Felipe Marinho Tavares
João Guilherme de Souza Alves Costa

Representation of the Computer Vision of Autonomous Vehicles using Computer Graphics

Itabira

2018

Felipe Marinho Tavares
João Guilherme de Souza Alves Costa

Representation of the Computer Vision of Autonomous Vehicles using Computer Graphics

Undergraduate Thesis presented to the Universidade Federal de Itajubá - Itabira Campus as requirement to the conclusion of the Computer Engineering course under the guidance of PhD Giovani Bernardes Vitor.

Universidade Federal de Itajubá – UNIFEI

Computer Engineering

Itabira Campus

Supervisor: Giovani Bernardes Vitor

Itabira
2018

Felipe Marinho Tavares
João Guilherme de Souza Alves Costa

Representation of the Computer Vision of Autonomous Vehicles using Computer Graphics

Undergraduate Thesis presented to the Universidade Federal de Itajubá - Itabira Campus as requirement to the conclusion of the Computer Engineering course under the guidance of PhD Giovani Bernardes Vitor.

Itabira, November 24, 2018:

Giovani Bernardes Vitor
Supervisor

Juliano de Almeida Monte-Mor
Committee Examiner

Érick Oliveira Rodrigues
Committee Examiner

Itabira
2018

Abstract

In order to bring a sense of security and reliability to an autonomous vehicle passenger, this work elaborates and develops a system with the use of computer graphics to provide an estimated representation in a 3D environment of the interpretation of computer vision algorithms. The proposed system has as input stereo camera images and a semantic image (that maps and differs types of objects in a scene), and have as output the display of a scene in a 3D environment. Two separate system modules have been developed, one with focus on computer vision for feature extraction and other focused on computer graphics for the representation of entities in a scene. The first system named Cloud Analysis obtained satisfactory qualitative results on feature extraction on generated disparity images and point clouds, the second system named 3D environment was developed with a compact stack approach in mind for the representation of real-world scenes, qualitative results are shown with tailored baseline projections and inputs from the first system feature extractor.

Keywords: Computer Graphics, Shaders, Entity-Component System, Point Clouds, Computer Vision, Image Depth, 3D simulation.

List of Figures

List of Tables

Contents

1 Introduction

The concept of Autonomous Vehicles introduces hopes for improvements to general locomotion, offering more security, comfort, and potential for new types of services. However, there are still obstacles which have to be worked on before its popularization: as the technology cultural/social acceptance and the required built-in cost.

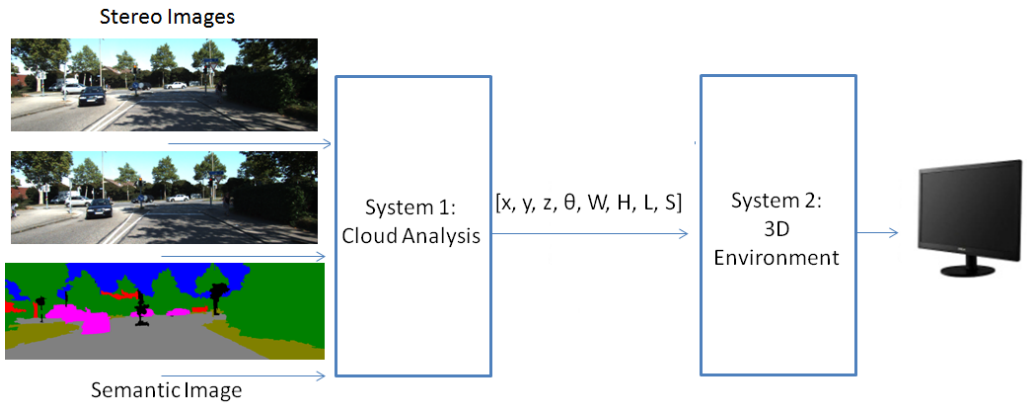
A possible solution to facilitate the dilution of the idea of machines driving general purpose vehicles would be to provide to users a visualization of what is being interpreted by the car algorithm. An approach that could also be implemented for case scenarios on supervision or surveillance.

The present work proposes the elaboration and development of a system for the representation in a 3D environment of the interpretation of autonomous cars algorithms as to provide a visualization interface for passengers.

For the development and validation of the proposed system are used datasets with real-world images acquired by stereo cameras (GEIGER et al., 2013) and semantic images that distinguish different categories of objects in a scene such as cars, sidewalks, and vegetation (VITOR, 2014).

Work was divided into the creation of two separate systems, the first one responsible for treating the input images and delivering as output a set of information for each identified entity in the scene, the second responsible for receiving entities and distributing them in an isolated graphical environment with predefined models in the correct position and size. The complete system data flow is shown in Figure 1.

Figure 1 – System flowchart



Source: Authors of this study.

The first system named Cloud Analysis is defined by four significant steps: disparity image generation, point cloud creation, semantic association, and feature extraction. It uses concepts of object-oriented programming (RUMBAUGH et al., 1991) in its implementation to organize data behavior.

The second system is named 3D environment and has as input information of entities, it represents them in a 3D simulation using computer graphics. The concepts of Entity-Component Systems (HOUSE, 2012) and Data-Oriented Design (LLOPIS, 2011) are used in its implementation as an approach for better performance results, thus requiring a more in-depth modeling of components and execution flow, which are explored in the development section.

As contribution, this project delivers an approximation of the environment identified by computer vision algorithms. The system purpose is to bring more confidence to users by offering an interface dedicated to the understanding of algorithms running underneath autonomous vehicles. As future contribution the system can be used for data visualization in case scenarios of surveillance, or accidents by the processing of stored data.

1.1 General Objective

Elaboration and development of a software for embedded systems to create an estimated representation of the interpretations of Computer Vision algorithms for Autonomous Vehicles.

1.2 Specific Objectives

- Creating 3D scenes with discriminating categories of objects from generated points clouds using disparity and semantic images;
- Describing entities in scenes using feature extraction;
- Survey of adequate software architectures and implementations for 3D Environment development;
- Development of the 3D Environment visualization software.

2 Literature Review

2.1 State of the Art

In this section similar works are cited with the goal of contextualizing our proposal, expressing relations and differences to approaches currently found in the industry and academic research.

2.1.1 Zoox: Simulation for Autonomous Driving

Zoox is a startup founded in 2014 with the goal of bringing autonomous mobility experiences to the market. The company is in development stage (ZOOX, 2014a) and doesn't have widespread marketing campaigns (ZOOX, 2014b). However, it stands out in the industry due to demonstrations of their autonomous-driving technology.

The startup made its "tech media" debut on Unreal Engine SIGGRAPH 2017 User Group (KENTLEY-KLAY, 2017) with a technology sneak peek of its simulation pipeline. There they showed a car with 8 sensors as demonstrated in Figure 2 and focused on presenting an elaborate virtual environment with the goal to enhance the passenger experience using visualizations of interpretations of autonomous-driving algorithms as shown in Figure 3.

Figure 2 – Zoox demonstration car

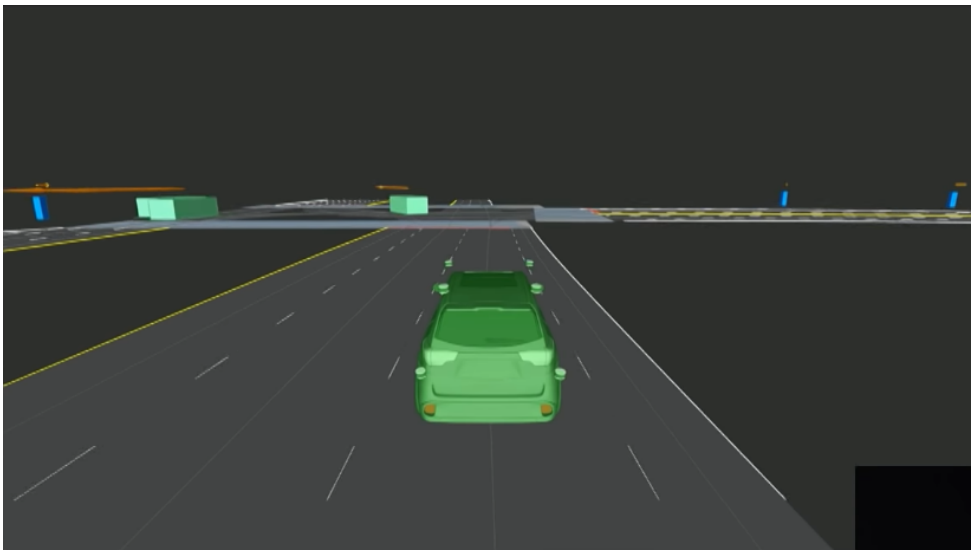
Source: Adapted from (KENTLEY-KLAY, 2017).

Figure 3 – Zoox simulation environment

Source: Adapted from (KENTLEY-KLAY, 2017).

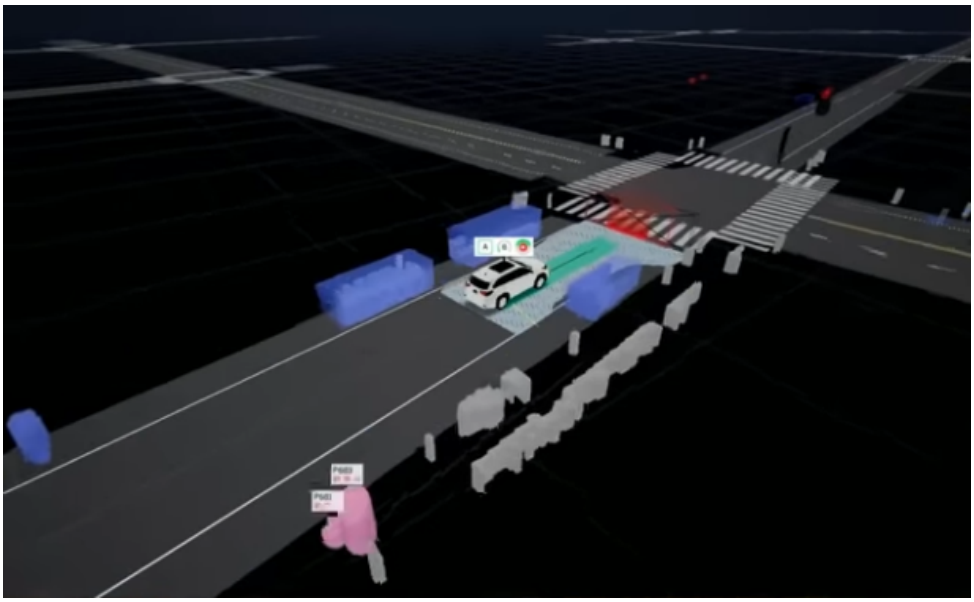
On 2018 Zoox introduced their Youtube channel (ZOOX, 2018b) with an evolved demonstration of the route network simulation from SIGGRAPH 2017, shown in Figure 4, with more entities and an instant route estimation of the autonomous-driving algorithm (ZOOX, 2018a) as shown in Figure 5.

Figure 4 – Zoox route network



Source: Adapted from (KENTLEY-KLAY, 2017).

Figure 5 – Zoox simulation environment with route estimation



Source: Adapted from (ZOOX, 2018a).

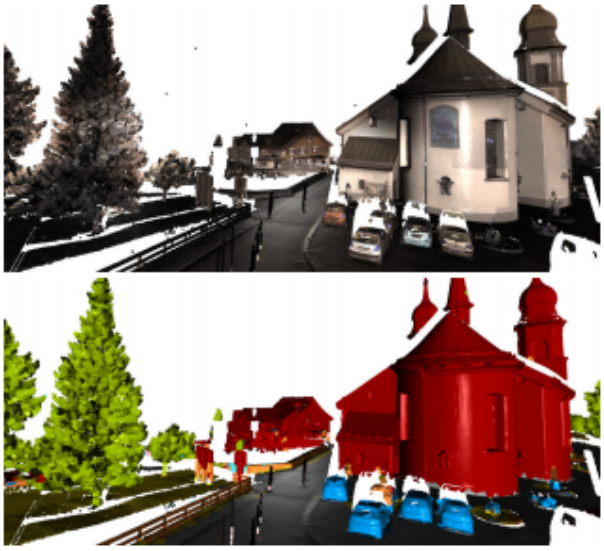
The current work relates to Zoox as both has as goal to provide visualization of autonomous vehicles algorithms. However, the cited startup considers more sensors (Velodyne, LiDAR) than our proposal (two stereo cameras) while also not necessarily focusing on performance for constrained systems.

2.1.2 Image Segmentation and Depth

One of the major problems in the 3D environment reconstruction and computational vision is to develop a system capable of accurately discriminating each of the elements present in the environment.

For the semantic association problem, (TATARCHENKO et al., 2018) proposes the use of deep convolutional networks in contrast to volumetric approaches operating directly on surface geometry. Crucially, the construction is applicable to unstructured point clouds and other noisy real-world data. The results can be seen in Figure 6.

Figure 6 – Top: point cloud from the Semantic3D dataset. Bottom: semantic segmentation produced by the approach.

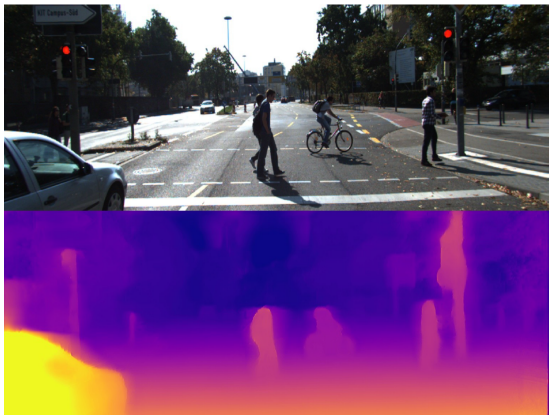


Source: Adapted from (TATARCHENKO et al., 2018).

The current work relates to (TATARCHENKO et al., 2018) by having point clouds as input. However, while (TATARCHENKO et al., 2018) associate semantic meaning to data, our work have as input semantic images obtained by the current autonomous-driving algorithm in the environment.

Aside from a semantic approach, to accurately discriminate elements in a scene for a virtual representation, it's important to estimate the distance and position of elements. (PILLAI; AMBRUS; GAIDON, 2018) is an example of work that tries to improve the state of art on depth estimation (results shown in Figure 7).

