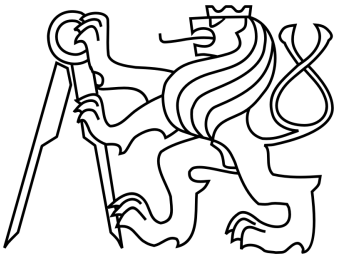
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics



MASTER'S THESIS

3D map estimation from a single RGB image

Author: Bc. Matěj Račinský

Thesis supervisor: doc. Ing. Karel Zimmermann, Ph.D.

In Prague, May 2018

Author statement for thegraduate thesis:

I declare that the presented work was developed independently and that I have listed all the sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the presentation of university theses.

Prague, date _____ _____

signature

Název práce: Odhad 3D mapy z jednoho RGB obrazu

Autor: Bc. Matěj Račinský

Katedra (ústav): Katedra kybernetiky

Vedoucí bakalářské práce: doc. Ing. Karel Zimmermann, Ph.D.

e-mail vedoucího: zimmerk@fel.cvut.cz

Abstrakt Tato práce se zabývá využitím virtuálních světů z počítačových her jakožto zdroje dat pro strojové učení, a odhadem voxelové mapy z jednoho RGB obrázku za pomoci hlubokého učení. Tato práce zahrnuje skripty pro napojení se na PC hru GTA V a sběr dat z ní pro tvorbu automaticky anotovaných datasetů, a implementaci hluboké neuronové sítě v TensorFlow.

Klíčová slova: Deep learning, Machine learning, GTA V, virtual world, depth estimation, voxelmap estimation, RAGE

Title: 3D map estimation from a single RGB image

Author: Bc. Matěj Račinský

Department: Department of Cybernetics

Supervisor: doc. Ing. Karel Zimmermann, Ph.D.

Supervisor's e-mail address: zimmerk@fel.cvut.cz

Abstract In this thesis we explore virtual worlds used as datasoutce for machine learning and voxelmap estimation from single RGB image with deep learning. This thesis describes principles and omplementation of hooking into GTA V and gathering data from it to create automatically annotated dataset, and implementation of deep neural network in TensorFlow.

Keywords: Deep learning, Machine learning, GTA V, virtual world, depth estimation, voxelmap estimation, RAGE

CONTENTS

CHAPTER
ONE

INTRODUCTION

This thesis aims to solve two problems. The first problem is the lengthy and slow process of manual dataset creation and annotation, and the second problem is voxelmap estimation from single RGB image. Due to increasing interest in synthetic datasets, this thesis aims to be the documentation for using GTA V as simulator for creation of synthetic datasets.

1.1. Problems with machine learning datasets

In recent years, both machine learning and deep learning has experienced great progress in many fields. Deep learning has outperformed many other machine learning approaches by using deep, high-capacity models trained on large datasets. Especially in the field of computer vision, neural networks achieve state of the art results in most of the tasks. Many tasks in computer vision are the first where deep neural networks achieve state of the art results before being used in other fields, and in this field deeper and deeper architectures are being proposed earlier than in other fields.

With larger amount of parameters, the need for large datasets is growing, with currect datasets unable to cover the need for annotated data.

Data has proven to be limiting factor in many computer vision tasks. The main problem is that manual data annotation is exhausting, time-consuming and costly. That is even more significant for pixel-wise annotation which is crucial for tasks of semantic segmentation. Pixel-wise annotated datasets are orders of magnitude smaller than image classification datasets. This is sometimes called “curse of dataset annotation”[27], because more detailed semantic labeling leads to smaller size of dataset.

Many novel neural network architectures are being proposed every year because of increasing computing power. With growing capacity and number of parameters in these new models, there is need for bigger and bigger datasets for training.

Automatic data gathering and automatic data annotation could potentially solve these problems of lack of datasets in many computer vision and related tasks.

1.2. Gaming industry to the rescue

In last decades, gaming industry has grown hugely and expanded from small and specific community into public society and became mainsteam industy.

The gaming industry became big driving force in many fields, and indirectly influenced even machine learning.

The mainsteam model of gaming is on personal computers, where each player has his own gaming PC, along with console gaming. Thanks to ever-growing number of players, lots of money got into industry and the growing demand for better graphics in games led to big improvements in both software-computer graphics and hardware-graphics cards. With lots of money being invested by players in their PCs, GPU manufacturers were able to deliver more powerful GPUs every year and we can see exponential growth of GPU computational power[25].

