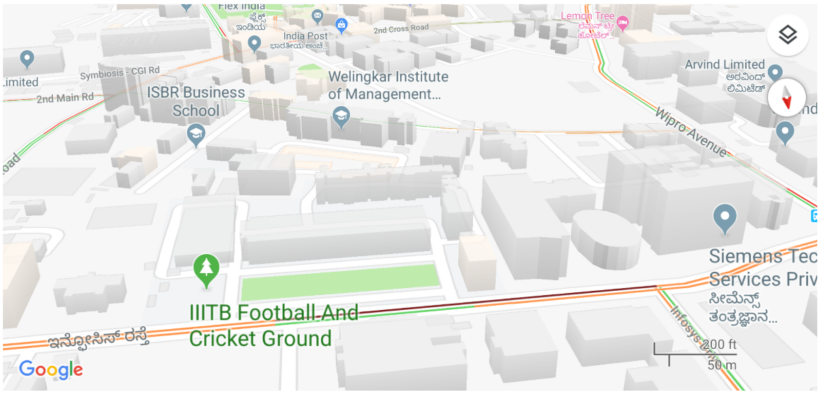


Figure FC1.1: ¿ Google Maps 5.0 with extruded details.

the starting point for modeling makes data capture process less complex. Additionally, machine learning and computer vision provide the technology to augment these 2D, RGB images with 3D information and create new uses for monocular images.

Despite interest in the study of the urban environment, in the past urban geography has been regarded as less topical in comparison to the other more established fields [30]. This can be explained partly by the nature of the urban environment that comprises a number of distinct elements from landscapes to transportation networks and various other socio-economic scenes [30]. However, as autonomous vehicles and robots are gaining momentum, augmented reality (AR) and virtual reality (VR) are becoming ubiquitous, and high-quality 3D content is supported by commercial mobile phones there is increased interest in the construction of urban geography for various purposes. One example relevant to this thesis is autonomous vehicle simulation. Researchers at the RAND Corporation has estimated that self-driving vehicles may have to drive up to 17.7 billion kilometers (km) before one can have reliable statistics on their safety to compare to human drivers [20]. While one could record 17.7 billion km of footage, modeling virtual worlds that can be manipulated at will to produce a variety of scenarios is more advantageous.

Figure FC1.2: A scene of urban road traffic set in Detroit from Ubisoft's game The Crew. Image credit: Detroit Metro Times

So far, no method has been able to perfectly reconstruct large urban scenes from captured data. As a result, scene modeling still remains a manual process relying on 3D artists. Most 3D artists typically follow the same workflow that usually starts with gathering inspiration and ideas from the Internet. Although it is not always necessary to use a reference image during modeling, it makes the process much easier. In addition, many video games, animation features, and simulation sequences are set in the real world, which makes reference to such images time saving and useful.

Once an idea of the intended environment is established, artists typically make a general concept of the scene they want to create as a simple sketch or directly in 3D software by setting up a camera and placing the basic objects like cuboids at the right positions. One of the important factors in setting up a realistic scene is scale. It is important to keep objects to scale even if the scene is just a mock-up of simple geometric objects. Once the location and the scale of every object in the scene are fixed the artist goes about replacing the basic objects with 'real' objects. The reference image comes in use at this point. Cuboids and other simple objects and replaced with 3D models of buildings, cars and other objects.

After the basic environment is modeled, the scene is lit with a main light and then detailed by adding smaller environment elements such as street lamps, bins, pavement

tiles, and street signs. More lights are added into the scene, objects are textured, and given material properties. It is easier to tweak materials to look good with lights rather the other way around. The scene is then rendered and post-processed in editing software. An example of one such modeled, 3D scene is shown Figure FC1.2. The scene is taken from a racing video game called that features realistic racing environments based on real American cities.

3D modeling of realistic environments is offered as a service. Two important aspects of the creation process is how close in appearance an artist can make the virtual environment look when compared to the real world, and how fast these scenes can be modeled. Sophisticated modeling tools like Blender, Maya, Cinema4D and 3ds Max are industry standard software packages used for 3D printing, animation, gaming, architecture, and industrial design. However, these tools assist in realizing an artist's creative ideas rather than the creation process itself. To this regard, a technology that is capable of delivering a usable, realistic 3D environment with minimal human input is an interesting focus of research that we believe has potential.

To put it concisely, there is demand for production ready 3D content that can be used as is in video games and other environments and much of this 3D content is based off images of real world scenes. We then pose the following research question: Is it possible to design a system for automated modeling of a scene depicted in a monocular image input, with the goal of improving modeling efficiency?

To address this question we propose a workflow that uses convolutional neural networks to estimate depth, semantic labels, and projection matrix calibration to synthesize a scene based on a monocular image. Our proposed workflow automates the initial steps of 3D scene modeling described above up to the stage where the primary objects on the scene have been placed and the scene is lit with the main light. There are several challenges to overcome in order to realize such a system including noisy depth estimation

and recovering the world space positions of objects in the image.

1.1 Our Contributions

To address our research question, we have identified three main modules of a workflow, namely, depth estimation, perspective correction, and model loading, as shown in Figure FC1.3. Our workflow exclusively synthesizes 3D virtual worlds of urban road traffic scenes with paved roads, parked/stationary and moving traffic, people, surrounding trees, and construction.

While depth estimation as an area of research is used mostly with autonomous navigation or point cloud reconstruction in mind, we propose using a depth estimation technique given by Godard et al [17] to recover depth from a monocular image. While it is almost intuitive for a human to infer the depth of various objects in a monocular image and construct a scene using the image as a reference, it is much harder for a computer to do so. The depth estimation technique used here is trained on stereo data, runs on monocular images, and produces high quality, dense depth maps. We have chosen this technique as it does not train on ground truth depth information and is capable of generalization to a certain extent. Moreover, the model can be retrained on easily acquirable stereo images allowing the workflow to be used on a wider class of scenes.

Images captured by a camera are projected perspectively. This means that parallel lines in the world space are no longer parallel in the image space. This affects the ability of a computer system from identifying the orthographic positions of objects in a scene. To recover the X-coordinate of these objects we modify a camera calibration method, Direct Linear Transform (DLT), to compute a mapping between 'known' 2D and 3D points assuming that all objects of interest are on the ground plane.

At the time of scene construction, ground truth semantic labels, and segment bound-

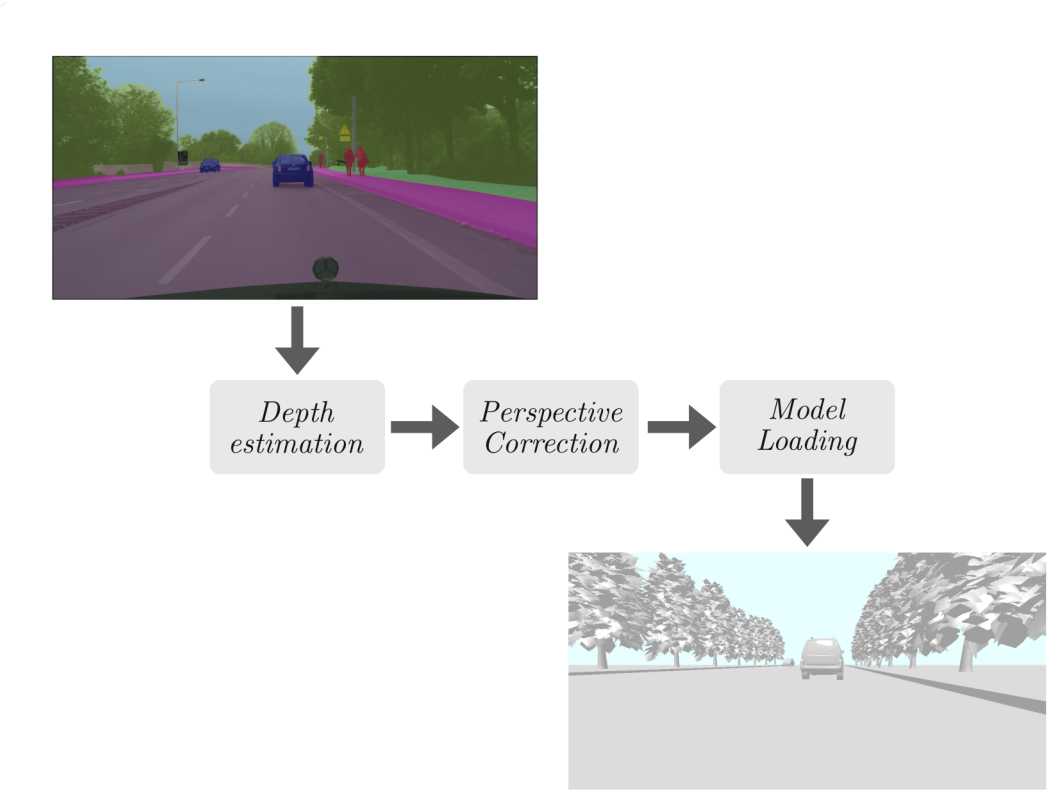


Figure FC1.3: Schematic of our proposed workflow. From the input image, depth is estimated for every pixel and aggregated over objects, and object positions in world space are obtained by perspective correction. A prefabricated 3D model of matching class is then loaded into the scene at the computed position.

aries are passed to a module of our workflow which computes the average depth for each object and its position before projection. A 3D model belonging to the same semantic class is placed in the scene at the computed position. Our prototype application only 'type matches' objects in images to the 3D models that are finally loaded. There are no exact matches here, for example, if a woman is walking, we still approximate with a 3D model of a man, suggesting a match of *human being* in the scene. Similarly a generic sedan mesh model for any *car* in the scene. A sample outcome can be seen in Figure FC1.4. Our proposed workflow constructs an approximate 3D virtual worlds from a monocular image and we have not found any prior work on such workflows.

The rest of the thesis is structured as follows: Chapter 2 provides an overview of 3D

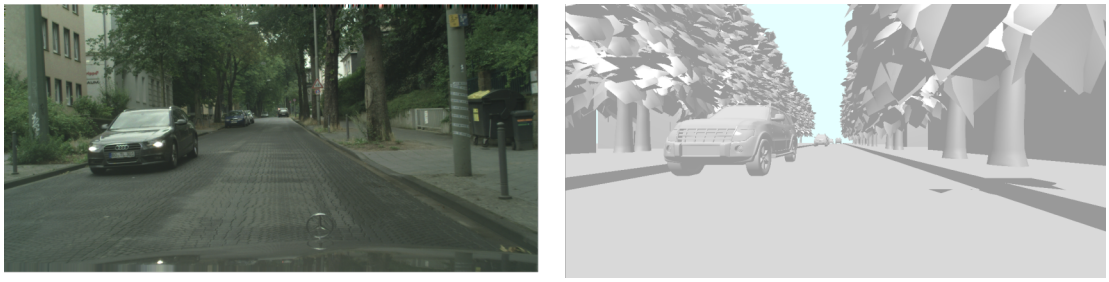


Figure FC1.4: Sample input and corresponding output obtained via our workflow.

modeling and reconstruction. Chapter 3 describes the depth estimation for monocular images proposed by Godard et al. [17]. Chapter 4 describes our inverse projection method. Chapter 5 describes our 3D synthesis pipeline and results on monocular images from the Cityscapes dataset. Chapter 6 presents our conclusions and discusses several directions for future work.

CHAPTER 2

BACKGROUND

3D models represent a physical object using a set of geometric primitives, which are usually points in 3D space that are connected to form triangles, quads, lines, or curves. 3D models can be created by hand, algorithmically (e.g. procedural modeling), scanned, or via photogrammetry. Specialized software is used by 3D artists to model objects. 3D scanners generally output point clouds, which measure a large number of points on the scanned object(s). While point clouds can be directly rendered and visualized using methods such as QSplat [27], these are often converted to mesh models or non-uniform rational basis spline (NURBS) patches through a process of surface reconstruction.

3D reconstruction from images and photogrammetry has been a popular area of research primarily in computer vision and visualization. Inferring the geometry of 3D scenes from 2D images is a challenging task because the image formation process is not generally invertible: from its projected position in a camera image plane, a scene point can only be recovered up to a one-parameter ambiguity corresponding to its distance from the camera. Much research effort has been devoted to the modeling of man-made environments using a combination of sensors and other visual modalities. One natural choice to satisfy the requirement of geometric modeling is the combined use of active range scanners such as LiDAR and digital cameras [3]. The captured data, which can be an image sequence, multiple viewpoints of the same scene with elevation data such

as LiDAR, is then used to solve the reconstruction problem for 3D visualization. The common goal in most cases is the accurate reconstruction of a scene to obtain models that will be useful for visualization, navigation, and other quantitative or qualitative analysis. Although we look at scene construction (creating an approximate scene with the same types of objects at the right position) rather than reconstruction, solutions to the latter have some overlap with the former in that they rely on depth inference and camera pose estimation. In our work, we have examined the former, and the latter is listed as future work.