Figure 7 – Trained Monocular disparity estimation network in a self-supervised manner using a synchronized stream of stereo imagery, relieving the need for ground truth depth labels



Source: (PILLAI; AMBRUS; GAIDON, 2018)

The current work doesn't focus on the creation of a new depth estimation method, as this topic deserves a research of its own. However, on the proposed system depth calculation is part of the pipeline, and a method for it will be explored on the development section.

2.1.3 CARLA: An Open Urban Driving Simulation

According to CARLA's Website (CARLA, 2017), it's a simulator developed from the ground up to support development, training, and validation of autonomous urban driving systems. A open-source project that provides open digital assets (urban layouts, buildings, vehicles) that were created for this purpose and can be used freely. It is a complex simulator platform that supports flexible specification of sensor suites and environmental conditions (such as in Figure 8). In this platform it's possible to test a several kind of system that needs a urban environment. In addition, the system can generate performance reports.

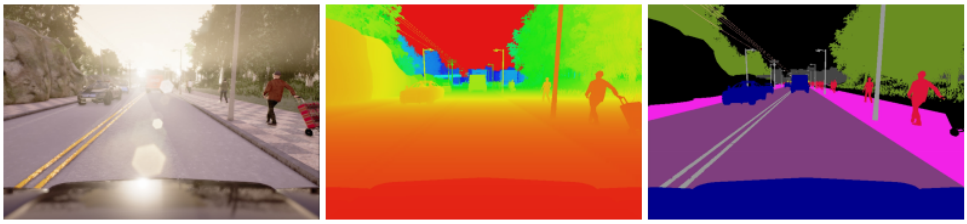
Figure 8 – A street in Town 2, shown from a third-person view in four weather conditions. Clockwise from top left: clear day, daytime rain, daytime shortly after rain, and clear sunset.

Source: Adapted from (DOSOVITSKIY et al., 2017).

The simulator engine is defined primarily in the paper (DOSOVITSKIY et al., 2017) as the development of the environment and sensors. The environment, as shown in Figure 8, is composed of 3D models of static objects such as buildings, vegetation, traffic signs, and infrastructure, as well as dynamic objects such as vehicles and pedestrians. And then the authors used the following steps to build urban environments: (a) laying out roads and sidewalks; (b) manually placing houses, vegetation, terrain, and traffic infrastructure; and (c) specifying locations where dynamic objects can appear (spawn).

About the sensors, CARLA allows for flexible configuration of the agent's sensor suite. At the time of writing, sensors are limited to RGB cameras and to pseudo-sensors that provide ground-truth depth and semantic segmentation, such as Figure 9.

Figure 9 – Three of the sensing modalities provided by CARLA. From left to right: normal vision camera, ground-truth depth, and ground-truth semantic segmentation. Depth and semantic segmentation are pseudo-sensors that support experiments that control for the role of perception. Additional sensor models can be plugged in via the API.



Source: Adapted from (DOSOVITSKIY et al., 2017).

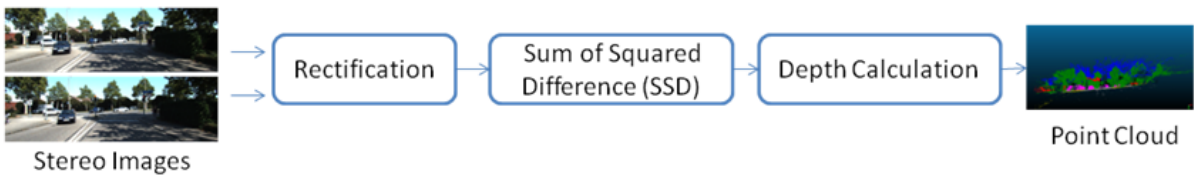
The sensors are be able to segmented 12 semantic classes: road, lane-marking, traffic sign, sidewalk, fence, pole, wall, building, vegetation, vehicle, pedestrian, and other. Therefore, CARLA manages not only to create a realist urban environment, but also to provide tools that allow the complete analysis of the environment around the car.

The current work relates to CARLA as both proposes software with the use of computer graphics to assist the field of autonomous driving. However, CARLA's provides a virtual environment to test autonomous driving algorithms and compare them, while this research is focused in representing an approximation of the environment surrounding a real autonomous car and considering constraints of the real-world (as intrinsic sensor noise and computational limitation).

2.2 Point Cloud Generation

This section contains concepts utilized to generate point cloud of a scene, and from it extract entity information by feature extraction methods. Point clouds generation steps are separated as camera calibration; disparity and semantic images generation and point cloud creation. These steps can be conceptual defined as the Figure 10 shows.

Figure 10 – Conceptual Flow to Generate a Point Cloud



Source: Authors of this study.

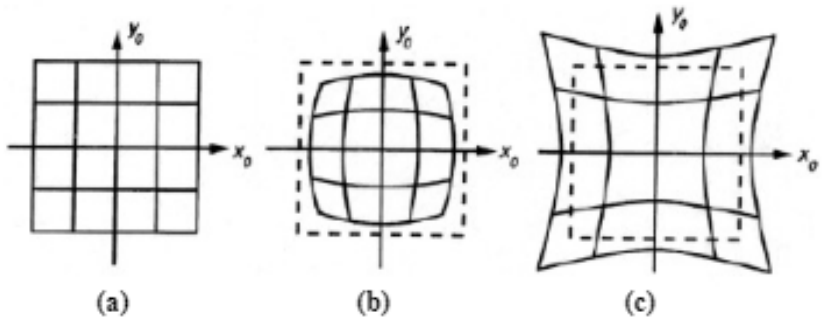
As in Figure 10 flow, one must first calibrate the system according to the camera

configuration used by conducting image rectification. Next, the Sum of Square Difference (SSD) calculation is used to obtain the disparity map and then calculate the depth of each scene element. The following subsections explain each of these steps in detail.

2.2.1 Camera Calibration

Cameras have a natural distortion due to physics constraints and different types of lens. The more common are the Barrel and Pincushion distortions, and, according to (VASS, 2003, p.1) “The first is due to the fact that many wide angle lenses have higher magnification in the image center than at the periphery. This causes the image edges to shrink around the center and form a shape of a barrel”. On the other hand, the Pincushion has the opposite effect, causing many wide angle lenses to have high magnification in the periphery of the image. Both are shown in Figure 11. It’s important to calibrate the cameras in order to eliminate these distortions as much as possible.

Figure 11 – Types of radial distortion (a) Ideal image with no distortion, (b) Barrel Distortion, (c) Pincushion Distortion.



Source: (PARK; BYUN; LEE, 2009).

In addition to distortion, calibration is also important for obtaining intrinsic parameters. According to (MEIRELES, 2002), "the intrinsic parameters serve to relate the pixel coordinates relative to the images with the coordinates of points of space measured in the referential system originating in the center of the chamber. These parameters depend exclusively on the physical characteristics of the camera (its internal geometry, lens type) ". (COELHO; TAVARES, 2003), in a laboratory report, shows the main intrinsic parameters to be observed: Focal Distance, Optical Center of the image, Coefficient of lag, and, of course, distortions.

The focal length, in pixels, represents the distance between the projection center and the image plane; The optical center represents the coordinates of the optical center of the image in frame memory; The lag coefficient represents the angle between the x-axis, y-axis, z-axis of the image in frame memory and the distortions representing the radial and tangential distortion coefficients.

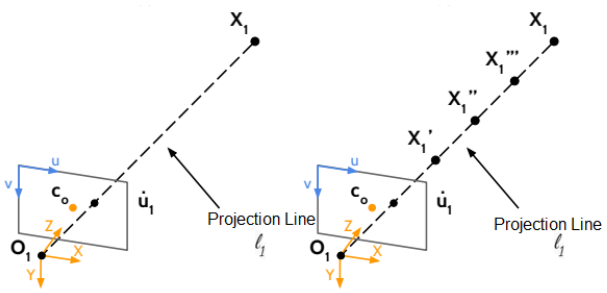
This research uses the distortions parameters, camera matrix and dataset with street pictures publicly available (GEIGER et al., 2013).

2.2.2 Epipolar Geometry

Epipolar Geometry refers to a mathematical operation that indicates how far apart the components of a scene are. With the help of two cameras (stereo cameras) it is possible to cross similar points of the same image and trace their spatial differences.

To understand the concept of epipolar geometry, it is necessary to understand the line of projection. Considering an image of any scene, as shown in Figure 12, it can be said that the projection line is the black line that cuts perpendicularly the image. In addition, for understanding, it is necessary to consider O_1 the plane that the camera is and U_1 the plane that shows the scene of the image.

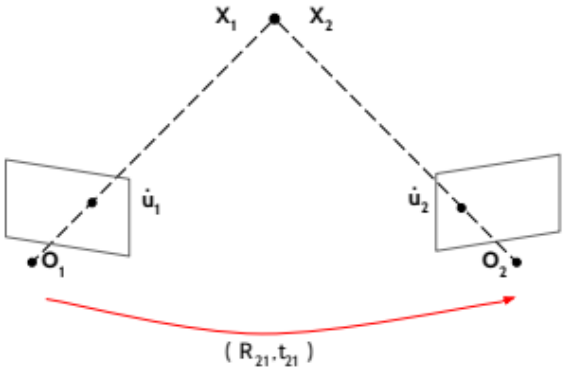
Figure 12 – Projection Line



Source: Adapted from (SILVA, 2016).

It can be noted by looking at the image that any object located above the projection line (represented by $X^{'}$, $X^{''}$ and $X^{'''}$) will be superimposed (or hidden). The disparity matrix serves precisely to understand, from images, the depth of this projection line. And for this, a second camera is used, as shown in Figure 13.

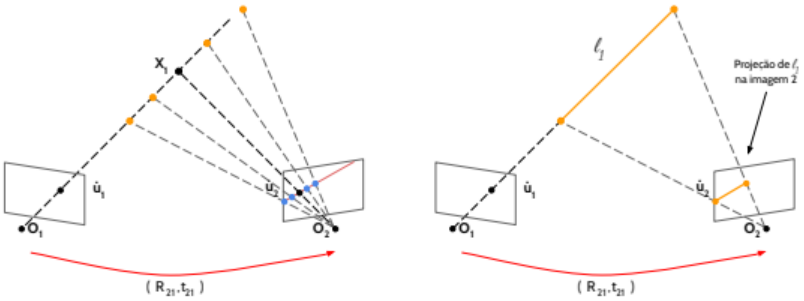
Figure 13 – Triangulation of a point



Source: (SILVA, 2016).

It is possible to see the projection line using a second camera. Therefore it's also possible to match common points between these images, such as in Figure 14.

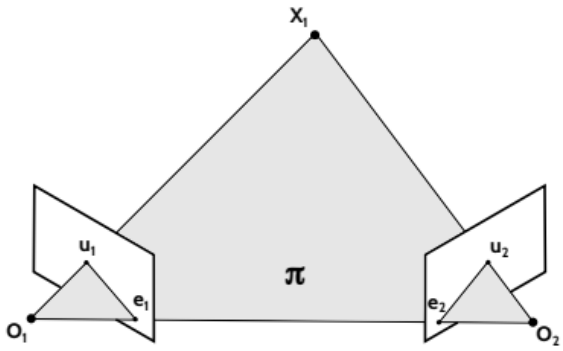
Figure 14 – Matching Points



Source: (SILVA, 2016).

By treating the two images, eliminating their distortions and rectifying one against the other, it is possible to obtain something called the "Baseline" (SILVA, 2016), shown in Figure 15.

Figure 15 – Baseline



Source: (SILVA, 2016).

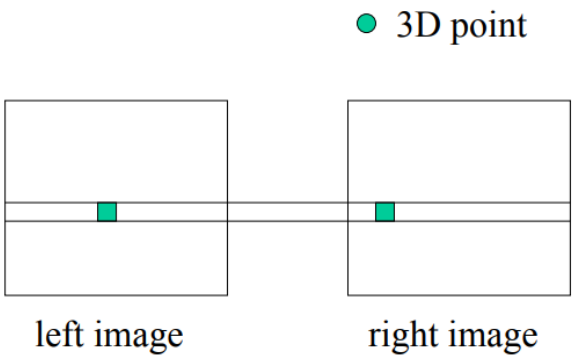
In his thesis (SILVA, 2016) explain that with this configuration it's possible to define a π plane.

The point e_1, called the epipolo, is a projection of the optical center of the camera 2 (O_2) in the plane of the image of the camera 1. The epipolo e_2 is found in the same way. The line is formed by points O_1 and O_2, where it connects the optical centers of the cameras is called the baseline. Its intersection with the planes of the images form epipolar lines.

2.2.3 Disparity and Sum of Squared Difference

Disparity matrix is a set containing the x-axis distance differences from two pixels obtained from left and right images. This matrix can be acquired from the pixels similarity (such as Figure 16) and Sum of Squared Difference (SSD) calculation, in code it can be obtained by a class called StereoSGBM using the OpenCV framework (OPENCV, 2018). This class match the analyzed function from two inputs: the left and right rectified pictures of the same scene, with requirement of having the images in grayscale.

Figure 16 – Camera Stereo configuration and Similarity of pixel



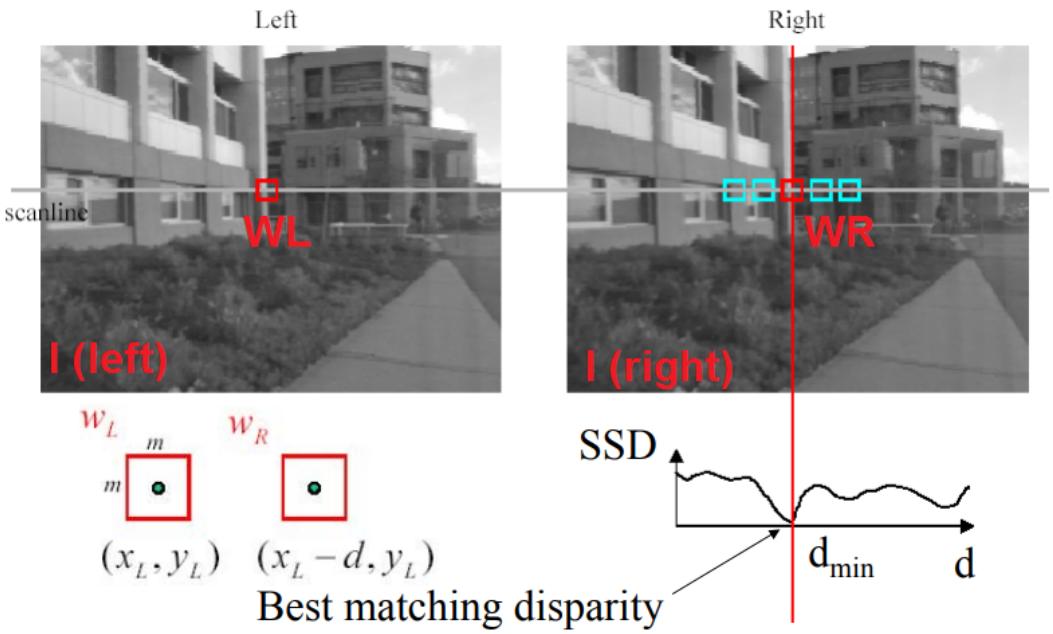
Source: (RAO, 2009).