Big companies in gaming industry have enough resources to develop the state of the art real-time computer graphics, which can we see in their products, AAA games with graphics very near to reality.

Recent papers[20, 24] show that we can use screenshots from PC games to obtain large automatically or semi-automatically annotated datasets, which improve learning, allow us to outperform same models trained only on real data and achieve state of the art results.

CHAPTER
TWO

RELATED WORK

[24] used GTA V to obtain screenshots and performed semi-automated pixel-wise semantic segmentation. Although the process was not fully automatic, the annotation speed per image was drastically increased, being 771 times faster than fine per-image annotation of Cityscapes [11] and 514 times faster than per-image annotation of CamVid[10]. They extracted 24 966 images from game GTA V, which is roughly two orders of magnitude larger than CanVid and three orders of magnitude larger than semantic annotations for KITTI dataset. They trained the prediction module of Yu and Koltun[28] and by using on $\frac{1}{3}$of the CamVid training set (which is) and all 24 966 GTA V screenshots, they outperformed same model trained on whole CamVid training dataset.

For images extraction, they use RenderDoc[21], stand-alone graphics debugger. It intercepts the communication between the game and the GPU and allows to gather screenshots. It's advantage is that it can be used for different games, allowing to gather datasets in various environments.

[20] use GTA V screenshots, depth and stencil buffer to produce car images and automatically calculate their bounding boxes.

On these generated data, they trained Faster R-CNN[23] only of screenshots from the GTA V game, using up to 200 000 screenshots, which is one order of magnitude bigger than Cityscapes dataset. Using only screenshots for training, they outperformed same architecture trained on Cityscapes, evaluating on KITTI dataset. They developed their own GTA V mod3.3 to hook into GPU calls and gather screenshots from here.

CHAPTER
THREE

TRANSFORMING GTA V INTO THE STATE OF THE ART SIMULATOR

In this thesis, Grand Theft Auto V (GTA V) game is used for creating synthetic, nearly photo-realistic dataset.

3.1. GTA V introduction

GTA V is action-adventure open-world video game developed by Rockstar North and published by Rockstar Games. The game was released on 17.9.2013 for Playstation 3 and Xbox 360[16] , in 18.11.2014 for PS4 and Xbox One and in 14.4.2015 it was released on PC, Windows[17].

The game is based on proprietary game engine, called RAGE (Rockstar Advanced Game Engine)[22], which is used as a base for most of Rockstar Games products.

Till the release on Microsoft, Windows, it has been in development for 5 years with approximate 1000-person team[15]. The world of GTA V was modelled on Los Angeles[19] and other areas of Southen California, with road networks respecting design od Los Angeles map.

As could be expected from AAA game like GTA V, motion capture was used to character's both body and facial movements.

There are several reasons why GTA V is better for dataset creationg than other games. To use a game for dataset creation, we have multiple requirements. The graphics of the game must be near photorealistic, since we try to to use it instead of photos for computer vision tasks. This disqualifies most of games, and leaves us only with AAA games produced by big companies and few other games with State of the Art graphics.

The other requirement is opssibility of good-enough way to interact with the game programatically. Usually we want to setup at least part of the environment before gathering data. This part heavily depends on community around the particular game.

Also the advantage of GTA V compared to some other games is abundance of models and various sceneries in its virtual world. It has complex transportation system of roads, highways, intersections, railroad crossing, tunnels, and pedestrians. It also has urban, suburban, and rural environments [14].

In gaming subculture, there are communities where people specialize in reverse-engineering of games and development of modifications to these games. These people are called modders or mod developers, and these unofficial modifications and extension of games are called mods. For few games, developers welcome this kind of activity and sometimes they even release tools to

ease the game modding. In most cases, the game developers simply don't care and in few cases, they actively fight against the reverse-engineering and modding.

The GTA V is second case, where Rockstar Games does not actively try to prevent the reverse-engineering, but they don't release any tools to ease it, either. This results in cyclic process of Rockstar Games releasing new version of game, including backward compatibility (BC) breaks, and community reverse engineering the new version and adjusting their mods to work with the new version.