2.1 3D Modeling and Sketchpad

3D modeling is simply defined as the process of creating a 3D, digital representation of an actual object or scene. In 1963, Ivan Sutherland wrote a revolutionary program called Sketchpad [32], for which he received the Turing Award in 1988. Sketchpad is considered to be the forefather of modern computer aided design (CAD) and 3D modelers. A Sketchpad user could sketch directly on a screen with a 'light pen'. Sketchpad was an interactive system. Using the light pen and the input buttons on it, a user could draw directly on the screen. The program supported the drawing of points, line segments, and arcs as basic elements, as well as allowed these to be saved as master symbols, which could be copied or instanced. This facility was used to create compound geometry.

Sketchpad supported explicit constraints, which are added to entities after they were drawn, as well as implicit constraints, which are created while entities were being drawn. For example, snapping, a feature that is still used in 3D modeling and illustrating software, originated from the Sketchpad. If a user, while drawing a line, brought the cursor close to the endpoint of another line, it would snap to the line being drawn

Figure FC2.1: Ivan Sutherland using Sketchpad on an MIT Lincoln Labs TX-2 computer [36].

to that endpoint and would remember that the two are connected during editing and moving. Sketchpad included seventeen different types of constraints, including vertical, horizontal, perpendicular, coincident, parallel, aligned, equal size, and more. These constraints could be combined to create more complex relationships between geometric entities. An image of Sutherland using Sketchpad is seen in Figure FC2.1.

Soon after Sutherland, Timothy E Johnson submitted his Master's thesis describing Sketchpad III, a 3D version of Sketchpad. At about the same time, Lawrence G. Roberts submitted his Ph.D. thesis, in which he had added support to Sketchpad for 3D solids, including assemblies and real-time hidden line removal. Sutherland, Johnson, and Roberts each made 16 mm movies, to demonstrate their work [36].

Sketchpad is important to be mentioned in the context of 3D modeling as it gave users the ability to create the first digital models ever. Additionally, it pioneered the concept of a graphical user interface (GUI). The sketching modules for future programs like SolidWorks were very much like Sketchpad. Today, the majority of 3D modeling for video games and movies is done using specialized software whose origins trace back

to Sketchpad.

2.2 3D Scanning

3D scanning is one of the most popularly used methods for acquiring 3D models. The output of a 3D scanner is a point cloud of geometric samples on the surface of the subject which can then be used to obtain the shape of the subject via some reconstruction method. 3D scanners, like cameras, have a field of view and can collect information about non occluded surfaces. Similar to how a camera collects color information about non-occluded surfaces within its field of view, a 3D scanner collects distance information, and color if needed. This allows the three dimensional position of each captured point to be identified.

For most scenarios, a single scan will not produce a complete model of the subject. Multiple scans from different viewpoints and directions are required to obtain information about all sides of the subject and construct a 360° model of the same. These scans are brought into a common frame of reference by a process called alignment or registration and are then merged to create a complete 3D scan of the subject. This entire process, going from scans to an aligned 3D representation, is known as the 3D scanning pipeline [5].

There are a variety of technologies for digitally acquiring the shape of a 3D object. Curless [10] divided them into two types: contact and non-contact. Non-contact solutions can be further divided into two main categories, namely, active and passive. Contact scanners examine the subject through physical touch, while the object is in contact with or resting on a stable, flat surface plate and is polished to a specific maximum of surface roughness. Contact scanners are typically used for digitizing clay models in the animation industry. Non-contact active scanners emit radiation or light and detect its reflection or radiation passing through an object in order to scan the object or

environment.

2.3 Reconstruction From Images

Non-contact passive 3D construction solutions do not emit any radiation but instead rely on detecting reflected ambient radiation — typically visible light. In most cases, passive hardware is a simple digital camera. Stereoscopic systems employ two cameras, slightly apart, looking at the same scene. By analyzing the slight differences between corresponding pixels in each camera image that is the disparity, it is possible to determine the distance at each point in the images. Photometric systems use a single camera, and take multiple images under varying lighting conditions to recover the surface orientation at each pixel. Other methods take multiple single camera shots of the same object from multiple views to create a reconstruction.

Structure From Motion

Structure from Motion (SfM) is based on the idea that given a scene depicted using two or more 2D views, a 3D point can be reconstructed by triangulation. SfM allows projection matrices and 3D points to be computed simultaneously using only corresponding points in each view. Given n projected points in m images, $U_{ij}j$, where $i \in 1...m, j \in 1...n$ represents the n^{th} point in the m^{th} image, the goal is to find both projection matrices $P_1...P_m$ and a consistent structure in 3D coordinates $X_1...X_n$ that relates U_{ij}. The early self-calibrating metric reconstruction systems [4, 11, 25] served as some of the first systems on 3D reconstruction from unordered Internet photo collections [31], and urban scenes [26].

Pollefeys et al. [26] introduced a large-scale, 3D reconstruction system to deliver models in the form of textured polygonal meshes. Their system incorporates data from a Global Positioning System (GPS) and an Inertial Navigation System (INS), if avail-

able, and uses SfM otherwise. Their core algorithm operates on frames from the video of a monocular camera as it moves in space. Salient image features are tracked across frames to provide 2D tracks that potentially belong to a 3D point following which they use 3D tracking/geo-location or SfM to estimate the camera pose. The feature points are examined using sparse scene analysis to determine three orthogonal sweeping directions (one for the ground and two for the facades). Stereo depth estimation computes depth maps from the obtained camera poses using a multi-view plane sweeping stereo algorithm. Lastly model generation creates a triangular mesh for each fused depth map and determines the texture mapping. It also removes duplicate representations of the same surface and fills some of the holes. The results of their work are seen in Figure FC2.2

Figure FC2.2: Result from more than one camera aided by GPS/INS data by Pollefeys et al. [26]

Incremental SfM is a popular strategy for 3D reconstruction from unordered image collections. In a typical incremental SfM system, two view reconstructions are first estimated upon successful feature matching between two images and 3D models are then reconstructed by initializing from good two-view reconstructions, repeatedly adding matched images, triangulating feature matches, and bundle-adjusting the structure and motion [35]. Incremental SfM is a popular strategy from reconstructing geometry from large-scale community photo collections. There exist approaches which source images

from the Internet, match the images to a specific scene and use SfM to generate sparse 3D models [2, 13, 31]. These follow the similar process of choosing two images to seed the reconstruction, then adding cameras using pose estimation, finding 3D points via triangulation followed by a non-linear refinement and different views are incrementally added. A reconstruction of Rome from an unstructured image collection on the Web by Agarwal et al. [2] is shown in Figure FC2.3

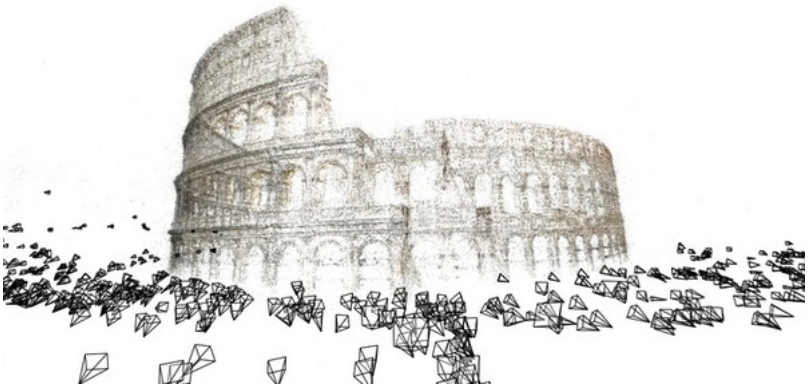


Figure FC2.3: Point cloud reconstruction of Rome using the incremental SfM technique by Agarwal et al. [2]

The approach proposed by Cohen et al. [8] optimizes SfM reconstruction by discovering symmetries and repetitions in the scene structure from multiple images and imposes these symmetry constraints to improve the accuracy of SfM algorithms.

It must be noted that given an appropriate number of views, SfM can reconstruct a sparse 3D view of any given scene. This differs from the goal of this thesis whose target scene is very specific to urban traffic and the goal of which is to mimic rather than reconstruct in exact detail. Although the work by Pollefeys et al. [26] seems closest in spirit to our work, it uses multiple views and a video sequence as well as GPS/INS data. Moreover owing to the 'holes' in the 3D reconstruction, the final modeled scene can be used more as a reference and not as a virtual environment in VR, games or other simulations.

Joint Semantic Segmentation and 3D Reconstruction

Joint semantic segmentation and 3D reconstruction approaches look at how image segmentation and dense 3D reconstruction can contribute to each other's task by constraining the solution using priors to yield smoother 3D reconstructions and/or segmentation. Lempitsky and Boykov [23] utilize the surface area as regularization prior and obtain the final surface representation indirectly via volumetric optimization. However, this returns only a binary decision on the occupancy state of a voxel, that is occupied or free, and does not take into account class-specific geometry. This leads to the motivation behind joint semantic segmentation and 3D reconstruction.

Kundu et al. [22] used an approach for joint inference of 3D scene structure and semantic labeling for forward moving monocular image sequences. Starting with monocular image stream, their framework produces a 3D volumetric semantic and occupancy map, which they claim is more useful than a series of 2D semantic label images or a sparse point cloud produced by traditional semantic segmentation and SfM pipelines respectively.

Kundu et al. derive a Conditional Random Field (CRF) model defined in the 3D space, that jointly infers the semantic category and occupancy for each voxel. This joint inference in the 3D CRF allows for more informed priors and constraints, which they have stated is otherwise not possible if solved separately using their traditional frameworks. Class specific semantic cues constrain the 3D structure in areas, where multi-view constraints are weak. Their reconstruction results are as shown in Figure FC2.4.

Hane et al. [18] used a multi-label volumetric segmentation framework that assigns some object class or a free-space label to voxels. The semantic labeling of each voxel influences the associated appearance of that voxel in 3D space, and hence, can influence a spatial smoothness prior to generating an accurate 3D reconstruction. Their approach is influenced by the notion that a class-specific regularizer guided by image appearances

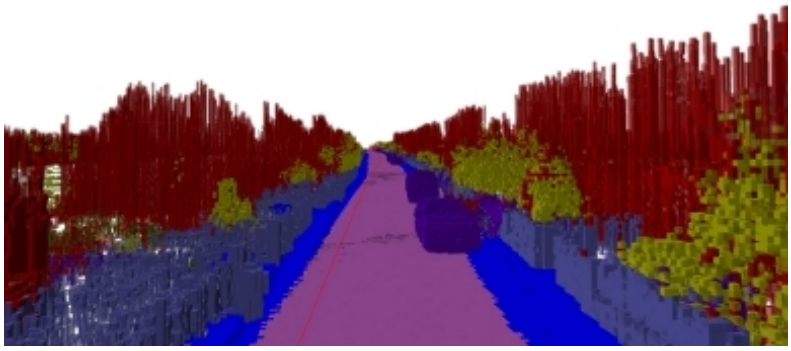


Figure FC2.4: Result of 3D reconstruction from monocular image sequence by Kundu et al. [22]. The final reconstruction is denser than traditional SfM methods.

can adaptively enforce spatial smoothness and preferred orientations of 3D surfaces, see Figure FC2.5.

Hane et al. proposed to learn appearance likelihoods and class-specific geometry priors for surface orientations from training data in an initial step. These data-driven priors are then used to define unary and pairwise potentials in a volumetric segmentation framework, complementary to the measured evidence acquired from depth maps. While optimizing over the label assignment in this volume, the image-based appearance likelihoods, depth maps computed using plane sweep stereo matching for each of the images, and geometric priors interact with each other yielding an improved dense reconstruction and labeling.

Figure FC2.5: Result of 3D reconstruction using class-specific geometric priors by Hane et al. [18]

Sengupta et al. [29] proposed an algorithm that generates an efficient and accurate dense 3D reconstruction with associated semantic labeling. The inputs to the algorithm are street level stereo image pairs acquired from a camera mounted on a moving vehicle. The depth-maps that are generated from the moving stereo pairs are fused into a global 3D volume. The street-level images are automatically labeled using a CRF framework that exploits the stereo images, and label estimates are aggregated to annotate the 3D volume, see Figure FC2.6. Semantic segmentation and reconstruction are not performed jointly.

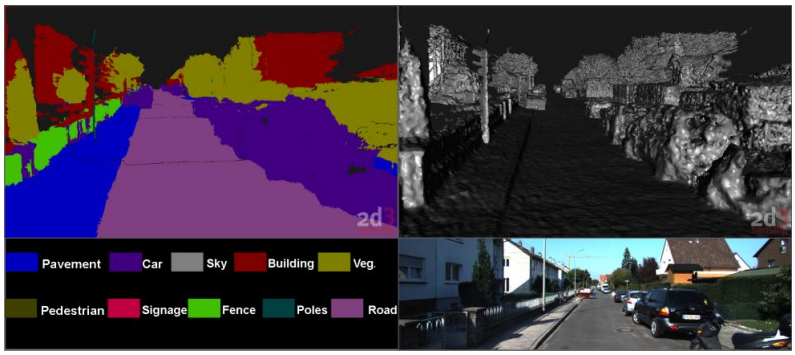


Figure FC2.6: Result of semantic labeling 3D reconstruction by Sengupta et al. [29].