According to the Figure 16, the point matching is performed when a common point is found in the two images and the SSD calculation is responsible to find this common pixel. This calculation can be defined by the following equation:

$$C_r(x,y,d) = \sum_{(u,v)\varepsilon W_m(x,y)} [I_l(u,v) - I_r(u-d,v)]^2 \tag{2.1}$$

From the moment images are rectified, it can be stated that: given a horizontal pixel row in an image, there will be a corresponding pixel row in the other image. The SSD calculation is responsible for analyzing pixels in a row and finding the most similar pixels between them. An approach to find the similarity between one pixel and another is to power of two the subtraction of these two pixels, as Equation 2.3 shows. Where I_l is the left image; I_r is the right image; u and v pixel position coordinates; and d the variation in x of the pixel. Figure 17 exemplifies the calculation in a more visual manner.

Figure 17 – SSD Calculation



Source: (RAO, 2009).

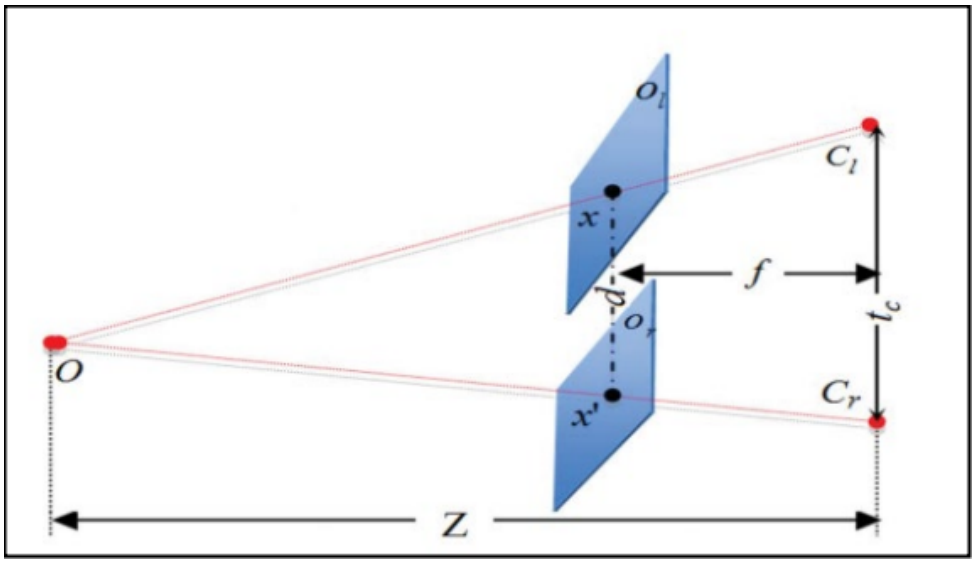
The analysis is faithfully represented by Figure 17, it is possible to observe how each step is performed: First, it defines which pixel should be analyzed in the first image; Next, the pixel row of the second image is defined as the first; Then, the SSD is calculated in the first pixel and its horizontal component varies in order to perform the same subtraction throughout the line; Lastly, it will have a vector of values that will represent which close each pixel of that line is in relation to the the first image pixel. The one with the lowest value will be the pixel most likely to match.

2.2.4 Point Cloud

Point Cloud is a 3D reconstruction of a stereo photo, generated from a disparity map. The function used to do this on the OpenCV framework is called reprojectImageTo3D() and has an important output: A matrix containing x, y and z values with an intensity value in gray scale that shows the distance of each pixel (point) - The closer the camera is to the point, darker is the pixel.

The reprojectImage3D function is an OpenCV feature used to become transparent the calculations and concepts used in the 3D scenario reconstruction. A calculation that uses as epipolar geometry principle and works two important concepts: disparity and depth. Using this geometry, the 3D problem become a 2D problem and thus, it's possible discuss about the disparity and depth concepts. Figure 18 evidence how these concepts are joined together for the reconstruction of a 3D scenario.

Figure 18 – Disparity and Depth



Source: (BATISTA; REGIS, 2013).

In the image processing conception, it has as disparity the proximity of the same object described in the two images and it has as depth the actual distance of the object in relation to the two cameras.

Similar to Figure 15, it can be stated that once a baseline is defined, the 3D problem around the depth of the elements in the scene becomes two-dimensional similar triangles. Therefore, it can be said that the height of this triangle can be represented by Equation 2.2, since the base of the smaller triangle can be described by the disparity equation, as shown in Equation 2.3.

$$Z = \frac{t_c \cdot f}{d} \tag{2.2}$$

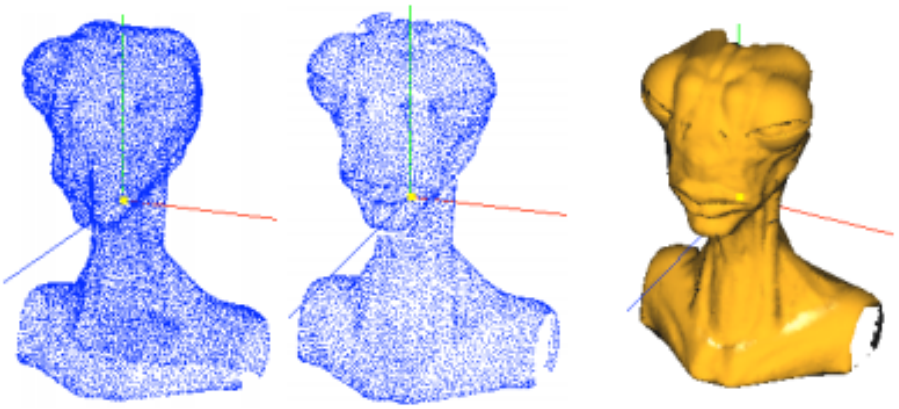
$$d = |x_l - x_r| \tag{2.3}$$

Thus, it's possible to discover how deep is each pixel and print them in a 3D scenario. This scenario is called a point cloud.

There are many applications with a point cloud in a 3D environment. It is usually used in conjunction with an algorithm to treat the surface of these points and reconstruct an approximate image from those points with graphic details. Thus, it's possible to reconstruct the real environment by a 3D environment with details. In the other hand, the point cloud can be used to acquire some useful data without a biggest graphic details, such as: size of different objects in scene, the center of mass, object directions and others.

The Figure 19 shows the point cloud and the possible applications with it.

Figure 19 – Point cloud examples



Source: (COSTA; PESCO, 2009).

In the first image, it can be noticed the pure point cloud – a 3D representation of a Alien with points. Alexandre, in his article (COSTA; PESCO, 2009), used a reflection conception to describe the image graphically. In the other hand, it's possible to discover information such as center of mass and substitute the point cloud to an alien's art.

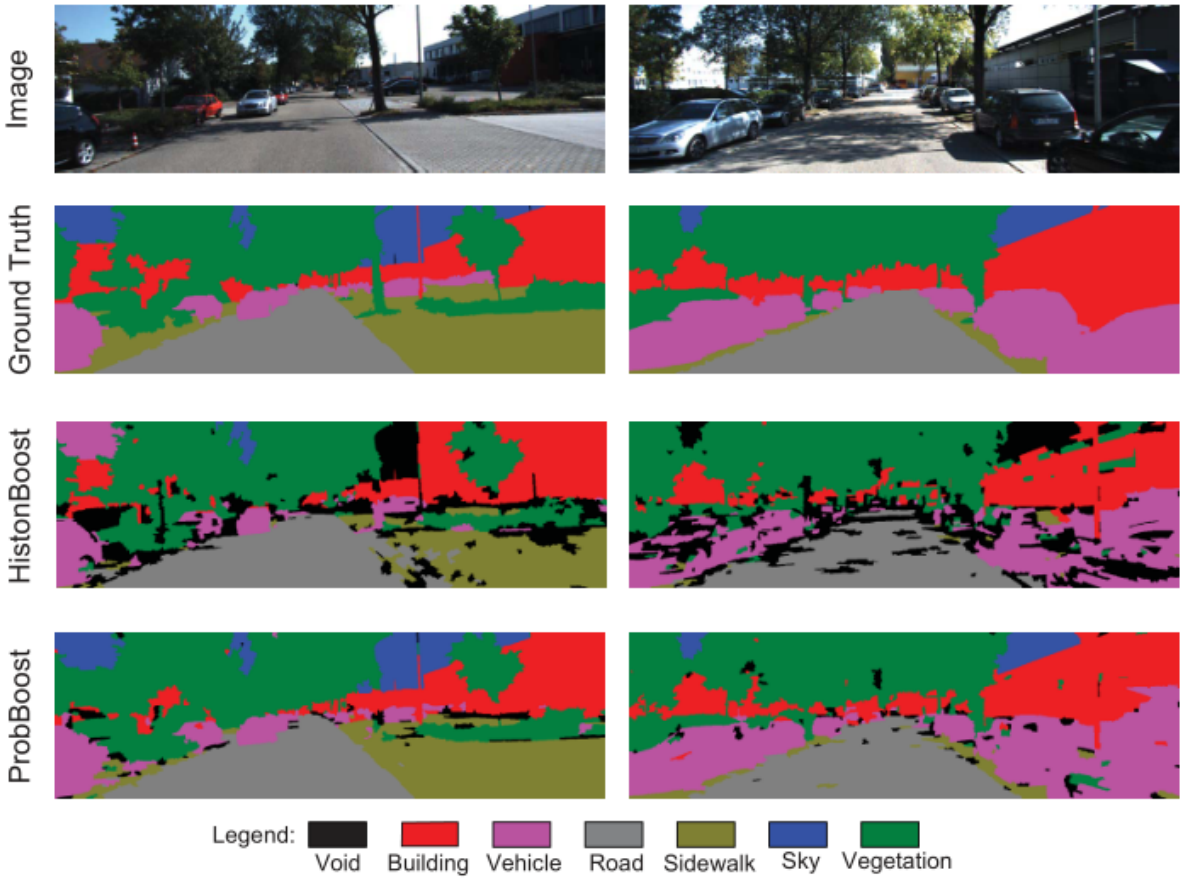
2.2.5 Semantic Context

A real image is made up of several objects and elements different from each other. In an urban environment, an image can contain pedestrians, cars, streets, animals, houses, poles and other elements that a human can easily differentiate. However, for an algorithm it is a complex task.

In his thesis, Giovani (VITOR, 2014) describes a code capable of performing this objects differentiation in an urban image and compares the results obtained with other algorithms used in the literature. With its algorithm, it is able to discriminate road, sidewalk, vehicles, buildings, sky, vegetation and void spaces - spaces without discrimination.

Giovani (VITOR, 2014) evidences the result of three approaches developed by him: ProbBoost, HistonBoost and ANN (Such as Figure 20). To use as a parameter and to train the neural network that will contribute to these approaches, the Ground Truth concept is used - an image that graphically represents the differentiation of the objects in the scene, as shown in Figure 21.

Figure 20 – Some results of Giovani's Thesis



Source: (VITOR, 2014).

Figure 21 – Semantic Image

Source: (VITOR, 2014).

In this project, the images generated in the Giovani's research (VITOR, 2014) named as Ground Truth are used as a semantic image. The urban environment images are analyzed with the help of the dataset provided by KITTI. KITTI's dataset is an urban image dataset created by Karlsruhe Institute of Technology (KIT).

2.2.6 Difficulties

(RAO, 2009) in his lecture explain that this stereo reconstruction pipeline has some problems. Such as:

- Camera calibration errors;
- Poor image resolution;
- Occlusions;
- Violations of brightness constancy (specular reflections)
- Large motions;
- Low-contrast image regions;

All of these points are used for the analysis of results.

2.3 Feature Extraction

This section approaches feature extraction methods for appliance on set of points with the final goal of describing entities on a scene. Since this project consider scene semantic information as input. Thus, there is exploration on methods for recognition of different categories.

2.3.1 Center of Mass

According to (HALLIDAY; RESNICK; WALKER, 2009, p. 219), calculating the center of mass of 3D particles is possible by the following equation:

$$x_{mc} = \frac{1}{M}\sum_{i=1}^{n} m_i x_i \tag{2.4}$$

$$y_{mc} = \frac{1}{M}\sum_{i=1}^{n} m_i y_i \tag{2.5}$$

$$z_{mc} = \frac{1}{M}\sum_{i=1}^{n} m_i z_i \tag{2.6}$$

Where m is the weight of the particle and M is the total weight of all particles together. In the case of a picture, all points have the same weight (value 1) and M is equal to the total number of particles. Thus, this equation can be simplified to:

$$x_{mc} = \frac{1}{M}\sum_{i=1}^{n} x_i \tag{2.7}$$

$$y_{mc} = \frac{1}{M}\sum_{i=1}^{n} y_i \tag{2.8}$$

$$z_{mc} = \frac{1}{M}\sum_{i=1}^{n} z_i \tag{2.9}$$

2.3.2 Box of lower and upper limits of the object

There are different ways of calculating limits for a object defined by a set of points. Two approached in this section are the use of center of mass & geometric centroid, and the analysis tool of variance.

The first method calculates the center of mass of an object, refer to it as origin, and for each direction calculates the geometric centroid of points, defining them as direction's limit. Repeating this procedure for all directions creates a box which represents the object's approximate size. This approach is not efficient for the reason that it will always set limits lower than the actual limit of the object. Because it is an average of points, this value will hardly match the maximum point of the object.

On the other hand, (MONTGOMERY; RUNGER, 2011) explain a more suitable way of finding these limits. The variance of a random variable is a measure of dispersion, or scatter, in the possible values for the variable. Thus, it's possible to find an approximate value of dispersion for the center of mass (υ) using the variance formula in Equation 2.10.

$$\sigma^2 = \sum_{x}(x-\upsilon)^2 f(x) \tag{2.10}$$

It's possible to find an approximate maximum point of different directions, and then create an upper and lower box of the object, when using Equation 2.10. Finally, the difference between the upper and lower limits is calculated in order to define the width, height and length values.