The modding community around the GTA V is based mostly on community around GTA IV, which was previous big game produced by Rockstar Games. So many tools are just GTA IV based and only modified to work with GTA V. Luckily, the community is large and productive, so we have many mods and many function in GTA V reverse-engineered and thus prepared for programatic interactions.

3.1.1. Cars

There is big variety of cars models. Specifically, the are 259 car models, all of them are listed here [5]. These models cars of various shapes and sizes, from golf carts to trucks and trailers. This diversity is representative of real distribution of vehicles. It even allows us to simulate environments with various types of vehicles, which would be very difficult in real environment. GTA V provides us many information about cars, more on this will be covered in section 3.5.

3.1.2. Pedestrians

GTA V has pedestrians and provides some information about them, more on this in section 3.5. The game has pedestrians of both genders and various ethnicities. Pedestrians appear in various poses, like standing, walking, sitting, many animations etc. The main drawback of GTA V is that all pedestrians are about the same height[14].

3.2. Automotive Simulators

Currently, there are some opensorce simulation platforms for automotive industry which could be theoretically used for creating synthetic datasets. But compared to AAA games like GTA V, they have much less resources and much less customers to finance the development. In result, simulators have worse graphics than AAA games and NPC (non playable characters) don't have as sophisticated behaviour. In GTA V, drivers mostly follow traffic regulations, traffic lights and traffic lanes, which leafs to very realistic environment better than simulators can provide.

3.3. GTA V modding ecosystem

Although the modding community is quite big, as it is in lots of opensource communities, essential part of community depends on one person. Here, it is Alexader Blade. In his free time, he reverse-engineered big part of GTA V and developed ScriptHookV[9], library enabling to perform native calls into GTA V in C++ and develop GTA V mods in C++. Currently, more people in community participates in reverse-engineering and they share their knowledge in GTA forum thread[8].

List of all reverse-engineered native functions is kept in following list [7]. Assumingly, GTA V contains ~5200 natives. There is no original native name list of functions in GTA V, name hashes are used instead. During reverse-engineering ande game decompilation, ~2600 native names were discovered using bruteforce and manual checking afterwards. For these functions, number of parameters and returns of these calls are also known. In the native functions list, for big part of functions we know their name, signature and how do they affect the game. The rest remains to be discovered yet.

When new version of game is released, in few days to weeks, new version of ScriptHookV is released, fixing BC breaks.

Other heavily used mod in community is ScriptHookDotNet2, which is built atop of ScriptHookV and creates bridge between C# language and ScriptHookV, effectively allowing to write GTA V mods in C#. It is availabe as open-source [12]. Along with creating bridge between C# and GTA V, it wraps most used native calls into classes, leveraging object-oriented paradigm for mod development using getters and setters as proxies for native calls.

Next notable mod is NativeUI[18]. It renders windows atop of GTA V GUI and allows us to define custom control panels for manipulating custom functionality in other mods.

Unlike most of other mods, these three mods act more as a framework for mod development.

Since GTA V is a game, it requires human interaction. For simulator-like behavior we would want the car to drive autonomously to crawl data without human interaction. This can be done using VAutodrive[13]. This allows us to use NPC automatic behaviour patterns for main player, letting the player randomly wander the world, even in car, without need of human assistance during crawling. Unfortunately, this package is not open-source.

Generally, the community is not united in their view on opensource. Some mods are available open-source on github. Other mods are being distributed only as compiled binaries[1]. Lots of modders develop mostly by trial and error, and no comprehensive documentation for mod development is available, unfortunately. There are some tutorials [26], but they are far from complete and provide only basic knowledge, leavin reader without deeper understanding of underlying principles.

Modders mostly meet online on few GTA forums, where they exchange knowledge [2, 3]. Github or StackOverflow, which are biggest information sources for usual software development, are not used much in GTA modding community. Due to this fact, these forums, along with source code of open-source mods comprise knowledge-base of mod development.

3.4. Simulation environment and development stack

In this thesis, I use mod based on [20] but enhanced to gain more control of the game and to obtain more information from the game.

In later text, I'll refer to some GTA V native functions or data structures which are output of GTA V native functions. To be consistent and to help understanding, I will use function names from native function list [7].