The motivation behind most 3D reconstruction techniques described here so far deals more with gathering 3D information of the scene or building a navigable environment of a scene for autonomous machines and less to do with the actual scene or landscape modeling. An approximate 3D reconstruction whether dense or sparse is sufficient given that the goal is to aid general understanding of the scene by a system. Moreover, none of the described works here considers a single monocular image as input. The input is either multi-view, stereo, or an image sequence with one or more cameras and the output is either a sparse or dense reconstruction of the scene with the additional goal of semantic voxel labeling.

This thesis differs in that it considers a single, monocular image and semantic labels as input, gets as output a complete 3D scene with 'stock' models, and is visually similar to that of the input image but is not a reconstruction of captured data. The synthesized

scene contains 3D models belonging to the correct semantic class but these models are not the exact instance of the semantic class in the image. This is acceptable given that the motivation here is to provide aid with automatic scene modeling inspired by real-world scenes and the synthesized 3D worlds can then be enhanced by a modeling artist or directly used for some purpose.

CHAPTER 3

DEPTH ESTIMATION OF MONOCULAR IMAGES

Depth estimation is a fundamental problem in computer vision and is a crucial step in the reconstruction of 3D scenes. When provided with accurate image correspondences, depth can be recovered deterministically for stereo images. Similarly, using a sequence of 2D images, 3D scene structure can be estimated by leveraging camera motion to determine a sequence of camera poses and in turn, estimate depth via triangulation of feature points from pairs of consecutive views. However, most algorithms able to recover depth from pairs of stereo images provide depth values only for these specific correspondence points in the viewed scene. This sparse 3D map can be sufficient for tasks such as navigation where the redundant information does not require a dense 3D map. In the case of terrain reconstruction, a dense map is required [6]. Additionally, such approaches rely on the assumption that stereo pairs or multiple observations of the scene under different lighting conditions or in time are provided. Yet the monocular case often arises in practice [12].

Depth estimation of a monocular image requires analysis of monocular depth cues such as line angles and perspective effects, known object sizes texture details, lighting, and atmospheric effects. Moreover, the task is inherently ambiguous, and a technically ill-posed problem: Given an image, an infinite number of possible world scenes may have produced it. Most of these are physically implausible for real-world spaces; there-

fore, the depth may still be predicted with considerable accuracy [12]. While humans do well at this task by exploiting perspective, scale relative to known objects, shadows, and occlusion, computationally estimating depth for a single, monocular image is an ill-posed problem.

Early approaches resorted to exploiting statistically meaningful monocular cues or features such as perspective and texture information, object sizes, object localization, and occlusions [7]. More recently, learning based techniques have seen a surge in popularity as well as success in depth estimation of monocular images. These have typically posed the task of monocular depth estimation as a supervised learning problem and attempt to directly predict the depth of each pixel in an image using models that have been trained on large collections of ground truth depth data that is obtained using specialized hardware [17]. Posing the problem as a supervised learning problem has its limitations in that a large collection of images and their corresponding pixel depths are required. Moreover, the trained models cannot be used to predict depth for scenes dissimilar from the training images.

Given that we explore the construction of 3D worlds from monocular, RGB images, a desirable solution to depth estimation would be one that accepts a single image as input, can be retrained without requiring ground truth depth, and is capable of generalization with respect to the image scene. A solution given by Godard et al. uses a fully convolutional model that does not require any depth data during training, and is instead trained to synthesize depth as an intermediate output. The authors take an alternative approach and treat monocular depth estimation as an image reconstruction problem during training by which neural net learns to predict the pixel-level correspondence between pairs of rectified stereo images that have a known camera baseline. Their formulation of the problem requires no ground truth pixel-depth data, operates on monocular images and generalizes to different images [17]. A detailed description of their method follows.

3.1 Method

The network architecture proposed in [17] performs end-to-end unsupervised monocular depth estimation with a novel training loss that enforces left-right depth consistency inside the network. The network first takes the left image of a rectified, stereo pair as input and uses the right image for supervision. The idea comes from the assumption that given accurate disparity estimates, one would be able to perfectly reconstruct the right view from the left view of a stereo pair and vice versa. Figure FC3.1 illustrates rectified, stereo image capture.

3.1.1 Depth Estimation as Image Reconstruction

The problem of learning based, monocular depth estimation can be expressed as given a single image I, to learn a function f that can predict the per-pixel scene depth, $d = f(I)$. Most existing learning based approaches treat this as a supervised learning problem, where they have RGB input images and their corresponding ground truth depth values at training. Here, the problem of depth estimation is posed as an image reconstruction problem during training. The intuition behind this is that, given a calibrated pair of binocular cameras, if the network can learn a function that can reconstruct one image from the other, then we have learned something about the 3D shape of the scene that is being imaged. At training time the network has access to two images, I^l and I^r, which correspond to the left and right RGB images of a calibrated stereo pair. Instead of trying to directly predict the depth as is done in the supervised case, the network learns a dense correspondence field d^r that, when applied to the left image, would enable reconstruct the right image.

The reconstructed right image $\tilde{I}^r = I^l(d^r)$. Similarly, the left can also be reconstructed from the right image and $\tilde{I}^l = I^r(d^l)$. Assuming that the images are projected

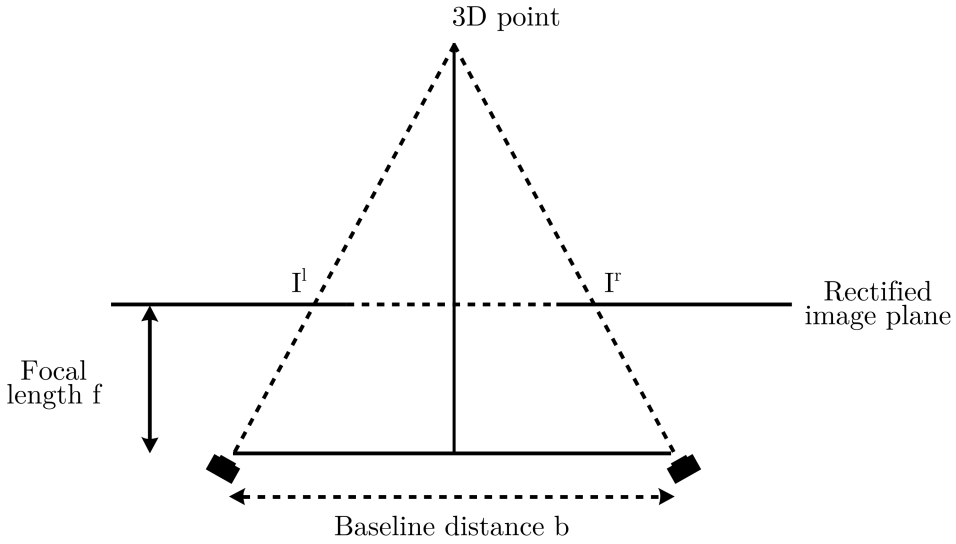


Figure FC3.1: Stereo image capture.

onto a common plane parallel to a line between optical centers or rectified, then d corresponds to the image disparity or distance between two corresponding points in the left and right image of a stereo pair - a scalar value per pixel that the model will learn to predict. Given the baseline distance b between the cameras and the camera focal length f, depth $\hat{d}$ can be trivially recovered from the predicted disparity, as

$$\hat{d} = bf/d \qquad \text{(Eqn 3.1)}$$

3.1.2 Neural Network

The network estimates depth by learning the disparities that warp the left image of a stereo pair to match the right one. The key insight of the method proposed in [17] is that by simultaneously inferring both left-to-right and right-to-left disparities using only the left input image, the network obtains better depths by enforcing them to be consistent with each other. The network generates the predicted image with backward mapping

using a bilinear sampler, resulting in a fully differentiable image formation model.

During training, the model is given a stereo pair and the target is one of the two images. This implies that the model must output an image governed by a reconstruction loss between the output and target image. Given that the goal is to generate a depth map, the model must output a disparity map. The network learns to generate per pixel disparities that produce say the right target image, by shifting the pixels from the left input image using an image sampler. A mesh grid with each pixel as a cell is shifted to the right (supposing the input image is the left image) according to the value in the disparity map. The final image is generated with backward mapping using a bilinear sampler on the left image using the transformed mesh grid where the floating point indexes are solved with bilinear interpolation.

A naive sampling of the left image to generate the right image produces a disparity map that is aligned with the target instead of the input image. However, the output disparity map must align with the input left image, since we want the depth map of the input image, meaning the network has to sample from the right image. Simply sampling the right image produces an input aligned disparity map; however, the inferred disparities exhibit certain artifacts and errors. [17] solves these artifacts by training the network to predict the disparity maps for both left and right views and sample from the opposite input images. This still only requires a single left image as input to the convolutional layers and the right image is only used during training. Using the left image to produce disparities for both images, improves quality by enforcing mutual consistency.

The network is a fully convolutional neural network and is composed of two main parts - an encoder and decoder. The decoder uses skip connections from the encoder's activation blocks, enabling it to resolve higher resolution details. Disparity predictions are outputted at four different scales, which double in spatial resolution at each of the

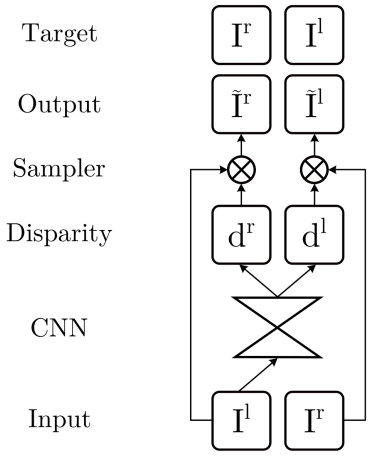


Figure FC3.2: Stereo Left-Right consistency.

subsequent scales. This improves the output resolution of the neural network. The network takes a single image as input and predicts two disparity maps at each output scale - left-to-right and right-to-left. Figure FC3.2 illustrates a summary of the network.

3.1.3 Loss Function

The loss at each output scale C_s is computed as

$$C_s = \alpha_{ap}(C^l_{ap} + C^r_{ap}) + \alpha_{ds}(C^l_{ds} + C^r_{ds}) + \alpha_{lr}(C^l_{lr} + C^r_{lr}) \qquad \text{(Eqn 3.2)}$$

where, C_{ap} is the appearance matching loss and pushes the reconstructed image to appear similar to the corresponding training input, C_{ds} is the disparity smoothness loss and imposes smoothness in the disparity map, and C_{lr} is the left-right consistency loss and prefers the predicted left and right disparities to be consistent. While each of the main terms contains both a left and a right image variant, only the left image is fed through the convolutional layers. The total loss C is computed as

$$C = \sum_{s=1}^{4} C_s \qquad \text{(Eqn 3.3)}$$

Appearance Matching Loss

The network generates an image by sampling pixels from the opposite stereo image using the disparity map. A sampler must take the set of sampling points (here the disparity map), along with the input feature map (here the input image) and produce the sampled output feature map (here the output image). The image formation model chosen in [17] uses the image sampler from Spatial Transformer Network [19] to sample the input image. The STN uses bilinear sampling where the output pixel is the weighted sum of four input pixels. A combination of single scale structural similarity (SSIM) index and l_1 norm is used to construct the appearance matching loss, C_{ap}. SSIM is a method used for measuring the similarity between two images as a quality measure of one of the images being compared, assuming the other image is regarded as of perfect quality.

$$L^{l_1}(P) = \frac{1}{N} \sum_{p \in P} |x(p) - y(p)| \qquad \text{(Eqn 3.4)}$$

is the l_1 loss where, where p is the index of the pixel and P is the patch; $x(p)$ and $y(p)$ are the values of the pixels in the processed patch and the ground truth respectively [37]. SSIM for pixel p is defined as,

$$SSIM(p) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \cdot \frac{2\sigma_{xy} + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \qquad \text{(Eqn 3.5)}$$

where μ_i is the mean of i, σ_i^2 is the variance of i, σ_{ij} is the covariance of i and j, $C_1 = (k_1 L)^2$ and $C_2 = (k_2 L)^2$ are two variables to stabilize the division with weak denominator, L is the dynamic range of the pixel-values, and $k_1 = 0.01$ and $k_2 = 0.03$ by default. The means and standard deviations are computed with a Gaussian filter with

standard deviation σ_G. The loss function for SSIM

$$L^{SSIM}(P) = \frac{1}{N}\sum_{p\in P} 1 - SSIM(p) \quad \text{(Eqn 3.6)}$$

C_{ap} compares the input image I^l_{ij} and its reconstruction $\tilde{I}^l_{ij}$, where N is the number of pixels,

$$C^l_{ap} = \frac{1}{N}\sum_{i,j} \alpha \frac{1 - SSIM(I^l_{ij}, \tilde{I}^l_{ij})}{2} + (1-\alpha)\left\| I^l_{ij} - \tilde{I}^l_{ij} \right\| \quad \text{(Eqn 3.7)}$$

The authors choose a simplified SSIM with a 3×3 block filter instead of a Gaussian and set α to 0.85.