2.4 3D software architecture

This section approaches 3D software architectures considering their motivations, pros & cons, and implementations. As that one of this project goals is the development of a 3D

environment using computer graphics.

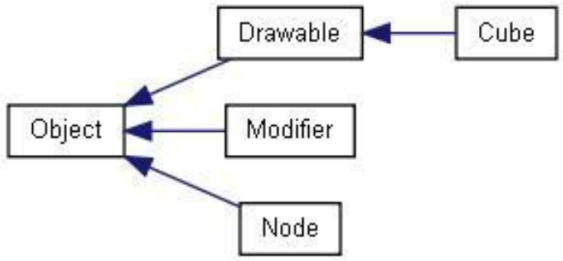
2.4.1 Software Architectures in the industry

When considering reference sources for software architectures (especially for 3D and simulation software) the industry related work, e.g. (LEONARD, 1999), stands out as the content theme doesn't appeal strongly to academic environment due to the evident product focused approach instead of theoretical.

Software architecture design choice has direct impact on the performance, as explored in (ALBRECHT, 2009b). Albrecht presented in GCAP 2009 a performance comparison of different data designs implementations.

The test scene tree consists of an object class containing general data (Object), class to render object (Drawable Cube), class to update geometrical transforms (Modifier), and container class which store access to objects (Node). The standard data for comparison is shown in Figure 22.

Figure 22 – Simple Scene Tree



Source: Adapted from (ALBRECHT, 2009b).

The test execution for data in the scene corresponds of:

- 11,111 nodes/objects in a tree 5 levels deep
- Every node being transformed
- Hierarchical culling of tree
- Render method being empty

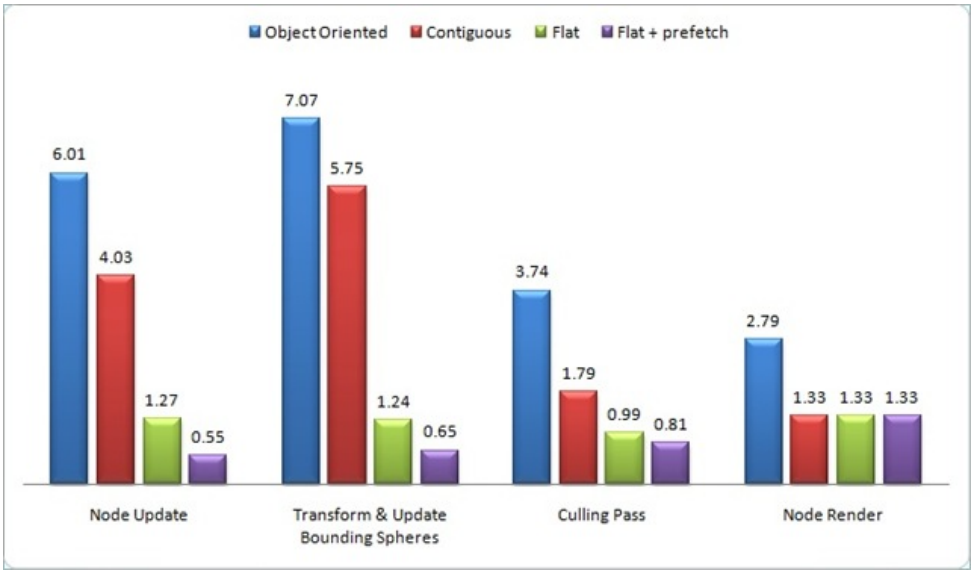
The different data organizations approaches iterate on the previous, and consisting of:

- Simple Object Oriented following the Scene tree organization

- Contiguous data allocation with use of homogeneous sequential sets
- Flat design by removing hierarchy of calls
- Flat design with prefetching due to predictable data accesses

Albrecht then presents the final test results for different data designs, shown in Figure 23. The most considerable performance gains were obtained by the continuous allocation of data (less encapsulation) and the flattening of hierarchy (less layers of abstraction).

Figure 23 – Cache impact of different data design approaches (values in milliseconds).



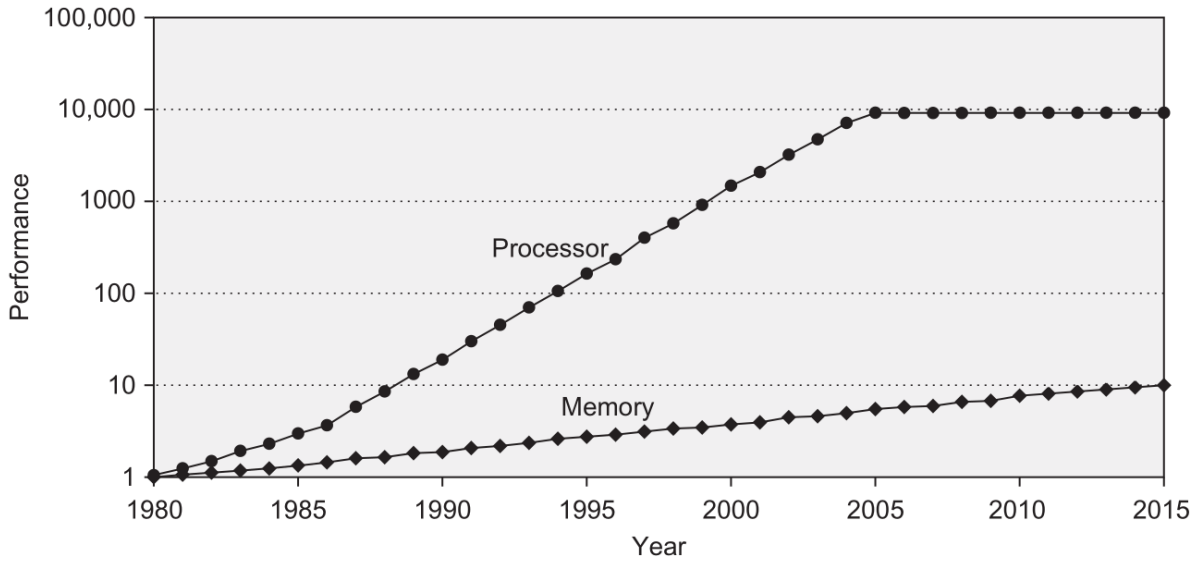
Source: Adapted from (ALBRECHT, 2009b).

By the previous shown study its evident that to obtain better performance what have to be explored is the way data is accessed and stored. This design approach is justified by the Processor-Memory Gap explored in the next subsection.

2.4.2 Processor-Memory Gap

The Processor-Memory Gap is described by the difference in time between processor memory request and the latency of DRAM access. It describes a problem where the processor performance improved in higher rate than the memory performance improved (HENNESSY; PATTERSON, 2017). Figure 24 shows that the gap increased over many years but is currently in a slowdown due to the lack of improvements in single processors. Although Figure 24 shows values for single core processors, the number of multiple cores in processors implies growth of the gap due to the demand of DRAM bandwidth as the number of cores grows.

Figure 24 – Single processor performance projections against the historical performance improvement in time to access main memory (Processor line shows the increase in memory requests per second on average. Memory line shows the increase in DRAM accesses per second).



Source: (HENNESSY; PATTERSON, 2017).

As this problem doesn't have an expected hardware solution, it ends up being the software layer job to work with the implied handicap when it appears as an obstacle for an application. In the next subsection we approach Data-Oriented Design, a pattern of software development that is less affect by the Processor-Memory gap problem as the number of memory accesses and cache misses tends to be reduced.

2.4.3 Entity Component Systems and Data-Oriented Design

Entity Component Systems (ECS) is an architectural pattern for software development. There is no concrete definition on its architecture scope, but it does often share Data-Oriented Design (DOD) techniques which focus on designing for performance (LLOPIS, 2011).

This architecture approach is better suited, for applications such as Game Engines or Real-Time Simulators, which are highly demanding processing software requiring constant main memory access to run in real-time.

Good case results of implementations for the ECS concept on the industry goes back decades (LEONARD, 1999) accompanied by the increased storage of data and the demand of its processing on applications. However, implementation efforts on ECS are still relevant, being focus of features for powerful software (JOHANSSON, 2018).

DOD main focus can be considered to be cache coherency. In consequence of this pursue

the architecture of code generated by it tends to be decoupled and simple in scope, due to not having a world-approximation as Oriented-Programming encourages.

As a mean to characterize Data-Oriented Design, (LLOPIS, 2011) consider its benefits and disadvantages. On Table 1 his discourse for each aspect of DOD are listed and simplified.

Table 1 – Data oriented design benefits (LLOPIS, 2011), adaptation as table by Authors of this study.

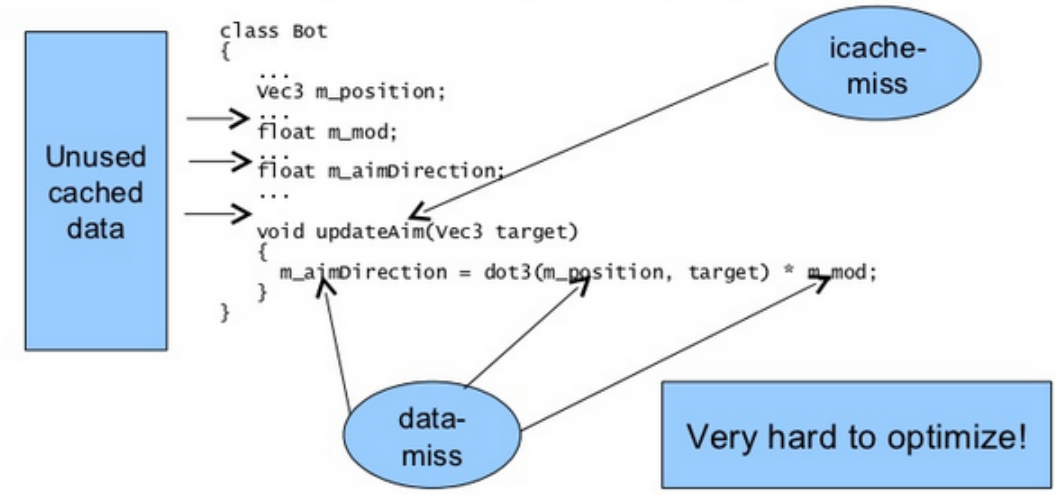
Benefits	**Description**
Cache utilization	Organization of data into large sequential blocks of homogeneous memory, processing it by running the same code on all its elements.
Parallelization	As we work from the data point of view, it becomes a lot easier to divide work up into parts that different cores can processes simultaneously with minimal synchronization.
Less code	Code that before was doing boring book-keeping, or getter/setters on objects, or even unnecessary abstractions, all go away.
Disadvantages	**Description**
It's different	So far, data-oriented design isn't taught in Computer Science curricula, ... so it is foreign to most team members.
Harder to see the big picture	Because of the emphasis on data and on small functions that transform data, it might be harder to see and express the big picture of the program [...]

A better grasp on the idea of DOD can be obtained with code examples. Consider the

two following implementations in C++ (COLLIN, 2010).

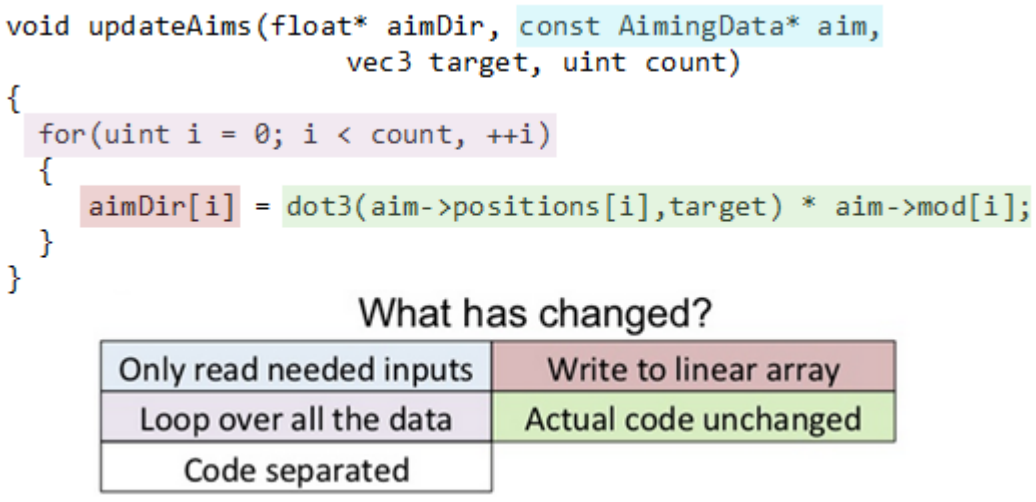
- Figure 25 uses Object-Oriented Programming and has the idea of creating a class Bot which stores data (attributes) and functionalities (methods);
- Figure 26 conforms to the Data-Oriented Design and has as focus generating the output data (in this case the "aiming direction"), doing that with a function which receives the input to be transformed.

Figure 25 – Object-Oriented Programming code example



Source: (COLLIN, 2010).

Figure 26 – Data-Oriented Programming code example



Source: Adapted from (COLLIN, 2010).

It is possible to better visualize the descriptions made in Table 1, by comparing the two previous implementations. Instead of trying to represent the problem and its elements using known items from the real world as OOP did, DOD focus on solving the problem in its most simpler iteration.

Considering previous data knowledge, as it is likely that there is more than one Bot and that "updateAims" will be called more than once, DOD optimizes memory requests by receiving as parameters the necessary input for processing (aligning it in cache) and doing the same operation for each cell of the "Bots" linear array. Thus eliminating cache misses and decoupling the code.

In conclusion, comparing DOP with the well spread Object-Oriented Programming (OOP), DOP implies the requirement of engineers to know their data, and that structure and code refactoring is continuously present during DOP development to eliminate bottlenecks (ALBRECHT, 2009a).