The basic architecture of C# mods come from the ScriptHookDotNet2, where each mod script extends the GTA.Script class. For each child of this class, we can set integer Inteval, Tick and KeyUp callbacks. Interval property determines how big interval in milliseconds is between consecutive Tick calls. Tick callback is being called periodically, so here we can set tasks which we want to perform periodically, e.g. screenshot gathering. The KeyUp callback is for interacting with user and reading the user's keyboard input. For data gathering mods, this is mostly used for

debugging purposes, script disabling or restarting.

Little, but useful note for debugging and developing scripts based on ScriptHookDotNet2: when changing mod compiled binaries, we don't have to restart the game for newer version of mod to be active. Pressing Insert causes all C# binaries to reload, causing new version of source code to load into game, which dramatically descreases time of feedback loop during development. This does not work for C++ mods and compiled .asi files.

3.5. GTA V native API and data obtaining

The data obtained from GTA V can be divided into multiple categories.

- Image data
- Rendering pipeline matrices
- GTA V internal data
  - entities
  - camera
  - player
  - other data via API

3.5.1. Image data

There are 3 image data. RGB image, depth image and stencil image.

RGB image is usual camera image. Depth image is content of GPU's depth buffer, in NDC. More detailed description of depth values is in subsection 3.6.3. The last is pixel-wise stencil buffer. The stencil semantics is explained in the next paragraph.

Stencil data

Stencil buffer contains auxiliary data per pixel. It is 8bit unsined integer, where 1.-4. bits (counting from LSB) contain object type ID, and 5.-8. bits contain certain flags.

That means there are 15 object types and 4 flags.

For some object type IDs, I reverse engineered its semantics based on corresponsing RGB image.

Object type ID binary	Object type ID decimal	Semantics
0000	0	backround (buildings, roads, hills...)
0001	1	pederstrian
0010	2	vehicle
0011	3	tree, grass
0111	7	sky

The background, pedestrian and vehicle IDs are most important for vehicles detection and semantic segmentation, and other object types are not so valuable for these tasks, which is why I didn't investigate further in semantics and they remain to be reverse-engineered.

For stencil flags, semantics was discovered for half of them.

position of bit	position in whole stencil value	Semantics
5	xxx1xxxx	artificial light source
8	1xxxxxxx	player's character

Sample stencil image can be seen in 3.2.

Extracting images from GPU's internal buffers

To understand how the image gathering works, we need to dive deeper into Microsoft Windows graphics.

In Microsoft Windows, the main graphics engine is DirectX (roughly Windows equivalent of OpenGL). One part of DirectX is Direct3D, which is used to render 3D graphics with hardware acceleration, and most importantly, it provides graphics API. The whole process of obtaining image data is done by mod provided as part of the paper [20]. The mod has two parts, called native and managed plugin.

Image data is being obtained by native plugin by hooking into Direct3D 11's present callback. That means the native call is replaced with custom code, which is being executed and then returns to the native call. In that custom code, content of GPU's buffers is copied. Specifically, the depth and stencil data are captured by hooking into Direct3D's ID3D11ImmediateContext:: ClearDepthStencilView and saving the buffers before each call. Because of optimizations applied by the graphics card drivers, the function needs to be rehooked into the clear function each frame. When saving each sample, the managed plugin requests all current buffers from the native plugin and the buffers are downloaded from the GPU and copied into managed memory.[20].

3.5.2. Rendering pipeline data

Direct3D allows us to get rendering pipelines through the D3D11_MAP_READ call. This way, the world, worldview, and worldviewprojection matrices are obtained. Let's denote them as world matrix = W, worldview by = VW, and worldviewprojection = PVW. We need to get individual matrices, which we obtain by multiplication by matrices inversion:

$$V = VW \cdot W^{-1}$$
$$P = PVW \cdot (VW)^{-1} = PVW \cdot W^{-1} \cdot V^{-1}$$

View and projection matrices are important on their own, without the world matrix. Their semantics and importance is more described in section 3.6. These matrices can be simply obtained by inversion as stated above. But this approach is not numerically stable, and sometimes causes resulting matrix to be incorrect and not usable for further usage. Another cavear of this approach is that during data gathering in higher speeds, the native call becomes laggy and resulting matrices are highly imprecise. Although we can obtain these matrices via the native call, they can be reconstructed with higher precision, as described in 3.6.

3.5.3. GTA V internal data

These data are probably most valuable compared to data gathering by other methods. In this section, I describe which data about various game objects can be obtained and how to obtain them.