Disparity Smoothness Loss

The disparities are encouraged to be locally smooth with an l_1 penalty on the disparity gradients ∂d. Depth discontinuities occur at image gradients, for example where one object occludes another (or another part of itself), or between adjacent faces of the same object. This cost is weighted with an edge-aware term using the image gradients ∂I,

$$C_{ds} = \frac{1}{N}\sum_{i,j} |\partial_x d^l_{ij}| e^{-\|\partial_x d^l_{ij}\|} + |\partial_y d^l_{ij}| e^{-\|\partial_y d^l_{ij}\|} \quad \text{(Eqn 3.8)}$$

Left-Right Disparity Consistency Loss

To overcome artifacts, the authors train the network to predict both the left and right image disparities, from only the left view as input to the convolutional neural network. To ensure coherence, an l_1 left-right disparity consistency penalty is used. This cost attempts to make the left-view disparity map equal to the projected right-view disparity

map.

$$C_{lr}^{l} = \frac{1}{N}\sum_{i,j} |d_{ij}^{l} - d_{ij+d_{ij}^{l}}^{r}| \qquad \text{(Eqn 3.9)}$$

The same equation is mirrored for the right-view disparity map.

3.2 Our Use and Results

A model trained on the Cityscapes dataset [9] was plugged into the pipeline to generate the depth map for the 3D construction. Figure FC3.4 shows results of a trained model on images from Cityscapes. Although the images used for testing the proposed pipeline were taken from Cityscapes, the same model was tested on arbitrary Web search result images of Indian roads for which assumptions of well-delineated infrastructure such as lanes, a small number of well-defined categories for traffic participants, low variation in object or background appearance and strict adherence to traffic rules are not largely satisfied [34]. The model is able to generalize and despite the differences in location, image characteristics, and camera calibration produces visually plausible disparity maps as in Fig. FC3.3. If the focal length and the aspect ratio are different from that of the training dataset, the depth map produced using Eq. Eqn 3.1 might not be reliable [17].

As mentioned, the input images used for reconstruction are taken from the Cityscapes dataset. For Cityscapes the camera baseline is 0.22m and the focal length is 2262 for a width of 2048 pixels. The disparities generated by the model are normalized by the image width, this is scaled by width to get the value in pixels. Finally,

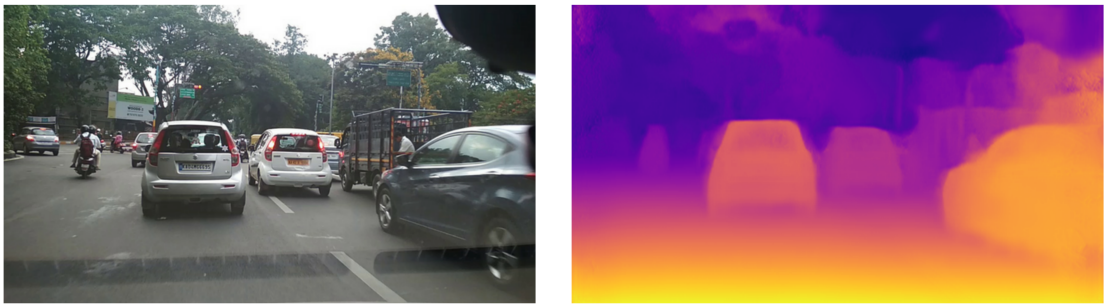
$$depth = \frac{0.22 \times 2262}{2048 \times disp} \qquad \text{(Eqn 3.10)}$$

3.3 Discussion

The model generates a good estimate of pixel depths. There are still some artifacts visible at occlusion boundaries due to the pixels in the occlusion region not being visible in both images [17]. Since the method mainly relies on the image reconstruction term, specular and transparent surfaces will produce inconsistent depths (as sometimes seen in vehicle windows). However, given its application here, where an average depth value is considered for every object, the numerical consequences of these artifacts do not matter much.

The described method requires rectified and temporally aligned stereo pairs during training. Retraining the model requires more easily available stereo image data as opposed to ground truth depth which is significantly harder to obtain and is often noisy. As shown above, a model trained on Cityscapes traffic scenes generalizes well enough to produce visually plausible disparity maps when tested on images on Indian traffic scenes.

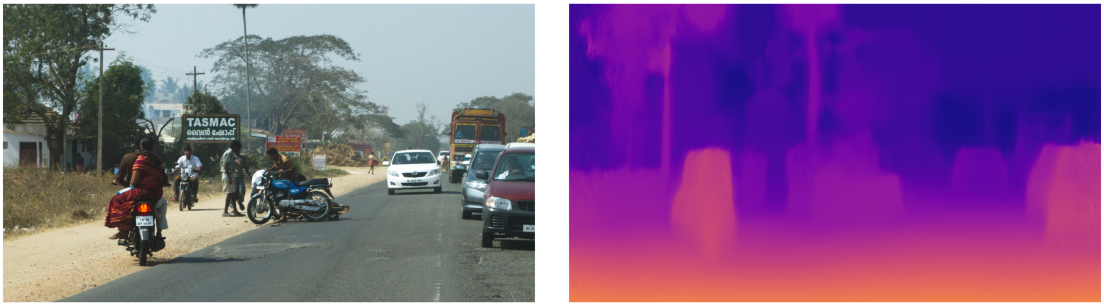
Comparable related work includes that of Eigen et al. [12], Liu et al. [24] and Gard et al. [14]. Eigen et al. [12] proposed a supervised, multi-scale network for prediction. A coarse-scale network first predicts the depth of the scene at a global level which is then refined within local regions by a fine-scale network with the coarse network's output passed to the fine network as additional first-layer image features [12]. Liu et al. formulated the depth estimation as a supervised, deep continuous Conditional Random Fields (CRF) learning problem, without relying on any geometric priors. Garg et al. [14], similar to Godard et al. [17], proposed an unsupervised model by training the network in a manner analogous to an autoencoder. At training time they consider a stereo pair and train a convolutional encoder to predict the depth map for the source image. They do so by generating an inverse warp of the target image using the predicted depth and known camera displacement in order to reconstruct the source image; the



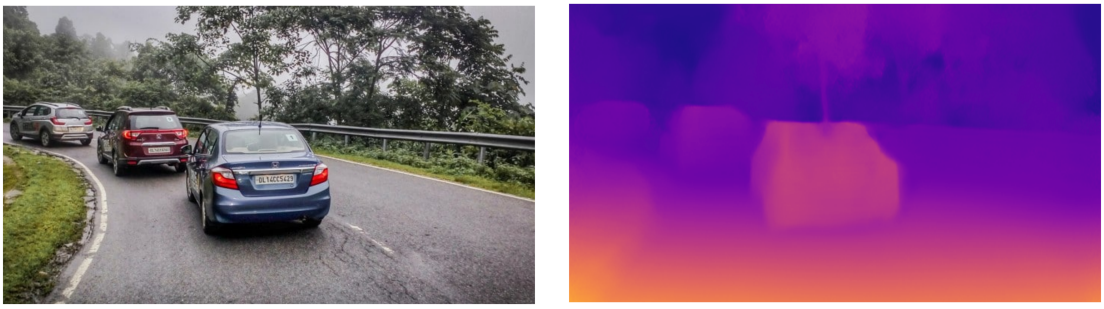
(a)

(b)

(c)

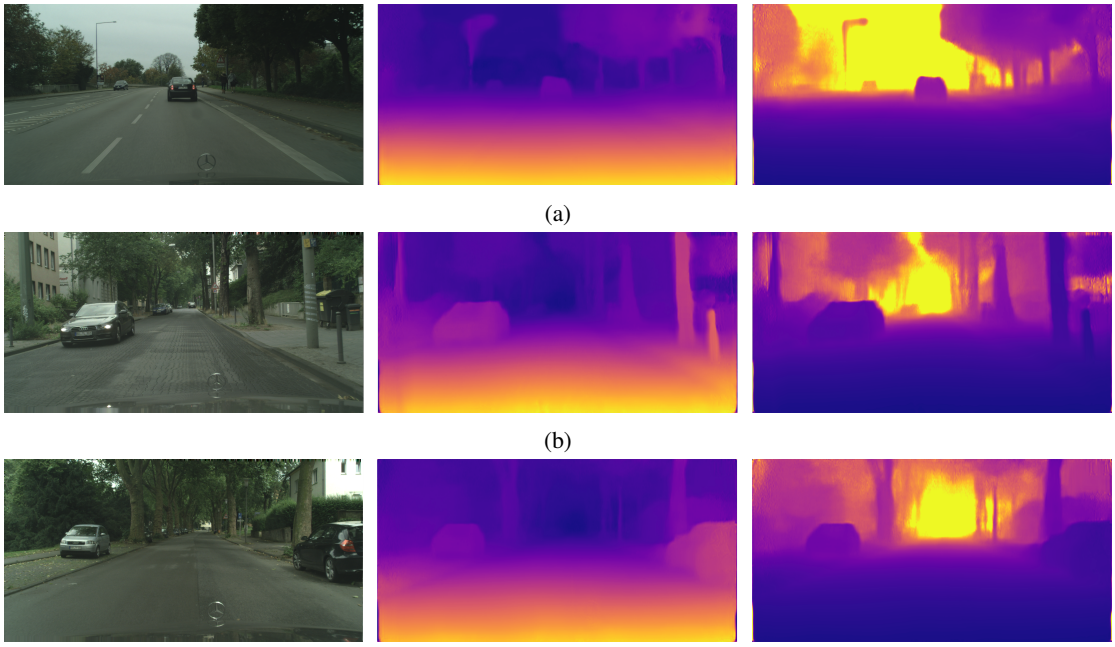


(d)



(e)

Figure FC3.3: Disparity map of arbitrary images outside of the training datasets.



(a)

(b)

(c)

Figure FC3.4: Images from the Cityscapes dataset, the output disparity map, and corresponding depth map.

photometric error in the reconstruction is used as the reconstruction loss for the encoder. The results in Table TC3.1 show that the method of Godard et al. [17] yields superior results over the others.

Our problem is to construct a 3D world from a monocular image. This method is easily plugged in as a module in the 3D synthesis pipeline. It was chosen for many reasons among which include its superior results over other supervised and unsupervised

Method	Supervised	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 125$	$\delta < 1.25^2$	$\delta < 1.25^3$
Eigen et al. [12] Coarse	Yes	0.214	1.605	6.563	0.292	0.673	0.884	0.957
Eigen et al. [12] Fine	Yes	0.203	1.548	6.307	0.282	0.702	0.890	0.958
Liu et al. [24]	Yes	0.201	1.584	6.471	0.273	0.68	0.898	0.967
Garg et al. [14]	No	0.169	1.080	5.104	0.273	0.740	0.904	0.962
Gordard et al. [17]	No	0.108	0.657	3.729	0.194	0.873	0.954	0.979

Lower is better | Higher is better

Table TC3.1: Comparison of results on KITTI [15] using the split of Eigen et al. [12], the predictions of Liu et al. [24] generated on a mix of the left and right images instead of just the left input images., the results of Garg et al. [14] and Godard et al. [17].

methods (see Table TC3.1), that it runs on a single image as input, can be retrained with more easily available stereo image pairs and generates visually plausible disparity maps for images that differ from training set scenes(see Figure FC3.3). With the emergence of stereo camera devices and new datasets like IDD [34] it would be a simple task to retrain the model for Indian traffic scenes and apply the same approach proposed in this thesis. [1]

[1] Godard et al. have recently proposed an improved version of [17], in [16] that can be trained on just monocular video. They have introduced three contributions: a minimum reprojection loss, computed for each pixel, to deal with occlusions between frames in a monocular video, an auto-masking loss to ignore confusing, stationary pixels, and a full-resolution multi-scale sampling method. Their model can be trained with monocular video data, stereo data, or mixed monocular and stereo data.

CHAPTER 4

PERSPECTIVE CORRECTION

Capturing a 2D image involves a perspective projection of a 3D world. Due to the projection, a captured image contains some distortion caused by a foreshortening factor and vanishing point projection. We can recover the world space position of an object in the scene along the Z-axis from the depth map obtained from the trained model described in Chapter 3. The position of an object along the X-axis cannot directly be recovered from the 2D image owing to perspective projection distortions by which parallel lines no longer remain parallel but appear to meet at some point.

Since the perspective view of the captured image distorts the actual shape of the road and other objects in the scene, which includes the width, height, and depth, or the X, Y, and Z components respectively, the image coordinates need to go through a pre-processing stage to correct or reverse the perspective distortion to obtain the undistorted world coordinates. In this chapter, we describe how we correct perspective distortion to obtain undistorted world coordinates of objects.

4.1 Perspective Projection Geometry

Perspective projection is a linear transformation where three-dimensional objects are projected onto a plane with the effect of distant objects appearing smaller than closer

objects. This also means that lines which are parallel in the original coordinate space appear to intersect in the transformed coordinate space or the projected image. For example, railway lines when pictured with perspective projection appear to converge at a single point in the distance, called the vanishing point. Photographic lenses, like the human eye, view the world in a similar fashion which is why perspective projection looks the most realistic.

The principal vanishing point is the vanishing point of all horizontal lines perpendicular to the picture plane. If, as is often the case, the projection plane is vertical, all vertical lines have no finite vanishing point on the picture plane. Perspective projection makes distant objects appear smaller to provide additional realism.

Many-to-one Mapping The projection of a point is not unique. Any point on the line OP in Fig. FC4.1 will be projected to the same point Q.

Scaling/Foreshortening When a line is parallel to the image plane, the effect of perspective projection is scaling. When a line is not parallel to the image plane, the effect of projective distortion is foreshortening, that is the dimension parallel to the optical axis is compressed relative to the frontal dimension.