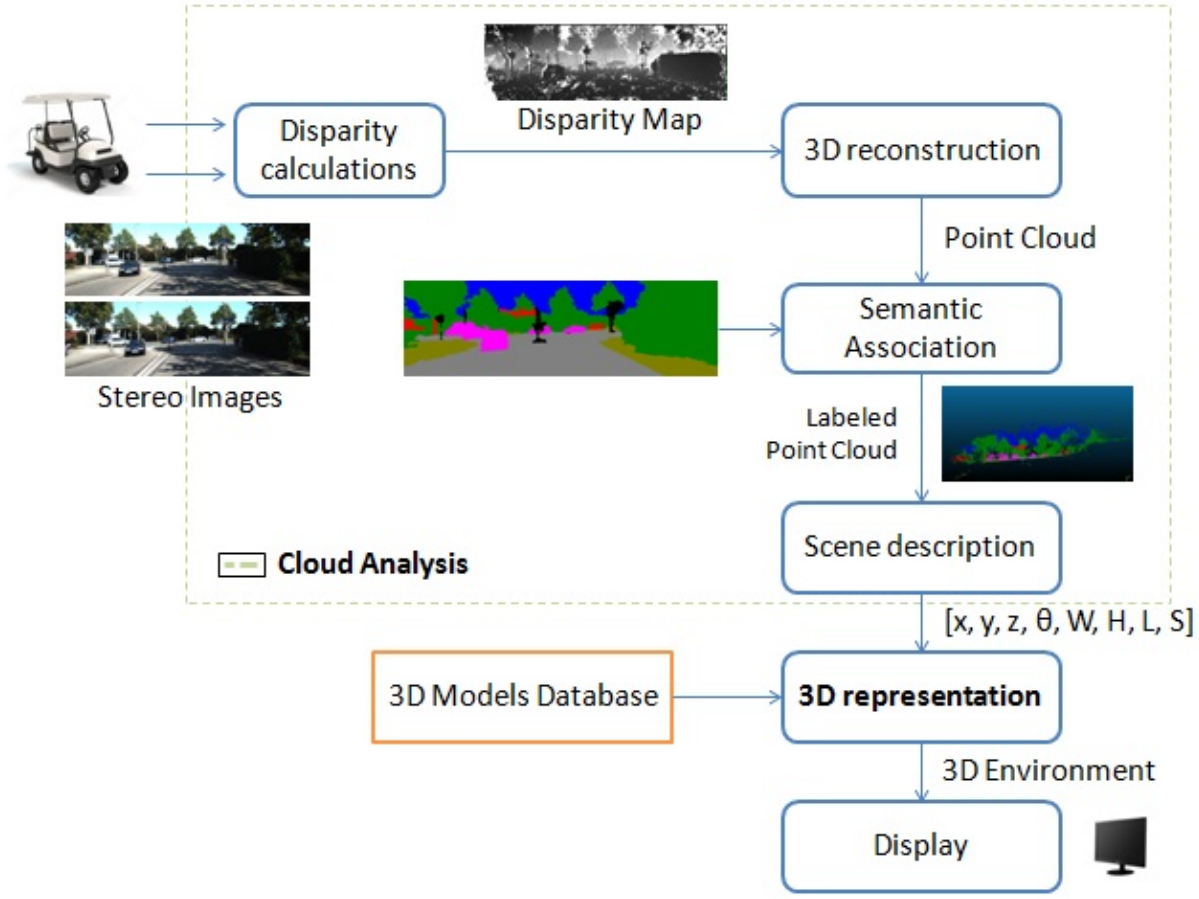
3 Development

The project consists of the development of two isolated systems which interacts with each other. The first system called Cloud Analysis has as input images of a stereo camera and its associated semantic image and as output information of center of mass and objects size in the image - street, house, car, tree, sky, sidewalk and unidentified empty spaces. It was decided to consider only the case scenario with one object of each category of the semantic image in the scene, because it is still not possible to distinguish between two objects of the same category.

The second system receives as input the values of the center of mass and size obtained by the first system and then generates a 3D environment of its own with all objects captured by the stereo camera. It uses the Entity Component System approach for its architecture and its implementation is in the C++ language using OpenGL graphics framework.

A representation of the whole system is shown in Figure 27, representing both point cloud analysis and 3D environment creation.

Figure 27 – Flow representing the entire system



Source: Authors of this study.

3.1 Cloud Analysis

Cloud Analysis is the system capable of generating a number of points which discriminates all categories of the semantic image and calculates their respective center of mass and sizes.

A simple example is to imagine a real urban environment with trees, cars, houses and others. Cloud Analysis has the function of analyzing this scenario and returning the mass center value and size of each of these elements (tree, car and others). It has as output a data tuple of each element, containing: coordinates in the 3D plane of its center of mass, Width, Height, Length and a descriptor that informs which category this element is inserted.

This system has well-defined steps: disparity image generation, point cloud generation, colored point cloud generation, and center of mass & size calculation.

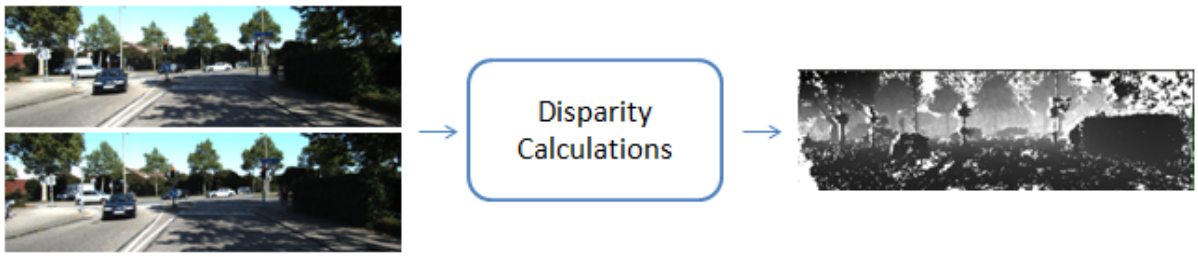
For a qualitative analysis, some different tests were performed and each partial flow

result can be visualized in the following topics.

3.1.1 Disparity Calculations

The first part of the Cloud Analysis system is the treatment of the two input images for the disparity calculation, as shown in Figure 28. This treatment was performed as follows: The camera calibration data was provided by KITTI dataset and the Sum of Squared Differences(SSD) calculation was used in order to discover the disparity value of each pixel.

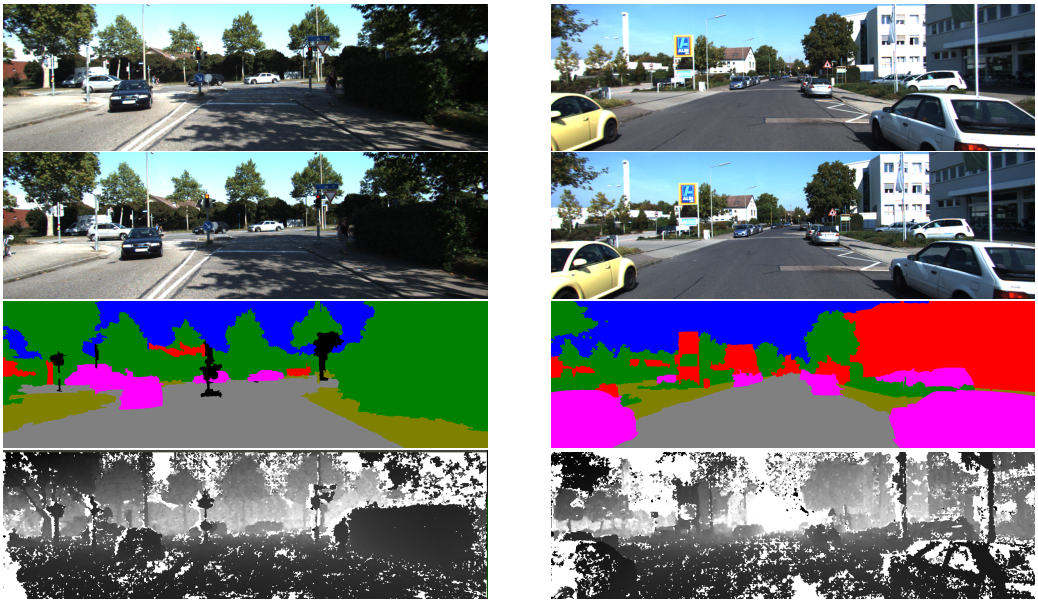
Figure 28 – Disparity calculation flow



Source: Authors of this study.

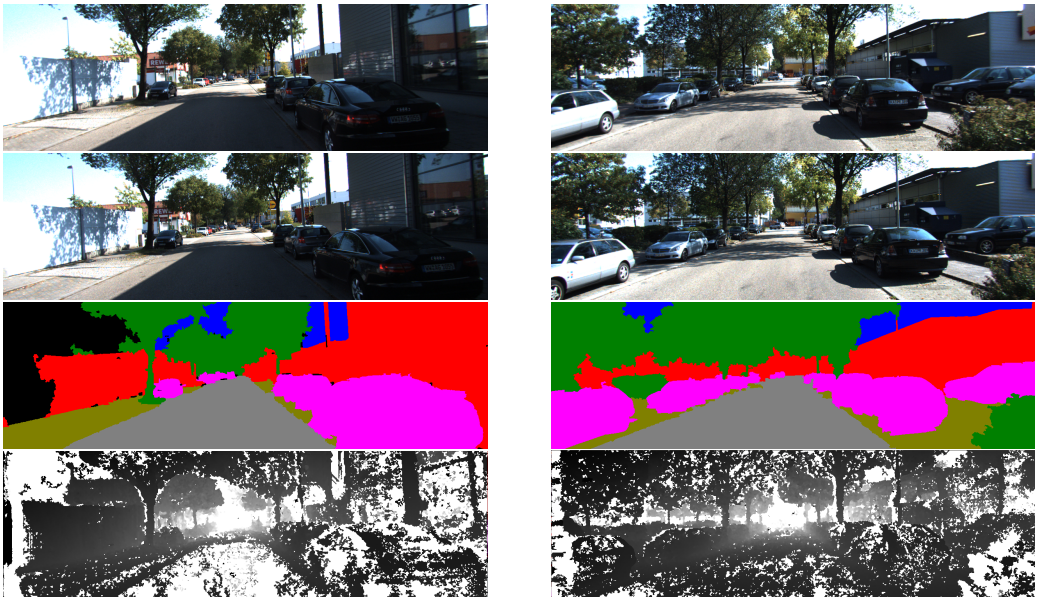
With camera calibration data, it is possible to define the positioning of each of the cameras and to align two images so that the already explained pixel assimilation is possible. The Figure 29 and Figure 30 show the results obtained in four different scenario tests.

Figure 29 – The first (left) and second (right) tests and results obtained for disparity image generation. The first three images of each column are inputs (the left and right cameras and the semantic image, respectively (VITOR, 2014)). The latter is an image of disparity generated.



Source: Authors of this study.

Figure 30 – The third and fourth tests and results obtained for a disparate image generation. The first three images are the system inputs (the left and right cameras and the semantic image, respectively (VITOR, 2014)). The latter is an image of disparity generated.



Source: Authors of this study.

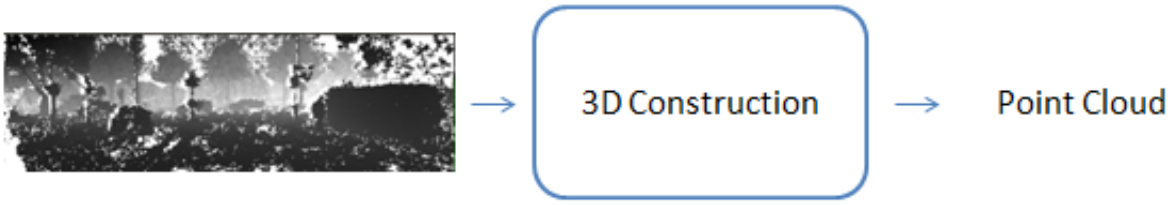
By analyzing the resulting disparity images, one can also perceive that there is a satisfactory proximity to reality. It is possible to perceive the depth of the elements in the images in

which the grayscale shows how close or far the components that make up the image are. The third test was shown to be less effective, both in the disparity image and in the color image.

3.1.2 3D Reconstruction

The following step in this flow is the point cloud generation, as shown in Figure 31.

Figure 31 – Point Cloud Generation Flow



Source: Authors of this study.

The Algorithm 1 contains the brief about the whole system developed to create a point cloud. As the Algorithm shows, the result is written in a *xyz* file with 3D points position. In addition to this output, the disparity image was saved so that it could be analyzed.

Using this algorithm, tests were performed using images obtained from Giovani's dataset (VITOR, 2014) shown in Figure 29 and Figure 30 including photos from the right and left camera, and semantic image.

Algorithm 1: Point Cloud Generation

input : Two OpenCV's cv::Mat matrices that represent the right and left photos. In addiction, the respective semantic image of the scene.
output : A file in *xyz* format that contains a set of 3D points associated to a value. This value represents the distance between the 3D point and the camera.

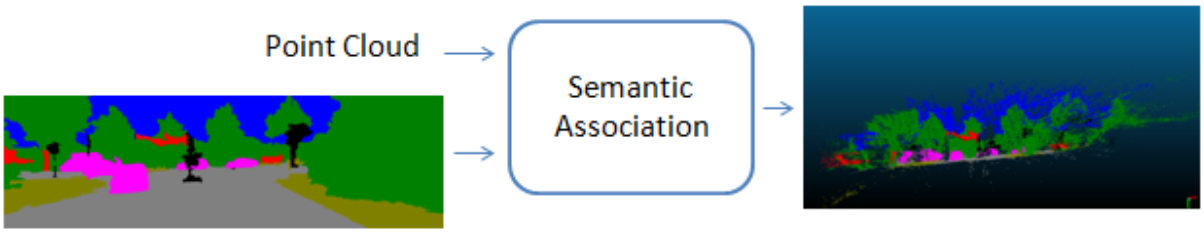
Camera Calibration according to data collected during camera setup;
Converting photos to grayscale images;
Setup specific values of calibration to use OpenCV's cv::StereoSGBM class;
Disparity image generation by the cv::StereoSGBM class with stereo images as input;
Normalize disparity image to 0-255 scale;
Create a Mat matrix called Points that store 3D points information of the cloud point from the OpenCV's cv::reprojectImageTo3D function;
for $i \leftarrow 0$ **to** *points.rows* **do**
    for $j \leftarrow 0$ **to** *points.cols* **do**
        Writing in a xyz file the X, Y and Z position of each point and the intensity value (grayscale) related to the distance between point and camera;
Printing the result;

The image of a point cloud obtained from the photos of left and right is a set of grayscale points with low visualization value as there is to differentiation of entity representation. For this reason, point cloud results were expressed only by the end, in the Colored Point Cloud that uses semantic information to define entities.

3.1.3 Semantic Association

Semantic association means a approach to create a point cloud colored, as shown Figure 32.

Figure 32 – Semantic Association Flow



Source: Authors of this study.