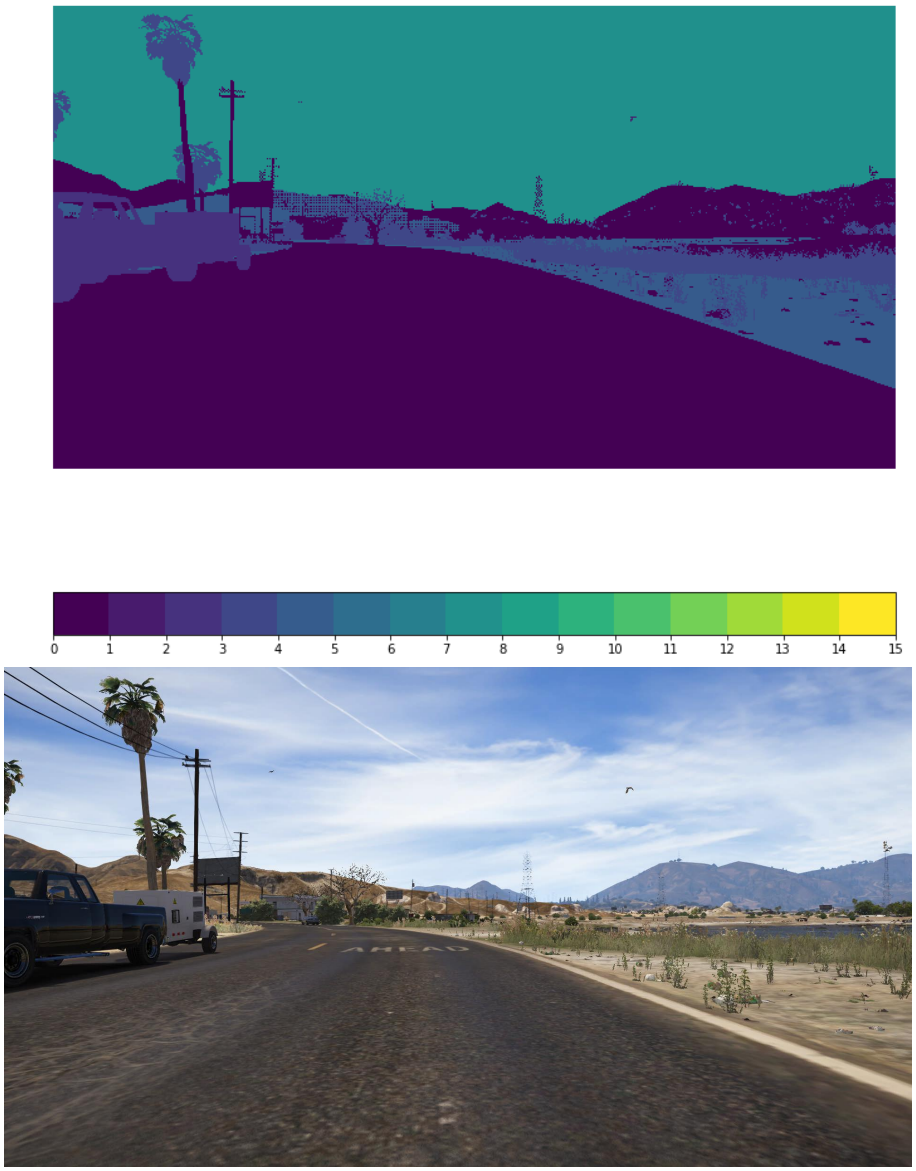


Figure 3.1.: Sample stencil object types image and corresponsing RGB image

All data in this section are obtained though native calls into GTA V, which are listed here [7]. As mentioned above, they can be called from C++ and C#. For convenience, and because most od my mod development is done in C#, I will also describe the C# wrappers. The ScriptHookDotNet2[12] wrapper heavily uses object properties. The C# API is divided into multiple classes, listed here. Each class has properties, whose getters and setters are implemented as calling the native functions. This feature is nice tooling and leads to more readable and mantainable code. I will describe parts of API which are most useful for synthetic datasets creation.