Effect of Focal Length As f gets smaller, more points project onto the image plane or the camera becomes wide-angled. As f gets larger, the field of view becomes smaller or more telescopic.

Lines, Distances & Angles Distances and angles are not preserved is perspective projection. Parallel lines do not project to parallel lines (unless they are parallel to the image plane).

Vanishing Point Parallel lines in space project perspectively as lines that on extension appear to intersect at a some point in the image plane called vanishing point.

4.1.1 Pinhole Camera

A pinhole camera is a camera with a point-sized aperture and no lens. The pinhole camera model is a good approximation to the behavior of most real cameras, and so provides an entry into understanding perspective projection geometry. It describes the mathematical relationship between the coordinates of a point in three-dimensional space and its projection onto the image plane by an ideal pinhole camera, that is where the camera aperture is a point and no lenses are used to focus light.

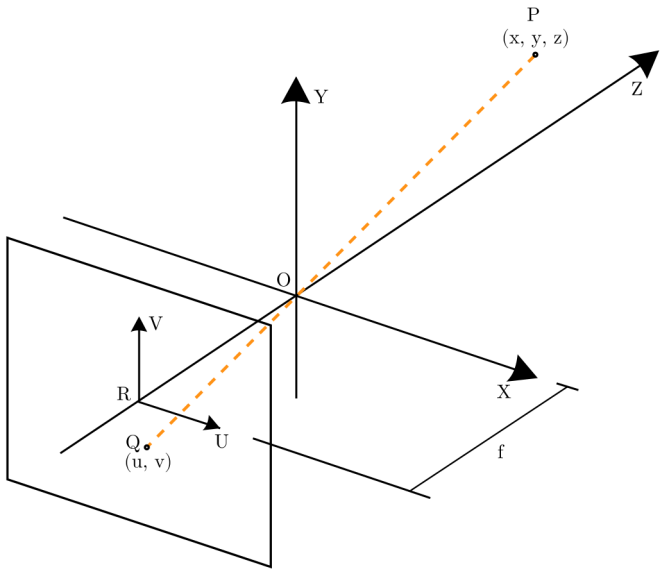


Figure FC4.1: Pinhole camera projection.

The projection of a pinhole camera is illustrated in Fig. FC4.1. The components of the figure are

- A 3D, orthogonal coordinate system with its origin at O. The three axes of the coordinate system are X,Y,X. The Z axis points in the viewing direction of the camera and is known as the optical axis, principal axis, or principal ray. The plane which is spanned by axes X and Y is the front side of the camera or principal plane.

- A pinhole camera with its aperture located at the origin O. The aperture of the camera is assumed to be infinitely small or a pinhole.
- An image or projection plane, on which the 3D world is projected through the camera aperture. The image plane is parallel to plane spanned by the X and Y axes and is located at a distance f from the camera aperture in the negative Z direction. This distance f is the focal length of the pinhole camera.
- A point R at the intersection of the optical axis and the image plane. This point is the image center.
- A point P somewhere in the world at (x, y, z).
- The projection line of point P into the camera aperture, that is the line joining points P and O.
- The projection of P onto the image plane, at Q or the point at which line PO intersects the image plane.
- A 2D coordinate system in the image plane, with the origin at R and axes U and V which are parallel to X and Y respectively. The coordinates of the projected point Q in this coordinate space are (u, v).

To understand how Q can be derived from P, we can use Fig. FC4.2 which shows the same scene as the Fig. FC4.1 but from above, looking down in the negative direction of the Y axis.

There are two similar triangles in Fig. FC4.2, that both have segments of the projection line as their hypotenuse. The catheti (sides adjacent to the right angle) of the left triangle are u and f and the catheti of the right triangle are x and z. Since the two

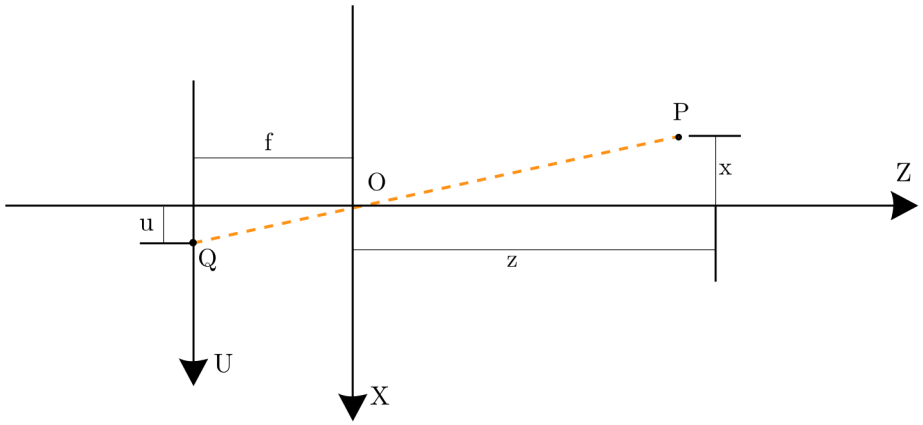


Figure FC4.2: Projection seen from Y axis.

triangles are similar,

$$\frac{u}{f} = \frac{x}{z} \text{ or } u = \frac{fx}{z} \qquad \text{(Eqn 4.1)}$$

Similarly,

$$\frac{v}{f} = \frac{y}{z} \text{ or } v = \frac{fy}{z} \qquad \text{(Eqn 4.2)}$$

This equation can be summarized as

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{f}{z} \begin{pmatrix} x \\ y \end{pmatrix} \qquad \text{(Eqn 4.3)}$$

Looking at Fig FC4.1 of the pinhole camera model, it is clear that as one moves further along the Z axis, the value of z will increase. From Eq. Eqn 4.3, it can be seen that an increase in the value of z will cause the object (of width u and height v) to reduce

in size and this phenomenon is known as a foreshortening factor. A vanishing point, on the other hand, occurs when a set of projected parallel lines appears to converge and intersect at a point, see Fig. FC4.3.

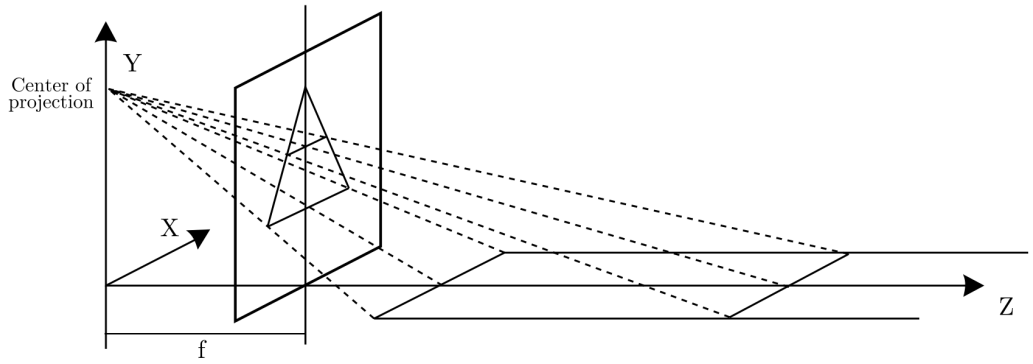


Figure FC4.3: Vanishing point as a result of perspective projection.

4.2 Perspective Correction

To synthesize a 3D world from an image, we require four main parameters —the semantic class of the object and its position in world space described by X, Y, Z. The type of objects present in the scene is obtained from a semantic segmentation of the image. Merely using the extents of a bounding box drawn around each object segment will be insufficient to recreate the scene in 3D given that the image has been captured in perspective projection. The dotted orange lines in Fig. FC4.4 show how the parallel lines of the road are no longer parallel. Similarly, the parked cars that would line up in a straight line in the real world, no longer look they are along a line parallel to the sidewalk. The imagined bird's eye view in Fig. FC4.5 is the ideal arrangement of the scene during the 3D world synthesis.

An accurate synthesis of the corresponding 3D world will require that the image

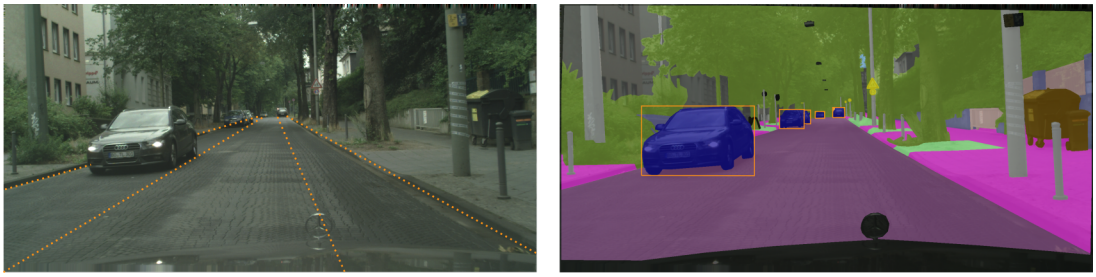


Figure FC4.4: An image from Cityscapes and its segmented image.

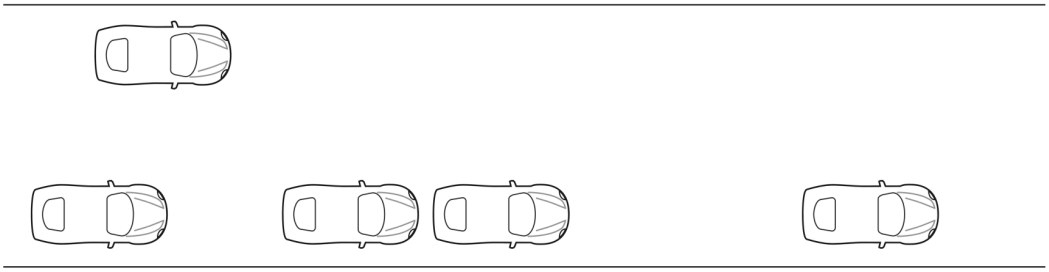


Figure FC4.5: Imagined bird's eye view of the road scene in Fig. FC4.4.

coordinates of each object first be remapped to its original world coordinates which can then be used to arrange the scene in 3D. The neural network described in Chapter 3 yields a dense depth map for an input image. Values within each object segment can be averaged to yield a single depth for each object. In other words, we have already recovered the Z coordinate of every object of interest in world space. Since the scenes in consideration are that of urban roads, we can make the assumption that all objects of interest —vehicles, vegetation, buildings— lie on the ground plane, or that the Y coordinate in world space for each object is 0. The only coordinate that remains to be found is the X coordinate.

In effect, to construct a 3D world out of an image scene we seek to derive the value of some matrix P where

$$U_i = PX_i \qquad \text{(Eqn 4.4)}$$

where U_i is a set of projected image coordinates, X_i is the corresponding set of coordinates in world space and, P is a projection matrix that transforms every X_i to U_i. Eq. Eqn 4.4 can be expanded as

$$\begin{pmatrix} u_i \\ v_i \\ 1 \end{pmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} \quad \text{(Eqn 4.5)}$$

Since all objects of interest have been assumed to be on the ground plane, $y_i = 0$, Eq. Eqn 4.5 can be simplified to

$$\begin{pmatrix} u_i \\ v_i \\ 1 \end{pmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{14} \\ p_{21} & p_{22} & p_{24} \\ p_{31} & p_{32} & p_{34} \end{bmatrix} \begin{pmatrix} x_i \\ z_i \\ 1 \end{pmatrix} \quad \text{(Eqn 4.6)}$$

Given a point U_i in the image space, z_i can be obtained by look up from the corresponding depth map, and x_i can be estimated as shown in Fig. FC4.6 and accordingly scaled in the graphics pipeline. The intuition is that the base of the image represents objects closest to the camera, and the points along this line can be assumed to be undistorted by perspective projection.

Once we have a set of point correspondences U_i and X_i we can derive the matrix P using some approximation method.

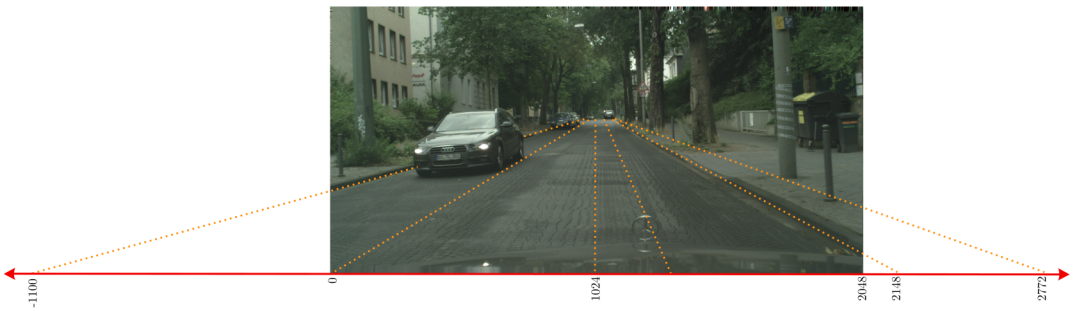


Figure FC4.6: A way to 'guesstimate' x_i.