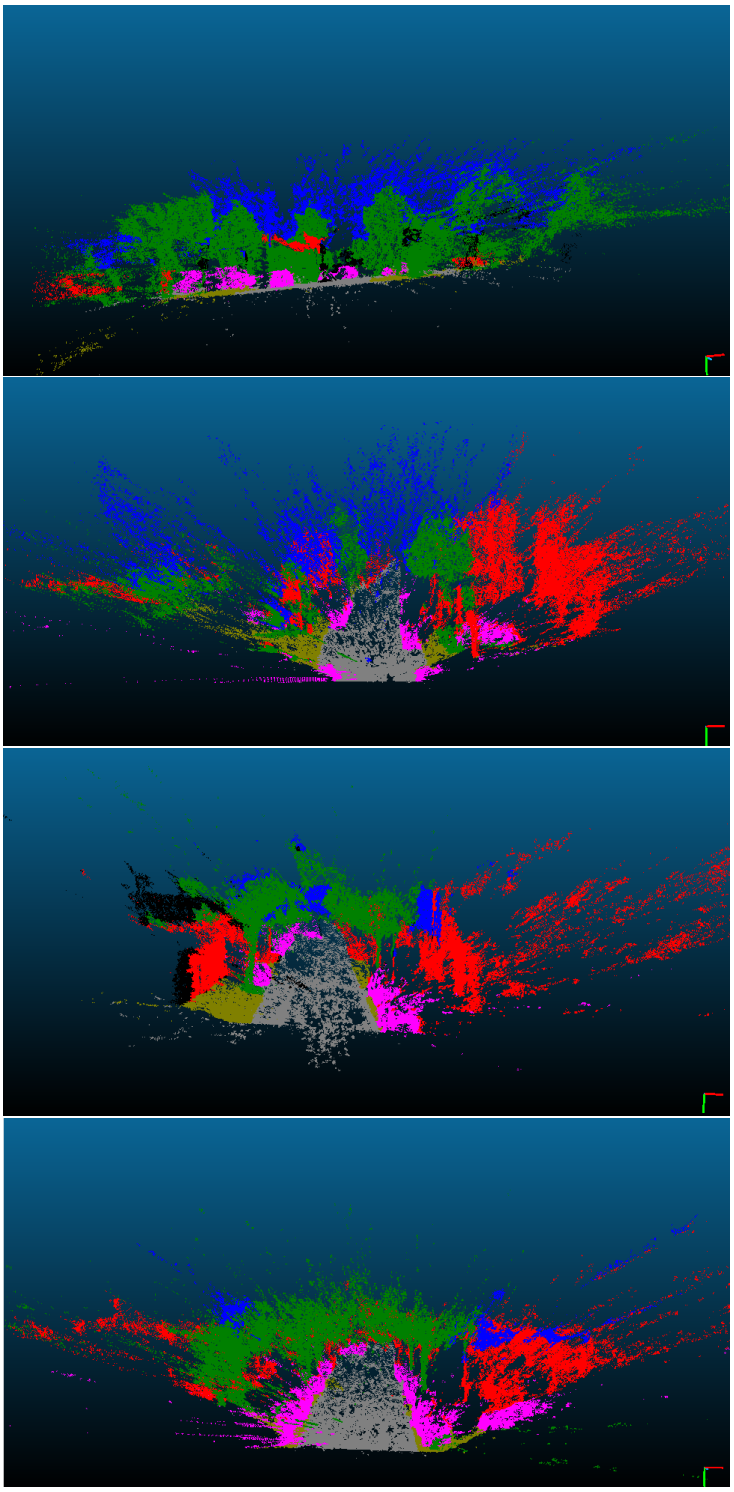
The following steps occurred from the following algorithm: Reading the whole Point Cloud xyz file generated in Algorithm 1; Reading the semantic image and record the respective RGB (Red, Green and Blue) values to each pixel of the image; Gathering the two values into a single file that contains the following attributes in each row: x, y, and z position of the pixel, its grayscale value and its RGB values.

There is a program called Cloud Compare that reads *xyz* files and shows the point cloud with color and interact with them. Using this program we were able to display the results shown in Figure 36.

In Figure 36, each image represents the result of a test, first picture represents first test and the numeration goes top-down. The first test proved to be the most efficient of all which showed a wealth of detail in both near and camera and distant images. The second and fourth tests were also close to reality, especially near the camera.

The worst test generated was the third test. In this test, the disparity image generated and its point cloud interpreted much of the street as something below the ground, as shown in Figure 36 (gray pixels filling the subsoil). It seems that it is related to the fact that the section with few details is very dark, which leads to difficulties in the generation of the disparity image. In addition, it is noted that the semantic image for this photo interpreted as void space much of the sky on the left.

Figure 33 – Colored Point Cloud results



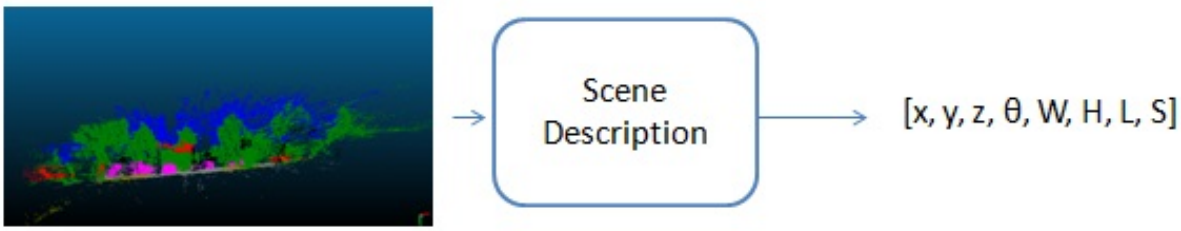
Source: Authors of this study.

3.1.4 Scene Description

After the 3D coordinate data and colors have been acquired in the algorithm to obtain the colored point cloud, it is necessary to discriminate the categories and calculate the center of

mass and size of each one of them. The flow to do this is represented in Figure 34. Note that x, y and z represent the x, y, and z coordinates of the center of mass; Theta is the direction of the object in the 3D scenario; W, H and L are the width, height and length that will define the space that the object occupies and S represents what kind of object it is (such as house, vehicle, street and others).

Figure 34 – Scene Description Flow



Source: Authors of this study.

The project is still not able to discriminate a car from another car in the same scene, or a tree from another. Thus, it was decided that the system would assume that there would be only one possible entity: one car, one tree, one building and so on. After the system is able to operate under these conditions it would proceed to a more advanced stage of distinguishing objects within the same category.

Before calculating the center of mass and the size of the categories contained in the scene, it was necessary to separate them. Thus, an algorithm was developed to read the *xyz* file of the colored point cloud and to separate into a data structure each of the points with equal colors. All the pink objects were stored in a structure called a car, the blues in the structure called sky and the logic remain for the other categories that the semantic image can distinguish.

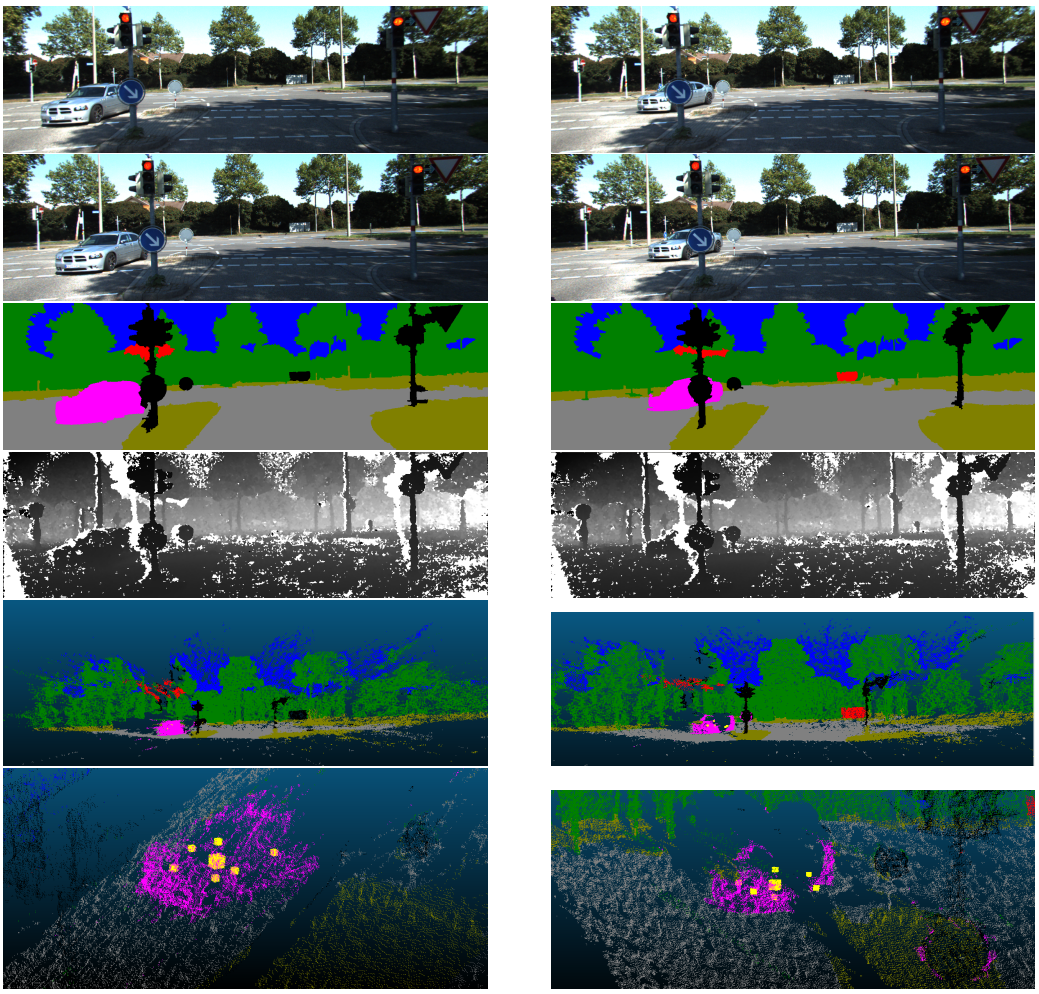
Since the assumption is that the system has one element of each category, it is sufficient to calculate the center of mass (Equation 2.7, Equation 2.8, Equation 2.9) of each structure separately. In the end, there are seven different center of mass values: One for car, building, tree, sky, sidewalk, street and unidentified void space.

At this moment, only the calculation of the average of points located in the different directions in relation to the center of mass is done, forming a type of box that would delimit the object boundaries.

To validate the center of mass values, some tests were performed where there is only one element of a certain category in the image. This element had its center of mass calculated and printed on Point Cloud Colored. The first test performed can be seen in Figure 35. These are two different images in which there is only one car - one car in the first column and one car behind a sign in the second column. In this test it can notice the entire development flow of

Cloud Analysis so far: Receiving three images as input and generating a disparity map, a Point Cloud, and output tuples with the center of mass information and size.

Figure 35 – First and second tests performed with only one car on the scene



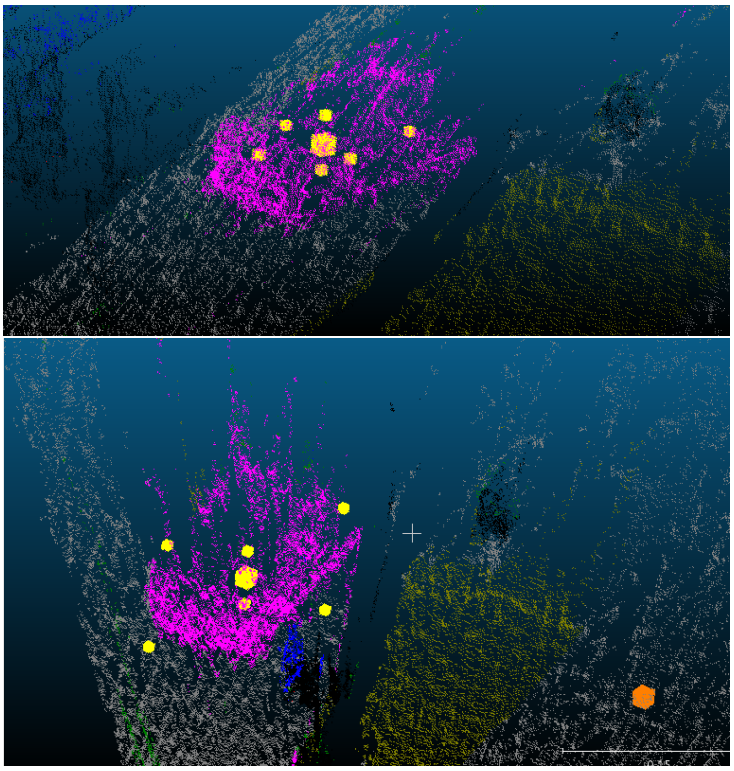
Source: Authors of this study.

In the last image you can see that the center of mass of the car is also being printed - visually close to what would really be the center of it. On the other hand, the size limits described as small yellow squares around the center of mass were not very efficient. All points are assuming values below the actual size of the vehicle.

It is also noted in the second column that, although the sign disrupts the complete view of the car, the center of mass values remained faithful.

This test quoted above was calculated from the average points located on each of the six sides around the center of mass, as explained in the Literature Review. In order to improve the obtained results, it was decided to apply the variance method. The result and difference can be seen in Figure 36.