4.2.1 Direct Linear Transform

This problem of determining the camera matrix from know scene points and projections is called the resection problem. Direct Linear Transform (DLT) is a popular method for determining camera calibration [1]. DLT uses a set of control points in the image space whose world space coordinates are already known. The goal is essentially to calculate the mapping between the 2D image space coordinates and the 3D object space coordinates. The correspondence between 3D and 2D takes the form of a 3×4 projection matrix P and is expressed by Eq. Eqn 4.4. While DLT is typically used to estimate camera pose, here we use it to simply find a mapping between our known point correspondences.

From Eq. Eqn 4.4, $P = K[R\ t]$, where K is a 3×3 matrix, R is a 3×3 rotation matrix, and t is a 3×1 vector. The 3×4 matrix $[R\ t]$ encodes the orientation and position of the camera with respect to a reference coordinate system. Given a 3D point in homogeneous coordinates X the product $[R\ t]X$ can be interpreted as the 3D coordinates of the scene point in the camera coordinate system. The 3×3 matrix K transforms the image plane in $\mathbb{R}^3$ to the real image coordinate system with unit pixels [33], see Figure FC4.7. At least 6 point correspondences are needed in order for the problem of finding the 3×4 matrix to be well defined.

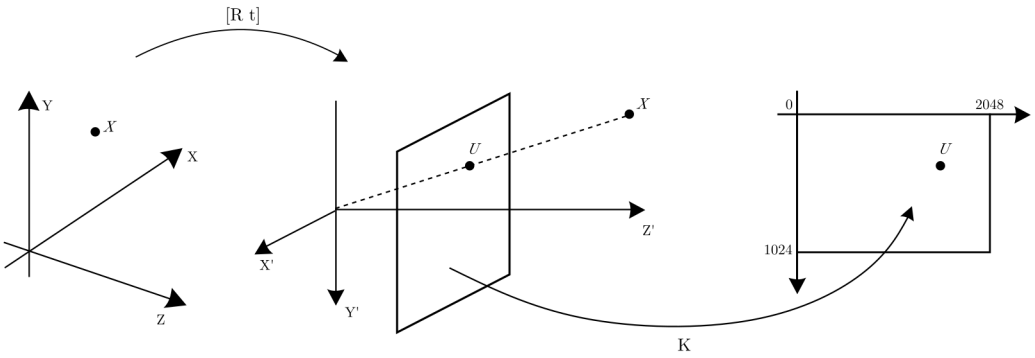


Figure FC4.7: Transformations involved in projection.

Given our assumption that all objects of interest lie on the ground plane or $y_i = 0$, the point transformation is simplified to Eq. Eqn 4.6. This corresponds to the 2D case of DLT where the solution matrix has dimension 3×3. This is additionally advantageous because a 3×3 matrix can be invertible and given a point in image space, the inverse of the solution matrix obtained can be used to obtain the approximate 3D position. The DLT method itself obtains a solution by formulating a homogeneous linear system of equations and finding an approximate null space of the system matrix.

Let p_i, $i = 1,2,3$ be 3×1 vectors containing the rows of P, that is,

$$P = \begin{bmatrix} p_1^T \\ p_2^T \\ p_3^T \end{bmatrix} \qquad \text{(Eqn 4.7)}$$

then Eq. Eqn 4.4 can be written as,

$$U_i = \begin{bmatrix} p_1^T X_i \\ p_2^T X_i \\ p_3^T X_i \end{bmatrix} \qquad \text{(Eqn 4.8)}$$

Since, $U_i = PX_i$, we can infer that U_i and X_i are parallel. Therefore,

$$U_i \times PX_i = 0 \qquad \text{(Eqn 4.9)}$$

$$U_i \times PX_i = \begin{bmatrix} v_i p_3^T X_i - p_2^T X_i \\ p_1^T X_i - u_i p_3^T X_i \\ u_i p_2^T X_i - v_i p_1^T X_i \end{bmatrix}$$

$$= \begin{bmatrix} 0^T & -X_i^T & v_i X_i^T \\ X_i^T & 0^T & -u_i X_i^T \\ -v_i X_i^T & u_i X_i^T & 0^T \end{bmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} \qquad \text{(Eqn 4.10)}$$

$$= 0$$

Since X_i is a 3×1 vector each 0 on the left hand side represents a 1×3 vector of zeros. Thus the left hand side is a 3×9 matrix multiplied with a 9×1 vector. However,

$$u_i p_2^T X_i - v_i p_1^T X_i = -u_i (v_i p_3^T X_i - p_2^T X_i) - v_i (p_1^T X_i - u_i p_3^T X_i)$$

that is the third equation in the matrix is linearly dependent on the first two and as such can be discounted. This leaves,

$$\begin{bmatrix} 0^T & -X_i^T & v_i X_i^T \\ X_i^T & 0^T & -u_i X_i^T \end{bmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = 0 \qquad \text{(Eqn 4.11)}$$

or,

$$AP = 0 \qquad \text{(Eqn 4.12)}$$

which represents a system of homogeneous equations in matrix form. From a set of n, $n \geq 4$, point correspondences we have a $2n \times 9$ matrix A formed by stacking each of the equations from their respective point correspondences. The projection matrix for a given camera can then be computed by solving the set of equations $AP = 0$.

This has a trivial solution at $P = 0$ and is, in fact, homogeneous in P. If $P \neq 0$ is a solution then kP is also a solution, where k is an arbitrary scalar. For this reason, we fix the length of the solution vector with the requirement that $\|P\|^2 = 1$.

The homogeneous system has no solution when it is overdetermined and A is full rank, i.e. $rank(A) > 3$. This is typically the cases — the system of equations will not have any exact solution due to noise in the measurements [33]. In these cases we look for a solution to $AP \approx 0$ and use the least squares criterion,

$$\min \|AP\|^2 \qquad \text{(Eqn 4.13)}$$

with $\|P\|^2 = 1$, i.e. we look for a solution to the homogeneous least squares problem. This results in the eigenvalue problem

$$\min_{\|P\|^2=1} \|AP\|^2 \qquad \text{(Eqn 4.14)}$$

and can be determined by a singular value decomposition of A into USV^T where P is a right singular vector of A corresponding to the smallest singular value of A or the last column in V. Once P has been determined, the actual matrix can be found by a

simple rearrangement from a 9D vector to a 3×3 matrix.

Theorem 1. *Each $m \times n$ matrix A (with real coefficients) can be factorized into*

$$A = USV^T \qquad \text{(Eqn 4.15)}$$

where U and V are orthogonal ($m \times m$ and $n \times n$ respectively), and

$$S = \begin{bmatrix} diag(\sigma_1, \sigma_2, ..., \sigma_r) & 0 \\ 0 & 0 \end{bmatrix}$$

where, $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r > 0$ are the singular values of A or the square root of the eigenvalues of $A^T A$ and r is the rank of the matrix.

If A has the SVD given by Eqn 4.15 then

$$A^T A = (USV^T)^T USV^T = VS^T U^T USV^T = VS^T SV^T \qquad \text{(Eqn 4.16)}$$

Since $S^T S$ is a diagonal matrix this means that V diagonalizes $A^T A$ and therefore $S^T S$ contains the eigenvalues and V the eigenvectors of $A^T A$. The diagonal elements of $S^T S$ are ordered decreasingly $\sigma_1^2; \sigma_2^2; ...; \sigma_r^2, 0, ..., 0$. Thus, to find an eigenvector corresponding to the smallest eigenvalue we should select the last column of V .

Using Equation Eqn 4.15 in Eqn 4.14 we get the minimization problem

$$\min_{\|P\|^2=1} \|AP\|^2 = \min_{\|P\|^2=1} \left\|USV^T P\right\|^2 = \min_{\|P\|^2=1} \left\|SV^T P\right\|^2 \text{ since } U \text{ is orthonormal} \qquad \text{(Eqn 4.17)}$$

$$\|P\|^2 = \left\|V^T P\right\|^2 \qquad \text{(Eqn 4.18)}$$

By defining $\left\|V^T P\right\|^2$ as $\|Y\|^2$ we reduce the problem to

$$\|Y\|^2 = \|SY\|^2 \qquad \text{(Eqn 4.19)}$$

Since S is a diagonal matrix and $\sigma_i \geq \sigma_{i+1}$, the least square solution reduces to $Y = [0\ 0 \ldots 0\ 1]^T$. The solution of the original problem is

$$P = (V^T)^{-1} Y = VY = V \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \qquad \text{(Eqn 4.20)}$$

which is the last column of V or the right singular that corresponds to that smallest singular value of A.

In summary, to solve for the projection matrix P

- Set up the linear homogeneous system $AP = 0$ for a minimum of 4 point correspondences
- Compute the singular value decomposition $A = USV^T$
- Extract the solution v from the last column of V
- Rearrange the solution to form a 3×3 matrix
- Compute the inverse of the matrix obtained.

4.3 Discussion

We conducted various experiments to obtain a low error mapping from image space and world space points, after which we chose DLT as the preferred method. We observed that as the choice and the number of selected points changes the projection matrix obtained, and this, in turn, affects the positions of objects of in 3D world space.

4.3.1 Monte Carlo Optimization

Monte Carlo methods use repeated random sampling to generate numerical results. The underlying idea is to use randomness to solve problems that might be deterministic in principle. As a method of numerical optimization, Monte Carlo methods can be used to minimize or maximize functions of some vector with large dimension.

In the case of projection matrix derivation, we experimented with Monte Carlo simulations as random perturbations of an initial identity matrix until it projected known 3D coordinates to known 2D coordinates. If a random perturbation caused a 3D in world space to get closer to its target 2D projection in image space, then we keep that matrix as an improvement over our initial (or previous) guess, else the simulation continues. We run this over several hundred epochs or until the matrix no longer fluctuates with perturbation.

To evaluate how close the matrix is to the correct solution, the method requires some function of the matrix. The value returned by this function is what we use to check whether one matrix is better than another. The function we chose here was the average of the sum of squared distances between the target 2D points and 2D points obtained by multiplication with the obtained matrix. To perturb the matrix, we generated a random number from a uniform distribution between some range.

When run over significant number epochs, in most cases the error metric is reduced to the order of 10^{-2}. However, when tested over points not included in the training set, the matrix does not perform accurately. In some cases, the error metric remains stuck at some value and no perturbation is able to reduce it.

4.3.2 Choice of Points for DLT

Figure FC4.8: Sample choice of points.

As the number of point correspondences increases, the error between the transformed points obtained by multiplication with the matrix obtained from the DLT method and the original points increases. Based on various experiments, we have observed that 5 to 6 points results in a projection matrix with a small error in mapping and gives visually acceptable results in an orthographic view, and then results that resemble the original photo when rendered with a perspective projection. In general 2 points along the pavement edges at the bottom and 3 or 4 points distributed among objects of consideration in the scene produce the best results. Figure FC4.8 illustrates a sample choice of points based on our guidelines.

Normalization of both image space and world space points does not improve the accuracy of the obtained matrix. In most cases, the synthesized scene looks ‘off’ in comparison to its target image.

CHAPTER 5

3D SYNTHESIS PIPELINE

Chapters 3 and 4 describe the methods behind monocular depth estimation and perspective correction respectively — the two main modules to the image synthesis pipeline. Monocular depth estimation is performed using a fully convolutional neural network that treats the input image as the left image of a stereo pair and instead of trying to directly predict the depth, outputs the dense correspondence field d that, when applied to the left image, would enable us to reconstruct the right image, where d actually corresponds to the image disparity. Depth is then trivially calculated from the predicted disparity as bf/d where b and f are the baseline and focal respectively. Perspective correction is performed by estimating a mapping between ground plane points, $Y = 0$, in 3D world space and 2D projected image space points. By assuming the groundedness of points, the projection matrix is reduced to a 3×3 matrix that is estimated using DLT and SVD.

In this chapter, we detail the modules of 3D synthesis pipeline. As previously explored, 3D scenes are traditionally constructed manually by 3D artists who very often take inspiration from or model parts of the real world. In contrast, our proposed workflow leverages advances in deep learning networks to automatically model urban traffic scenes. By automating object placement, our proposed pipeline could save much time and effort that goes into the manual design and laying out of such scenes.

5.1 Pipeline Description

The proposed pipeline is intended for use on urban, outdoor scenes with straight roads and the primary objects of interest being cars, people, trees, buildings, and sidewalks. The image set around which this pipeline has been implemented is the Cityscapes dataset [37]. An overview of our pipeline is provided in Figure FC5.1.

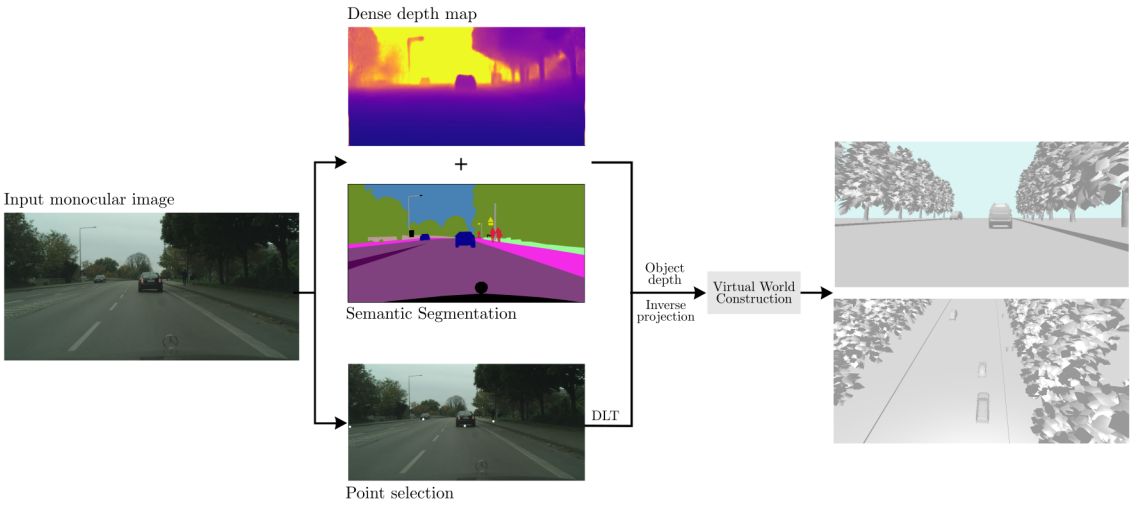


Figure FC5.1: 3D synthesis pipeline.

5.1.1 Dataset and Model

We use the Cityscapes dataset [9], a benchmark suite and large-scale dataset meant to train and test approaches for pixel-level and instance-level semantic labeling. Cityscapes is comprised of a large, diverse set of stereo video sequences from many European cities. Several hundreds of thousands of frames were acquired from a moving vehicle during the span of several months, covering spring, summer, and fall in 50 cities, primarily in Germany.

Images in the dataset are recorded with an automotive-grade 22 cm baseline stereo

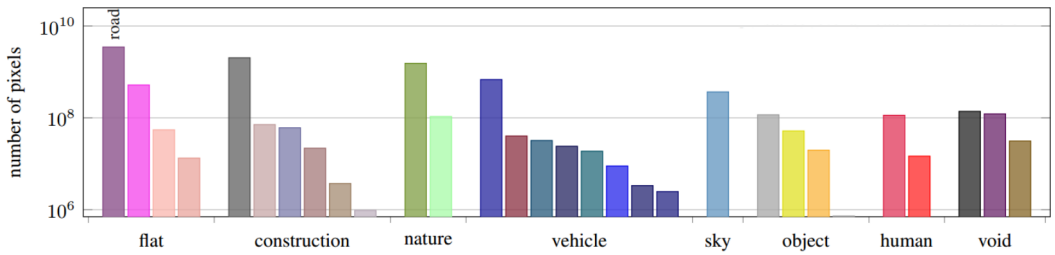


Figure FC5.2: Number of finely annotated pixels per class and their associated categories taken [9].

camera. The sensors were mounted behind the windshield and yield high dynamic-range (HDR) images with 16 bits linear color depth. Each 16-bit stereo image pair was subsequently demosaiced and rectified. 5000 images are finely annotated by semantic class —we use these images to test our pipeline. These fine pixel-level annotations consist of layered polygons and required more than 1.5 h on average for a single image. Annotators labeled the image from back to front such that no object boundary was marked more than once. Each annotation thus provides a depth ordering of the objects in the scene [9]. A table of semantic classes is provided in Table TC5.1 and the distribution of per pixel classes is shown in Figure FC5.2. Our classes of interest are sidewalk, building, person, vegetation and car.

Class	Category
unlabeled	void
ego vehicle	void
rectification border	void
out of roi	void
static	void
dynamic	void
ground	void
road	flat

sidewalk	flat
parking	flat
rail track	flat
building	construction
wall	construction
fence	construction
guard rail	construction
bridge	construction
tunnel	construction
pole	object
polegroup	object
traffic light	object
traffic sign	object
vegetation	nature
terrain	nature
person	human
rider	human
car	vehicle
truck	vehicle
bus	vehicle
caravan	vehicle
trailer	vehicle
train	vehicle
motorcycle	vehicle
bicycle	vehicle
license plate	vehicle

Table TC5.1: List of classes and their corresponding categories used to semantically segment images in Cityscapes.

We use a pre-trained model by Godard et al. [17], trained on Cityscapes (which has a train/val/test set with 22,973 training images). The total size of all the images 110 GB. The model was trained for 50 epochs, with a 512x256 resolution, a batch size of 8 and a VGG encoder-decoder.

5.1.2 Depth Estimation

A single input image is run through the depth estimation model described in Chapter 3 trained on Cityscapes. The network outputs a disparity map which is then converted to a depth map using the formula bf/d. The baseline and focal for Cityscapes is given to 0.22m and 2262 pixels respectively. The model takes on the order of 35 milliseconds to predict a dense depth map for a 512×256 image on a modern GPU [17].

The dense depth map obtained gives a depth value for every pixel in the image. We pass the ground truth semantic labels along with the raw, predicted depth values to the depth estimation module of our pipeline to estimate the average depth of each object of interest. The Cityscapes images are labeled from back to front such that no object boundary was marked more than once [9]. While the labeling of segments accounts for occlusion of one object by another the predicted depth map does not. Only the visible pixel is given a depth value. Therefore, while computing average depth, occluded objects can get inaccurate. In addition, depth estimation becomes noisier for objects farther away from the camera [17].

To correct for this, we make the assumption that the depth of all points along the same line $Y = c$ in image space and that are on the ground plane in world space have the same depth. In Figure FC5.3 each colored line represents a different value, and all points on the line (not including the breaks) have the same depth. We compute the

bounding box of each segment and consider the Y value (increases downwards) of the bottom corner to be the point at which the object touches the ground in the image. An object with a higher value of Y must be closer to the camera and have a smaller depth value.

The objects are sorted in increasing order of average depth and their ground points are compared in two ways:

- If two objects have comparable Y, say less than 10 pixels values, and their average depths differ by more than some threshold, then we set the average depth of both to be the average-depth of the closer object.
- If an object with a higher Y value has greater depth than the closet object with a lower Y value, we move the former behind the latter in 3D space.

Figure FC5.3: Depth values along the lines $Y = y$ in image space and on the ground plane can be assumed to have the same depth. Each coloured line represents a different value, and all points on the line have the same depth. Note that no line is drawn through objects since these are not on the ground. In terms of Y, $a > b > c > d$ and in terms of corresponding depth, $a < b < c < d$.

Once the average depths of every object are computed, we run a simple collision detector to ensure that no object intersects into another object along the Z-axis. The

final average-depth value obtained is used to translate a model of the particular object class along the Z-axis in the rendering engine.

5.1.3 Perspective Inversion

The user selects ground points on the image to estimate the inverse projection matrix. The world space depth is obtained by a simple look up from the predicted depth map at the selected pixel index, and the user gives a guessed estimate of what the world space X coordinate is. No Y coordinate is required since we assume all selected points are on the plane $Y = 0$. The guesstimate X-coordinates are given as pixel values, and not in meters (see Figure FC4.6). A capture of this interaction is seen in Figure FC5.4.

The projection matrix is computed by a singular value decomposition of the matrix A as described in Chapter 4 and its inverse is taken. The user gets an idea of the accuracy of the matrix obtained with his/her choice of points by the root mean square error (RMSE) between the results of multiplying the selected image points with the computed matrix and the guesstimated points. The RMSE reported is in pixels.

To obtain the position of each object in world space the X-coordinates of the left and right corners of the bounding box around each segment is multiplied by the inverse matrix obtained. The center of the object is translated along X by this value in the rendering engine.

5.1.4 Model Loading

We manually curated a small database of models based on the classes of interest from Cityscapes. We fixed the orientation of every model in 3D space as well as manually set the scale of every object with respect to another other. The X-coordinate, Z-coordinate, and class label are sent to the rendering engine. The database is looked

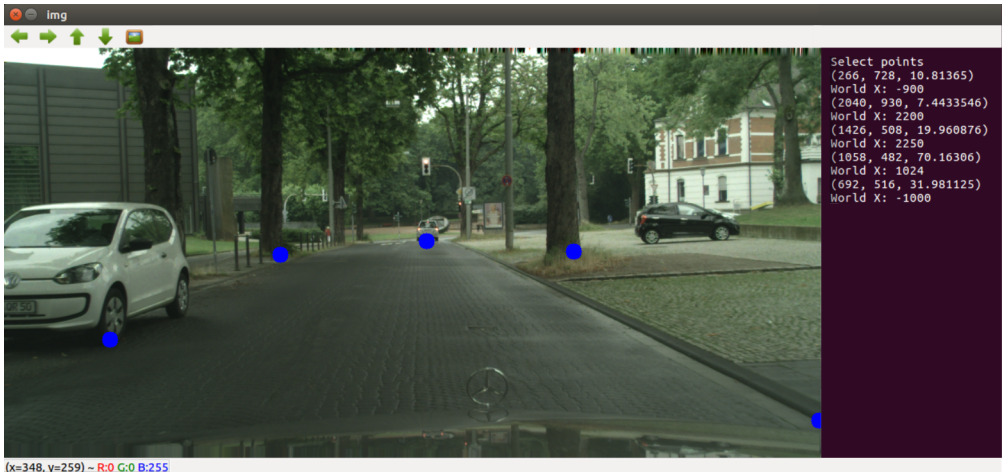


Figure FC5.4: Point selection by the user. A point on the image is double clicked and the 'guesstimated' X coordinate is entered by the user. The corresponding Z coordinate is picked up from the depth map. Once points are selected DLT is used to obtain the transformation matrix by computing the SVD on the linear homogeneous system of points. By increasing the number of known point correspondences, the accuracy in the mapping decreases.

up using the class label and a model corresponding to the label of a particular segment is loaded into the scene at the world space coordinates, with fixed orientation and scale. For example, if an object in the image has been semantically segmented as 'car' the rendering engine loads a generic hatchback or SUV (using a random number generator) to construct the 3D world. A woman in an image is semantically labeled as 'person' and the engine will load a generic model of a male human, Similarly, a generic tree is loaded for 'vegetation' even if the object in the image might be a shrub.

At present no interactive user interface has been designed to allow a user to change the type of 3D model that is loaded into a scene for a particular object class. This can be considered as future user interface additions to enhance the kind of 3D world our prototype application is able to construct. The $,Y,Z$ positions of each model in the final scene is written to a text file for future use or loading by some engine.

5.2 Results

We tested our pipeline on images of straight roads from Cityscapes with our primary objects of interest are vehicles, trees, sidewalks, people and buildings. The implementation is limited to straight roads without breaks and curves. We have rendered the scene without any textures, using a perspective projection in OpenGL, ambient lighting, a Phong shader, and a shadow map. The rendered result may look slightly different owing to the difference between the image camera's orientation and the OpenGL camera and projection. In the results seen in Figure FC5.5 one can see that object models have been loaded at the right positions and right depth with minor differences.

5.3 Discussion

The accuracy of the 3D scene produced or its resemblance to the input image in terms of object placement is highly dependent on the points that are chosen by the user and the error in the matrix mapping. However, if integrated into a modeling software like Blender, adjustments can be models using moving tools. This will not be cumbersome since our pipeline models a large portion of the scene and minor adjustments and fine-tuning of the scene can be done by hand.

One limitation of our pipeline is pose or orientation estimation of every object. We have not identified a method capable of detecting the orientation of an object with respect to the camera. In our prototype implementation, we make assumptions for the orientation of vehicles and the side of the road they are on, but nothing more. Again, our statement above could apply here as well —once the basic is scene is modeled orientation and other adjustments to position can be done by hand. Object pose estimation is an area of research in machine learning and we hope that perhaps a combination of template matching and deep learning can be used to automatically estimate the rotation

Figure FC5.5: 2D input image and corresponding 3D construction using our proposed pipeline.

of each object about the Y-axis.

Real, urban scenes are rich in variety with respect to the kind of cars, the kind of vegetation and building facades. We have provided a very limited database of generic models for each object class in our proof of concept application. As mentioned an implementation of our workflow can be extended to include a richer database that additionally allows the user to select the model being loaded into the scene for a particular class.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this thesis, we have proposed a workflow to automatically construct 3D scenes based on a reference monocular image with minimal user input. The workflow we have proposed can be operated to generate 3D visualizations of urban traffic scenes which can be used as a base for AR and VR environments, urban planning, video games, and simulation environments. The main inspiration behind our work is the time and manual effort that goes into scene modeling as well as the vast collection of images on the Web which can be used to construct 3D worlds and are used by artists anyway. An implementation of our workflow is meant to speed up the process of scene modeling and constructs a basic 3D scene that can later be detailed. Two key ideas that drive our workflow are the use of deep learning to estimate depth and perspective correction to get real-world positions from an image.

Prior approaches to modeling from images look more at the reconstruction of 3D scenes to aid navigation or provide point cloud visualizations rather than the construction of high quality, 3D scenes that can be used in the above mentioned areas. We make a distinction between the terms reconstruction and construction to indiciate the difference in the output and the goal of each. Our workflow has been designed keeping a single, monocular image as input at the heart of it. The depth estimation network proposed by Godard et al. [17] was specifically chosen since although trained on stereo

pairs, the trained model runs on a single, monocular image. Moreover, since the network does not rely on ground truth depth, it can if needed, be retrained with stereo data that is more easily attainable that RGBD data. Our workflow is able to produce good approximations of scenes as demonstrated.