Figure 36 – Average method vs Variance method



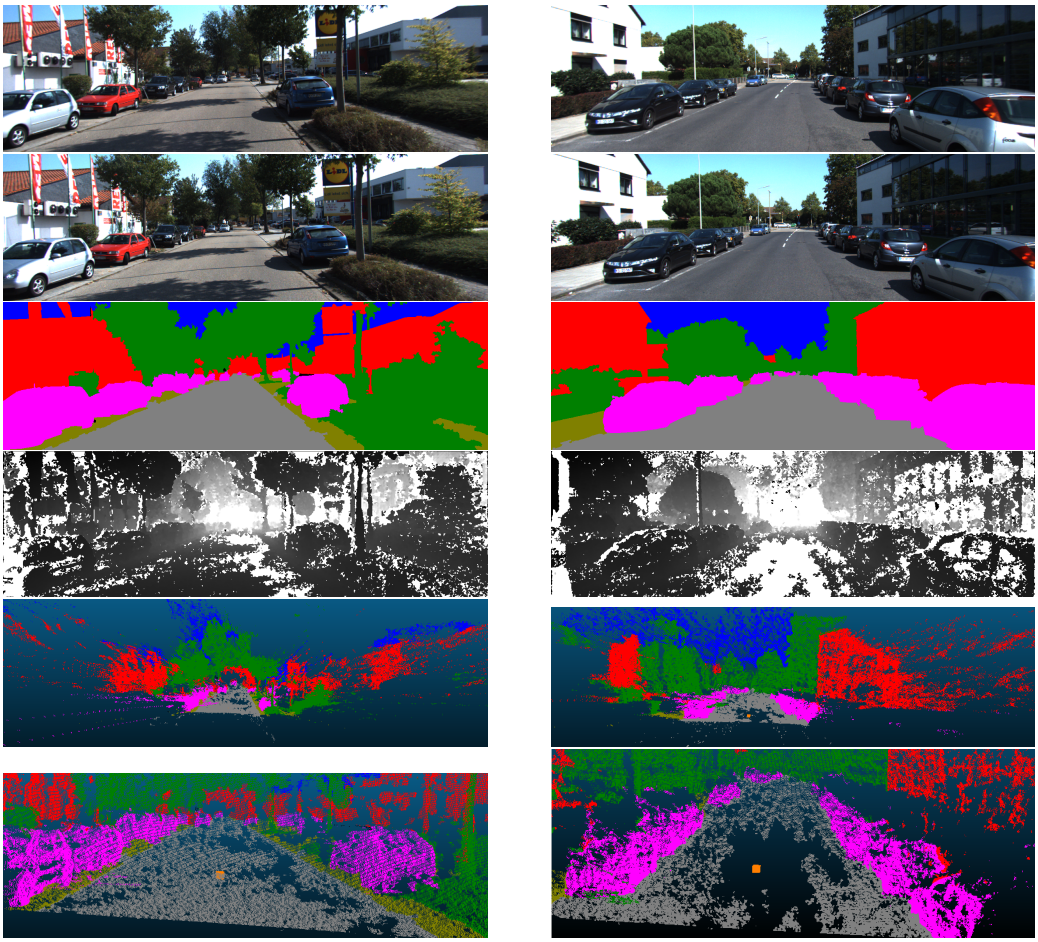
Source: Authors of this study.

It can see that the bounding box is still not perfect. Before it was too small and now, it is bigger than the real. Despite this, both results proved adequate to create a representation of the car's bounding box.It was defined to used the variance.

Another test was performed considering situations where there is only one street. The function of this test is to validate the center of mass calculation of other categories besides vehicles. The street will have a different treatment in the 3D environment, however, the output of Cloud Analysis will be similar to vehicles. As shown in Figure 37, the street's center of mass is located at a visually expected point, marked orange in the images.

The results obtained from the center of mass (both in Figure 35 and in Figure 37) are defined in a range of values different from the 3D environment. The functions and calculations performed for the Point Cloud generation return as a result values approximately between 5 and -5 floating values - which is far from representing a metric close to the metric of reality.

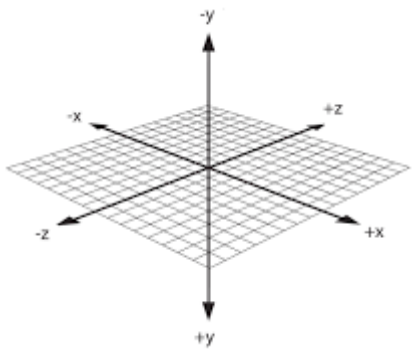
Figure 37 – First and second tests performed with only one street on the scene



Source: Authors of this study.

In addition, it was noted that the Cartesian plane used follows the pattern shown in Figure 38, where the camera located in the car is taken as the axis (0,0,0).

Figure 38 – Cartesian plane used in Point Cloud



Source: Authors of this study.

The mass centers of the two cars analyzed in the test of Figure 35 can be seen in Table 2 while that of the test of Figure 35 can be seen in Table 3 and them exemplified how lower these numbers are.

Table 2 – X, y, and z axis coordinates of the test of Figure 35

Teste	X-axis	Y-axis	Z-axis
Car 1	-0.417017	0.0258149	0.628768
Car 2	-0.427101	0.0166311	0.78519

Table 3 – X, y, and z axis coordinates of the test of Figure 37

Teste	X-axis	Y-axis	Z-axis
Street 1	-0.193943	0.0495407	0.698221
Street 2	-0.153351	0.0435096	0.700954

Therefore, one has to challenge the construction of a metric to convert these small coordinate values into useful values in the 3D environment

3.1.5 Errors analysis.

This system has well-defined stages for its development and this has the benefit of the facility in identifying errors before are propagated.

The first stage is camera calibration. The calibration itself can carry various types of errors and, according to (MAZON; ZACCHI; MARTINS, 2011), these can be categorized as:

- Gross Errors: This error is caused by the inability or neglect of the measurer;
- Accidental (or random) mistakes: These errors are caused by the imperfection of our senses, atmospheric irregularities and inevitable minor errors in instrument construction;
- Systematic errors: Errors caused by non-conforming measures (such as dilated timing, bent bolt and dumbbell post);
- Rounding Errors: Typically caused by the use of numbers with a few decimal places.

It is possible that the system is subject to all these errors, although there are efforts to avoid them, such as using as many decimals as possible, both in calibration and in subsequent steps.

In the following steps of calculating the disparity map it is possible to highlight some reasons for the difficulties already reported, such as low-contrast image regions. This problem

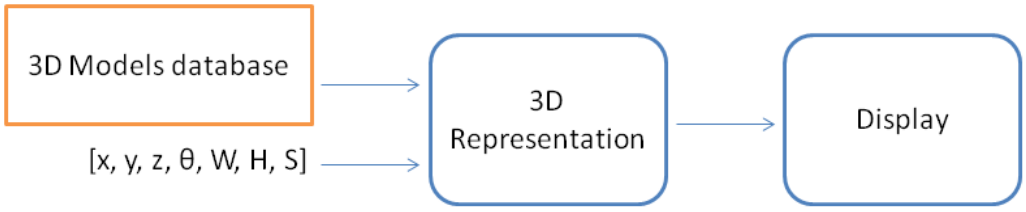
impacted the results shown in order to create holes in the disparity map generating a noise that propagated to the end of the flow and hampered the faithful acquisition of the features.

Another factor that contributed to the difficulty in generating a plausible graphical environment was the fact that the Point Cloud is generated by small coordinates. These values were between 5 and -5, with six decimal places in each coordinate. These low numbers contributed to the propagation of the error, since any small rounding in the center of mass calculation or in the bounding box will already result in very disparate values when passed to the 3D environment.

3.2 3D environment

The 3D environment is expected to receive as input the identified entities, their information (position, direction and size), and standard 3D Model that better represent it, and as output display an approximate scene from what was identified by the computer vision algorithm. A general flow of expected input and output is shown in Figure 39.

Figure 39 – 3D representation flow



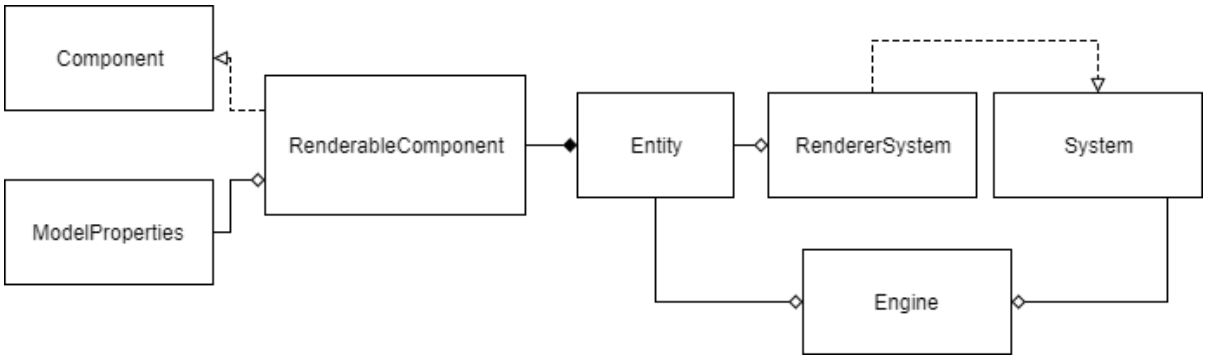
Source: Authors of this study.

It was decided to develop the 3D environment system using C++ in conjunction with the graphical framework OpenGL, as it was considered the possibility of resource constraints, such as usage of CPU & Memory and processing time. The system uses the OpenGL Shading Language (GLSL) for manipulation of VertexShaders and FragmentShaders. These choices, aside from the previous points, were based on authors' affinity with languages and frameworks while also allowing for more control on the program execution and better comprehension of the graphics pipeline of modern frameworks since both options are in a considerable low abstraction layer.

3.2.1 Software Architecture

Following the Data-Oriented Design concept presented in the Literature Review chapter, an Entity Component System was projected as shown in Figure 40.

Figure 40 – 3D environment simplified diagram



Source: Authors of this study.

The control class of the system can be identified in Figure 40 as the class Engine, it has methods to initialize the main loop of the software and contains calls for defined Systems. Aside from the importance of the class Engine, three other classes deserve a more thorough description:

- Entity

  Represents the objects of the program, it maps the components existent in the software such as objects in a scene.

- Component

  Represents the differentiation of data stored in the program, it is mapped by the Entity class and all it's processing is dependent and executed by System classes.

- Systems

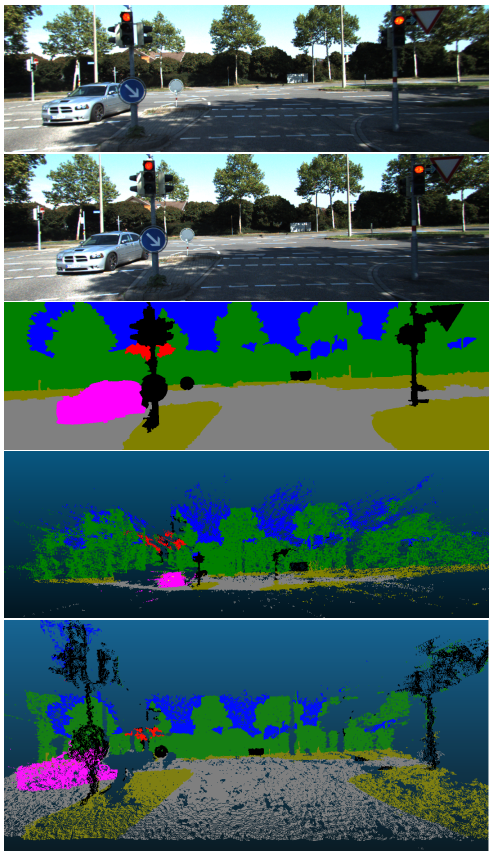
  Are the real operators of the engine, it runs a defined data transformation (like defined on RendererSystem) on all the correct data of the program in a sequential manner). For example, RendererSystem conducts data transformation of all the entities which contains the Component's inherited class "RenderableComponent".

3.2.2 Display Evaluation

The 3D environment is capable of loading objects with information provided by the Cloud Analysis system. The input is the position coordinates (x, y, z), its orientation ($theta$), its size (Width, Length, Height) and semantic class (Car: 1, Street: 2).

To evaluate the functionality of the 3D environment, a scene with one car was selected as shown in Figure 41.

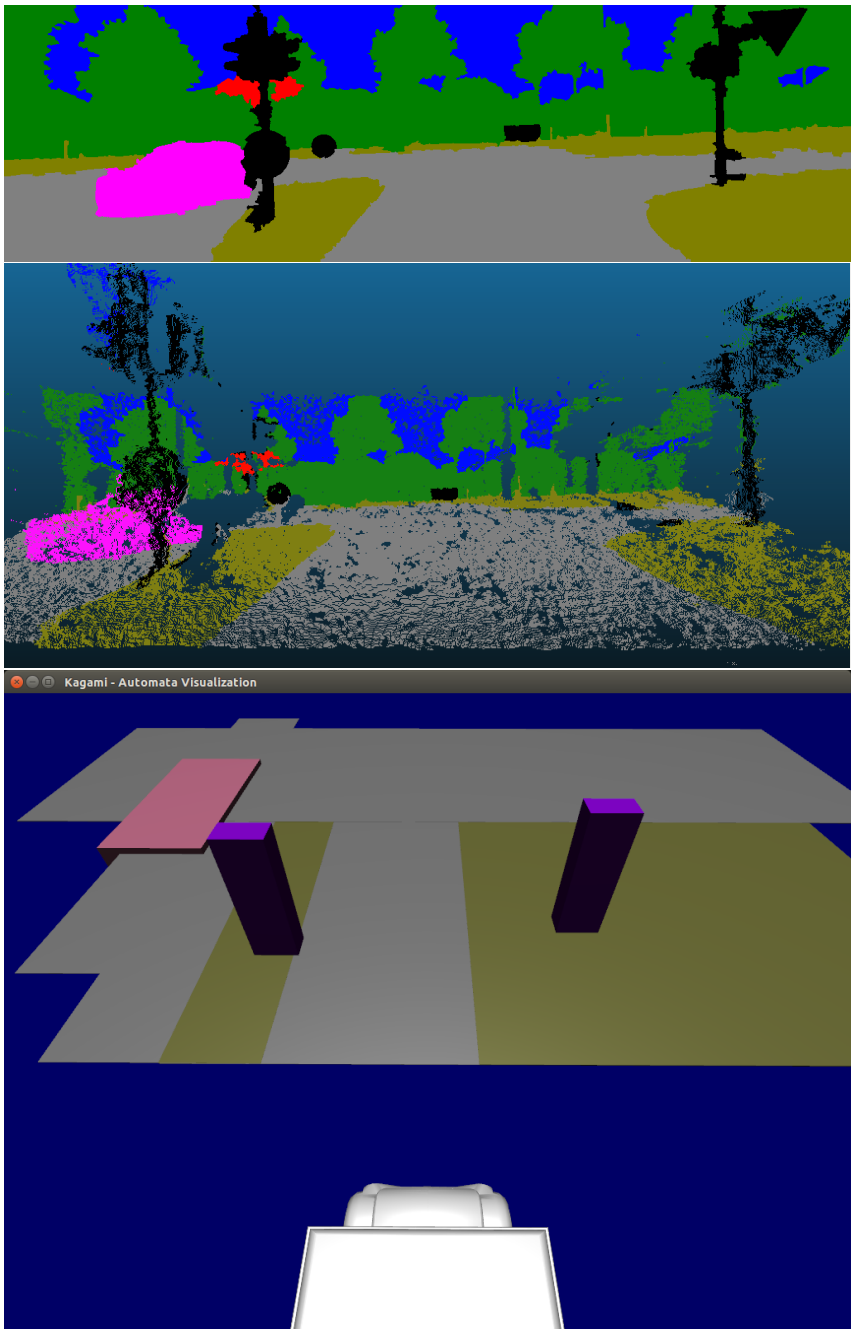
Figure 41 – Scene for visualization test



Source: Authors of this study.

Thus, using Figure 41 the point cloud as reference (bottom image), a baseline 3D environment input was crafted using (VITOR, 2014) semantic colors as shown in Figure 42.

Figure 42 – Crafted baseline

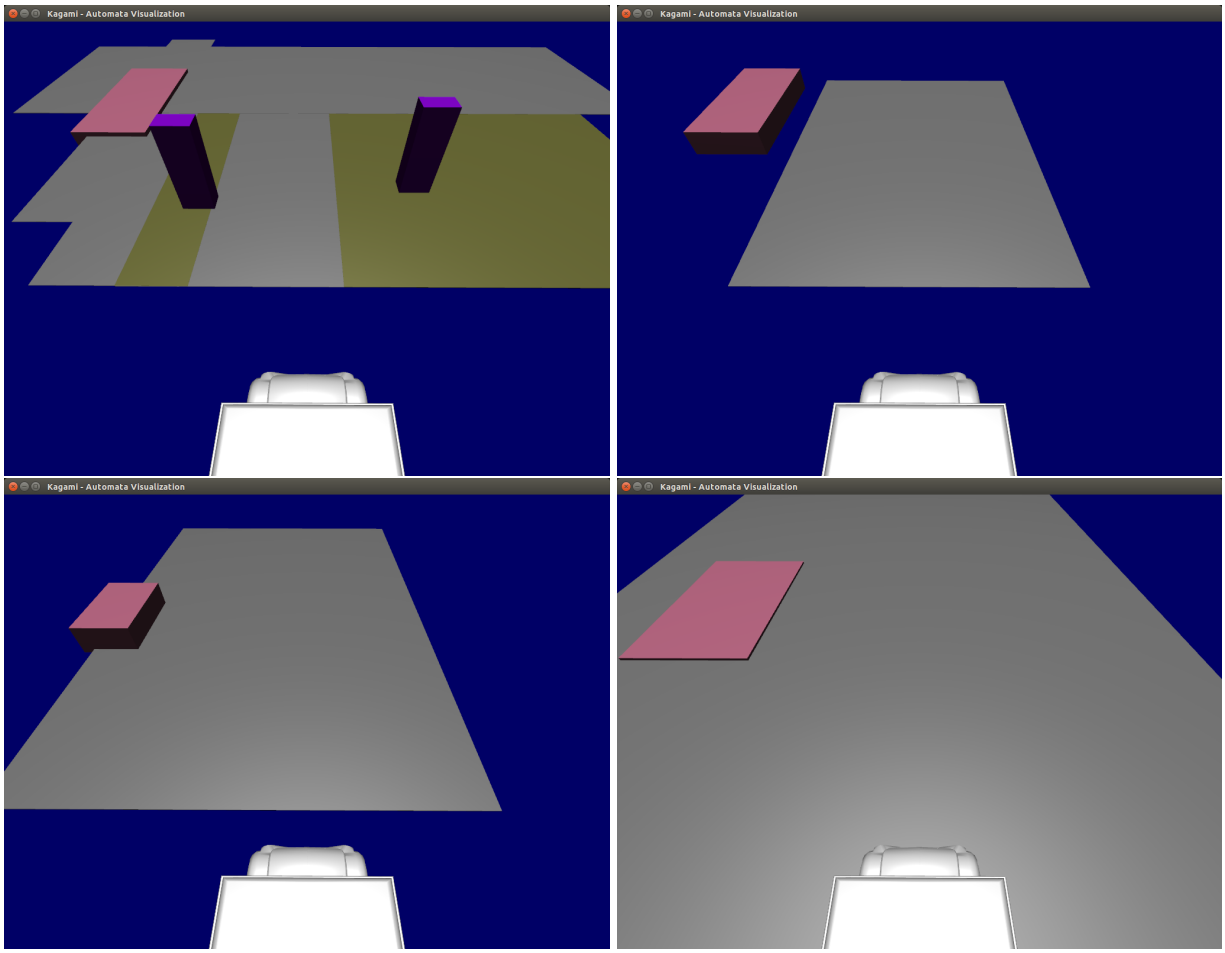


Source: Authors of this study.

To evaluate the performance of the first system feature extractors, Figure 43 shows:

- Top-Left: Crafted baseline;
- Top-Right: Simplified crafted baseline showing only car and street;
- Bottom-Left: Center of mass feature;
- Bottom-Right: Variance feature.

Figure 43 – Feature extractors comparison



Source: Authors of this study.

Comparing feature extractors, it's possible to say that the center of mass method showed better results in relation to position but bad due to it's smaller scale in relation to the baseline, while the variance method have better coverage of the baseline car. When considering a user point of view for an interface, the variance method performed better for car representation.

Since the current Cloud Analysis doesn't provide more than one entity for one semantic class, the street object is on the comparison picture for spatial reference.

Case scenarios are limited due to the constraint of one type of semantic class. However, it's possible to confirm the visualization functionality of the 3D environment.

4 Conclusion

It's possible to affirm that an end-to-end version of the proposed system was properly elaborated & developed, and that the set goals of the research were achieved. However, better results for the visualization were expected when considering that only one semantic class had to exist to enable feature extraction. Thus, limiting the number of test scenarios and variety in the 3D environment representation.

For better performance of the first system, a more elaborate method that could be explored in future work would be to consider clustering entities with the same semantic image information. Thus, still making use of the advantageous semantic information offered by the autonomous-driving algorithm for its decision making, and now considering more than one entity with the same semantic class in a scene. A second point of improvement would be to explore approaches to reduce point cloud noise by using more elaborate depth estimation methods.

To enhance the second system results, future work could involve conducting tests on real embedded systems for autonomous-driving while profiling the software performance, and optimizing it when needed. A parallel approach could involve the building of a display prototype to receive real users feedback and evaluation.

Finally, an approach on surveillance and monitoring could be explored and evaluated. Possibly considering the use o sensors from more than one vehicle connected by network, and also exploring the possibility of map update following the graphical display procedure achieved in this research.

Bibliography

ALBRECHT, T. *The Latency Elephant*. 2009. Available at: <http://seven-degrees-of-freedom.blogspot.com/2009/10/latency-elephant.html>. Accessed on: 23 set. 2018. Cited in page 35.

ALBRECHT, T. *Pitfalls of Object Oriented Programming*. 2009. Available at: <http://harmful.cat-v.org/software/OO_programming/_pdf/Pitfalls_of_Object_Oriented_Programming_GCAP_09.pdf>. Accessed on: 23 set. 2017. Cited 2 times in pages 29 and 30.

BATISTA, N. A. R.; REGIS, C. D. M. *Obtenção da disparidade e dos mapas de profundidade em vídeos 3D*. 2013. Available at: <http://periodicos.ifpb.edu.br/index.php/principia/article/viewFile/113/88>. Accessed on: 03 set. 2018. Cited in page 24.

CARLA. *Carla Official Website*. 2017. Available at: <http://carla.org/>. Accessed on: 31 ago. 2018. Cited in page 15.

COELHO, M. C. F. S. P.; TAVARES, J. M. R. S. *Toolbox de Calibração de Câmaras para Matlab*. 2003. Available at: <https://web.fe.up.pt/~tavares/downloads/publications/relatorios/Relatorio_toolbox.pdf>. Accessed on: 03 set. 2018. Cited in page 18.

COLLIN, D. *Introduction to Data-Oriented Design*. Electronic Arts / DICE, 2010. Available at: <http://www.slideshare.net/DICEStudio/introduction-to-data-oriented-design>. Accessed on: 22 set. 2018. Cited in page 34.

COSTA, A. M.; PESCO, S. *Visualização e Reconstrução para Nuvem de Pontos*. 2009. Available at: <http://www.puc-rio.br/pibic/relatorio_resumo2009/relatorio/mat/alexandre.pdf>. Accessed on: 12 out. 2018. Cited in page 25.

DOSOVITSKIY, A. et al. CARLA: An open urban driving simulator. In: *Proceedings of the 1st Annual Conference on Robot Learning*. [S.l.: s.n.], 2017. p. 1–16. Cited 2 times in pages 16 and 17.

GEIGER, A. et al. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013. Available at: <http://www.cvlibs.net/publications/Geiger2013IJRR.pdf>. Accessed on: 03 abr. 2017. Cited 2 times in pages 9 and 19.

HALLIDAY, D.; RESNICK, R.; WALKER, J. Centro de massa e momento linear. In: ____. *Fundamentos da Física*. 8. ed. Rio de Janeiro: LTC - Livros Técnicos e Científicos Editora S.A., 2009. chap. 9, p. 218–221. Cited in page 27.

HENNESSY, J. L.; PATTERSON, D. A. *Computer Architecture: A Quantitative Approach*. 6. ed. Cambridge: Morgan Kaufmann, 2017. Cited 2 times in pages 30 and 31.

HOUSE, M. *"Explanation of a way to represent and implement entity component systems"*. 2012. Available at: <https://gamedev.stackexchange.com/questions/31473/what-is-the-role-of-systems-in-a-component-based-entity-architecture>. Accessed on: 01 set. 2017. Cited in page 10.

JOHANSSON, T. *Job System Entity Component System (Presented by Unity Technologies)*. Game Developers Conference (GCD), 2018. Available at: <http://schedule.gdconf.com/session/job-system-entity-component-system-presented-by-unity-technologies/856383>. Accessed on: 22 set. 2018. Cited in page 31.

KENTLEY-KLAY, T. *Simulation for Autonomous Driving - Zoox*. 2017. Available at: <https://youtu.be/otmxoK4lCNw?t=23m14s>. Accessed on: 31 ago. 2017. Cited 2 times in pages 11 and 12.

LEONARD, T. *Postmortem: Thief*: The dark project. Gamasutra, 1999. Available at: <http://www.gamasutra.com/view/feature/131762/postmortem_thief_the_dark_project.php?page=1>. Accessed on: 22 set. 2018. Cited 2 times in pages 29 and 31.

LLOPIS, N. High-performance programming with data-oriented design. In: LENGYEL, E. (Ed.). *Game Engine Gems 2*. Natick: A K Peters, 2011. chap. 15, p. 251–261. Cited 5 times in pages 6, 10, 31, 32, and 33.

MAZON, H.; ZACCHI, G. P.; MARTINS, R. Calibração de câmeras e fontes de erros para triangulação fotogramétrica. *Anais XV Simpósio Brasileiro de Sensoriamento Remoto - SBSR*, 2011. Available at: <http://marte.sid.inpe.br/col/dpi.inpe.br/marte/2011/07.29.19.16/doc/p1602.pdf>. Accessed on: 07 out. 2018. Cited in page 47.

MEIRELES, R. *Interface com computador por Controlo Visual de Cursores*. 2002. Available at: <https://paginas.fe.up.pt/~ee97107/Relatorio_de_Projecto_FINAL_PARTE_2.pdf>. Accessed on: 03 set. 2018. Cited in page 18.

MONTGOMERY, D. C.; RUNGER, G. C. Mean and variance of a discrete random variable. In: ____. *Applied Statistics and Probability for Engineers*. 5. ed. Hoboken: John Wiley and Sons, 2011. chap. 3.4, p. 74–76. Cited in page 28.

OPENCV. *Camera Calibration and 3D Reconstruction*. 2018. Available at: <https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html>. Accessed on: 03 set. 2018. Cited in page 21.

PARK, J.; BYUN, S.-C.; LEE, B.-U. Lens distortion correction using ideal image coordinates. In: CONSUMER ELECTRONICS. 2009. Available at: <https://www.researchgate.net/publication/224599781_Lens_Distortion_Correction_Using_Ideal_Image_Coordinates>. Accessed on: 22 out. 2018. Cited in page 18.

PILLAI, S.; AMBRUS, R.; GAIDON, A. Superdepth: Self-supervised, super-resolved monocular depth estimation. 2018. Available at: <https://arxiv.org/pdf/1810.01849.pdf>. Accessed on: 12 ago. 2018. Cited 2 times in pages 14 and 15.

RAO, R. *Stereo and 3D Vision*. 2009. Available at: <https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect16.pdf>. Accessed on: 03 set. 2018. Cited 3 times in pages 22, 23, and 27.

RUMBAUGH, J. et al. *Object-Oriented Modeling and Design*. 1st. ed. New Jersey: Prentice Hall, 1991. Cited in page 10.

SILVA, L. de A. *Aproximação Planar por Partes Para Reconstrução 3D Densa*. Master's Thesis (Master's Thesis), 2016. Available at: <http://repositorio.ufes.br/jspui/handle/10/9558>. Accessed on: 22 set. 2018. Cited 3 times in pages 19, 20, and 21.

TATARCHENKO, M. et al. Tangent convolutions for dense prediction in 3d. 2018. Available at: <http://vladlen.info/papers/tangent-convolutions.pdf>. Accessed on: 12 ago. 2018. Cited 2 times in pages 13 and 14.

VASS, G. Applying and removing lens distortion in post production. In: THE SECOND HUNGARIAN CONFERENCE ON COMPUTER GRAPHICS AND GEOMETRY. 2003. Available at: <http://www.vassg.hu/pdf/vass_gg_2003_lo.pdf>. Accessed on: 22 set. 2018. Cited in page 18.

VITOR, G. B. *Urban Environment Perception and Navigation using Robotic Vision*: Conception and implementation applied to autonomous vehicle. Phd Thesis (PhD Thesis), 2014. Cited 8 times in pages 5, 9, 25, 26, 27, 39, 40, and 50.

ZOOX. *Zoox LinkedIn Page*. 2014. Available at: <https://www.linkedin.com/company/zoox-inc/>. Accessed on: 02 ago. 2018. Cited in page 11.

ZOOX. *Zoox Official Website*. 2014. Available at: <https://zoox.com/>. Accessed on: 31 ago. 2017. Cited in page 11.

ZOOX. *Zoox Fully Autonomous Driving*. 2018. Available at: <https://www.youtube.com/watch?v=868tExoVdQw>. Accessed on: 02 ago. 2018. Cited 2 times in pages 12 and 13.

ZOOX. *Zoox Youtube Channel*. 2018. Available at: <https://www.youtube.com/channel/UCh5q-FtihPqzTbgEkZQRy3g/videos>. Accessed on: 02 ago. 2018. Cited in page 12.