6.1 Future Work

Curved Roads

Any curvature in roads has been discounted, that is, our workflow currently works well for straight roads. Computing lane curvature is slightly more complex, and an inverse projection as we have done so will not work. Algorithms to compute curvature such as [21, 38] can be integrated to generate scenes with curving roads.

Choice of Models

The prototype of this application does a simple lookup in a small library of 3D models that we have curated based on the class of an object in the scene. A generic 3D model is loaded giving a user no choice into what instance of the class is loaded. Our workflow can be upgraded to include a user interface that allows a user choice of what model is loaded into the scene similar to [28] where the semantic class of the object is known the user is posed with multiple model instances of that class who can then choose the closest match, see Figure FC6.1. This would allow the user to pick a 3D model that is the closest to an object in the scene, creating more visual similarity.

Detailing Scenes

The objects of interest in our present workflow are trees, vehicles, buildings, sidewalks, and people. This is partly due to the fact that depth estimation for small/slender objects in the scene is noisier than for larger objects. With objects like vehicles, noisy

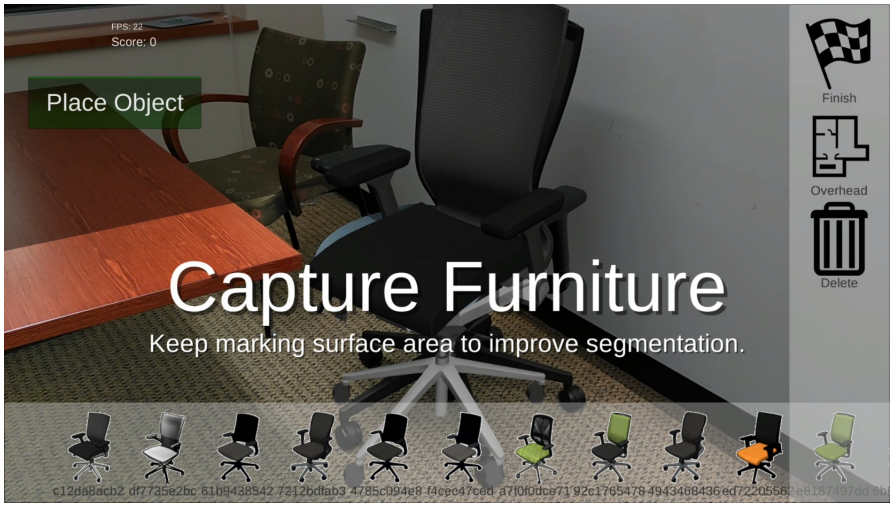


Figure FC6.1: Model selection by Sankar [28]

depth values for pixels within the object segment are averaged out unlike with more slender objects like poles. However, any urban scene contains barricades, road markings, shrubs, poles, street lights, traffic lights, signs and so on. An example of a more detailed scene is given in Figure FC6.2. While improvements in monocular depth estimation could help, to allow for further detailing this workflow could be made part of a consumer modeling software like Blender, where once the basic scene has been automatically modeled the user has the option of continuing the modeling process with the generated scene.

From the modeling process described in Chapter 1, a bare-bones scene is first modeled and then details are added in. It follows naturally that introducing such an automated scene modeling system into professional software like Blender, to automate the initial modeling process, would allow a user more time to pay attention to the detailing.

Aerial Images

The same idea can be extended to construction from aerial images. This would be easier given that the aerial view is likely to be undistorted. Depth and positional along X can be directly inferred. However, aerial images are considerably harder to obtain

Figure FC6.2: A more detailed scene of an urban road. Image credit: Shutterstock

than street level images.

Orientation

The orientation or pose of objects in the scene is not automatically estimated. Current object orientation is fixed based on the which half of the image they lie on. For example, traffic on the left lane can be usually considered to be facing the user and vice versa. However, this need not always be true. Cars can be parked perpendicular to sidewalks, people can be facing the side of a road, on road traffic can be facing either forwards or backward. A learning-based algorithm that computes the rotation of an object about the Y axis can be used to construct more accurate scenes. Given that semantic segments are passed to the implementation, these could be used to localize objects of interest and train a learning algorithm to correctly predict orientation.

City Modeling

Image-wise constructions can be fitted together to create large urban geographies as seen in Figure FC6.3. Multiple images could be used each starting off where the previous image ends and the modeled scenes could be stitched together to create one large scene. (This would require that curved or turning roads are captured.)

Deep Learning for Placement

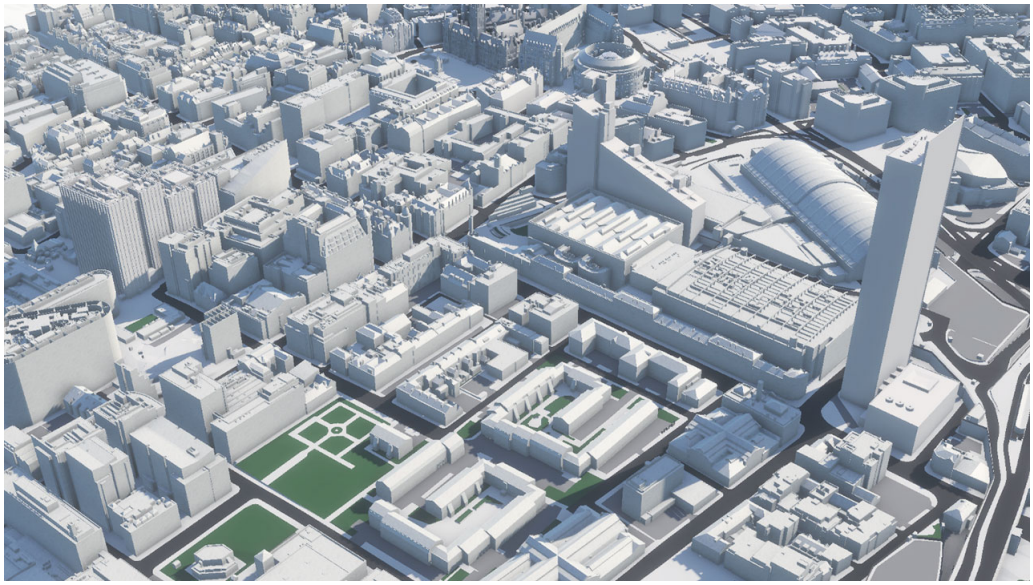


Figure FC6.3: 3D digital model of Manchester. Image credit: VU.CITY.

Given the success of deep learning in learning functions, it would be interesting to see if deep learning models could be used to predict the corresponding world space coordinates for given pixel space coordinates. Since there are dedicated methods used to determine depth, this would narrow down to learning a function that can predict world X-coordinates.

6.2 Summary

In this thesis, we introduced the problem of automatically constructing or synthesizing 3D worlds from monocular images, along with a workflow for the automated modeling of a subset of 3D, urban scenes from single images. Once the basic scene is set up using our woeklow it can be further detailed, animated and used as a production worthy 3D environment. Given the demand for 3D content and the tendency of artists to look at real-world images for inspiration and reference we strongly believe that an automated system capable of generating 3D scenes that are visually similar to an input image can speed up modeling time as well and make it easier to capture the real world

in 3D. Our prototype works for a subset of urban traffic scenes and we think it presents a useful direction for future research.

Bibliography

[1] Y. Abdel-Aziz and H. Karara. Direct linear transform from comparator coordinates into object space coordinates. In *Proceedings of the Symposium on Close-Range Photogrammetry*, volume 1, pages 1–18, 1971.

[2] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski. Building rome in a day. In *2009 IEEE 12th international conference on computer vision (ICCV)*, pages 72–79. IEEE, 2009.

[3] A. Akbarzadeh, J.-M. Frahm, P. Mordohai, B. Clipp, C. Engels, D. Gallup, P. Merrell, M. Phelps, S. Sinha, B. Talton, et al. Towards urban 3D reconstruction from video. In *Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT)*, pages 1–8. IEEE, 2006.

[4] P. Beardsley, P. Torr, and A. Zisserman. 3D model acquisition from extended image sequences. In *European conference on computer vision (ECCV)*, pages 683–695. Springer, 1996.

[5] F. Bernardini and H. Rushmeier. The 3D model acquisition pipeline. In *Computer graphics forum*, volume 21, pages 149–172. Wiley Online Library, 2002.

[6] M. Bertero, T. A. Poggio, and V. Torre. Ill-posed problems in early vision. *Proceedings of the IEEE*, 76(8):869–889, 1988.

[7] A. Bhoi. Monocular depth estimation: A survey. *arXiv preprint arXiv:1901.09402*, 2019.

[8] A. Cohen, C. Zach, S. N. Sinha, and M. Pollefeys. Discovering and exploiting 3D symmetries in structure from motion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1514–1521. IEEE, 2012.

[9] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3213–3223, 2016.

[10] B. Curless. From range scans to 3D models. *ACM SIGGRAPH*, 33(4):38–41, 1999.

[11] F. Dellaert, S. M. Seitz, C. E. Thorpe, and S. Thrun. Structure from motion without correspondence. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (Cat. No. PR00662)*, volume 2, pages 557–564. IEEE, 2000.

[12] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in Neural Information Processing Systems*, pages 2366–2374, 2014.

[13] J.-M. Frahm, P. Fite-Georgel, D. Gallup, T. Johnson, R. Raguram, C. Wu, Y.-H. Jen, E. Dunn, B. Clipp, S. Lazebnik, et al. Building rome on a cloudless day. In *European Conference on Computer Vision (ECCV)*, pages 368–381. Springer, 2010.

[14] R. Garg, V. K. BG, G. Carneiro, and I. Reid. Unsupervised cnn for single view depth estimation: Geometry to the rescue. In *European Conference on Computer Vision (ECCV)*, pages 740–756. Springer, 2016.

[15] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361. IEEE, 2012.

[16] C. Godard, O. Mac Aodha, and G. Brostow. Digging into self-supervised monocular depth estimation. *arXiv preprint arXiv:1806.01260*, 2019.

[17] C. Godard, O. Mac Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 270–279, 2017.

[18] C. Hane, C. Zach, A. Cohen, R. Angst, and M. Pollefeys. Joint 3D scene reconstruction and class segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 97–104, 2013.

[19] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015.

[20] N. Kalra and S. M. Paddock. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice*, 94:182–193, 2016.

[21] Z. Kim. Robust lane detection and tracking in challenging scenarios. 2008.

[22] A. Kundu, Y. Li, F. Dellaert, F. Li, and J. M. Rehg. Joint semantic segmentation and 3D reconstruction from monocular video. In *European Conference on Computer Vision (ECCV)*, pages 703–718. Springer, 2014.

[23] V. Lempitsky and Y. Boykov. Global optimization for shape fitting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2007.

[24] F. Liu, C. Shen, G. Lin, and I. Reid. Learning depth from single monocular images using deep convolutional neural fields. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):2024–2039, 2015.

[25] R. Mohr, L. Quan, and F. Veillon. Relative 3D reconstruction using multiple uncalibrated images. *The International Journal of Robotics Research*, 14(6):619–632, 1995.

[26] M. Pollefeys, D. Nistér, J.-M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S.-J. Kim, P. Merrell, et al. Detailed real-time urban 3D reconstruction from video. *International Journal of Computer Vision*, 78(2-3):143–167, 2008.

[27] S. Rusinkiewicz and M. Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 343–352. ACM Press/Addison-Wesley Publishing Co., 2000.

[28] A. Sankar. *Interactive In-Situ Scene Capture on Mobile Devices*. PhD thesis, 2018.

[29] S. Sengupta, E. Greveson, A. Shahrokni, and P. H. Torr. Urban 3D semantic modelling using stereo vision. In *2013 IEEE International Conference on Robotics and Automation*, pages 580–585. IEEE, 2013.

[30] N. Shiode. 3D urban models: Recent developments in the digital modelling of urban environments in three-dimensions. *GeoJournal*, 52(3):263–269, 2000.

[31] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3D. In *ACM transactions on graphics (TOG)*, volume 25, pages 835–846. ACM, 2006.

[32] I. E. Sutherland. Sketchpad a man-machine graphical communication system. *Simulation*, 2(5):R–3, 1964.

[33] L. University. Lecture 3: Camera calibration, DLT, SVD, 2013.

[34] G. Varma, A. Subramanian, A. Namboodiri, M. Chandraker, and C. Jawahar. Idd: A dataset for exploring problems of autonomous navigation in unconstrained environments. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1743–1751. IEEE, 2019.

[35] C. Wu. Towards linear-time incremental structure from motion. In *2013 International Conference on 3D Vision (3DV)*, pages 127–134. IEEE, 2013.

[36] E. Yares. 50 years of CAD, 2013.

[37] H. Zhao, O. Gallo, I. Frosio, and J. Kautz. Loss functions for image restoration with neural networks. *IEEE Transactions on Computational Imaging*, 3(1):47–57, 2017.

[38] S. Zhou, Y. Jiang, J. Xi, J. Gong, G. Xiong, and H. Chen. A novel lane detection based on geometrical model and gabor filter. In *IEEE Intelligent Vehicles Symposium*, pages 59–64. IEEE, 2010.