Camera

The camera is probably one of the most crucial parts of the API. There is Gameplay Camera, which is the default camera used during usual gaming. This camera can be manipulated, but its usage is limited. Other aproach is usage of scripted cameras, which can be fully controlled programatically. We can create multiple scripted cameras and switch between them, but the community discovered there is hard limit of 26 cameras at time[4]. Camera can be created by calling

```
Camera camera = World.CreateCamera(new Vec-
tor3(x, y, z), new Vector3(x, y, z), float fov);
```

which returns handle to the new camera. The position is in world coordinates in meters. The rotation is in degrees in Euler angles. The fov argument is vertical field of view in degrees. The detault value is 50. One can of course call the underlying native functions directly, but this wrapper helps with managing the handles.

All scripted cameras can be desctroyed by calling the

```
World.DestroyAllCameras();
```

Switching to the scripted camera can be done by calling

```
camera.IsActive = true;
```

, and switching from this camera to some other by

```
camera.IsActive = false;
```

Right after creating the camera, when we don't to switch to this camera immediately, we need to deactivate it by these two lines of code

```
camera.IsActive = false;
World.RenderingCamera = null;
```

this code ensures that camera is created properly. We don't know precisely, why is this needed, but this is the price for using the closed and reverse-engineered codebase. By calling the

```
camera.IsActive = true;
World.RenderingCamera = camera;
```

scripted camera becomes active and view is switched to this camera. All properties of camera can be set simply by calling setters

```
camera.position = new Vector3(x, y, z);
camera.rotation = new Vector3(y, x, z);
camera.nearClip = distance;
camera.farClip = distance;
camera.FieldOfView = fov;
```

and read by calling getters

```
Vector3 position = camera.position;
Vector3 rotation = camera.rotation;
float distance = camera.nearClip;
float distance = camera.farClip;
float fov = camera.FieldOfView;
```

The near clip and far clip is again in meters. So we can set all needed parameters simply by calling getters and setters. So if we want to use camera, we simply set parameters and then activate it. This creates static camera. There is only one caveat, which is probably bug in the API: when setting the camera rotation Euler angles, the second angle is in first position and first angle is in the second position in the setting vector. Otherwise the camera won't be rotated correctly.

Sometimes we want the camera to be moving, e.g. when gathering data by driving car. Camera can be attached to any entity by calling

```
camera.AttachTo(entity, new Vector3(x, y, z));
```

the second parameter is relative offset to middle of the attached entity. The offset is in model coordinate system which means its position moves and rotates with attached entity. During dataset gathering, I used this code

```
camera.AttachTo(Game.Player.Character.CurrentVehicle, new Vec-
tor3(0f, 0f, 0.4f));
```

to attach camera to the front part of player's current vehicle.

3.6. Reverse-engineering the RAGE rendering pipeline

As mentioned above3.1, GTA V uses proprietary game engine, Rockstar Advanced Game Engine (RAGE). The basic premise of rendering pipeline is same as in well known graphics engines like OpenGL. The pipeline is shown in figure 3.2. Following section will be discussing mostly computer graphics related problems. Due to some terminology inconsistency between computer graphics and computer vision, all terms used here will be compuer graphics related. Probably most confusion here could be caused by projection matrix. In computer vision, projection matrix is projection from 3D to 2D, the matrix reduces dimension. In computer graphics, all coordinates are kept in 4D, in homogeneous coordinates as long as possible. Here the projection matrix

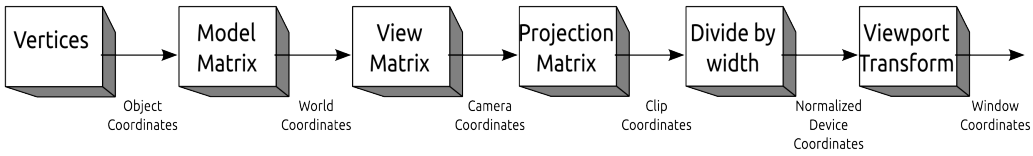


Figure 3.2.: Rendering pipeline

prepresents projection from frustum seen by eye into cuboid space of Normalized Device Coordinates.

In is part, I will describe some transformations between individual RAGE coordinate systems. Some points here will have part of name in lower index. The name of coordinate system will be denoted in upper index. In RAGE there are 6 coordinate systems.

Name	Abbreviation	Example point x
Object Coordinates	O	x^O
World Coordinates	W	x^W
Camera Coordinates	C	x^C
Clip Coordinates	L	x^L
Normalized Device Coordinates	NDC	x^{NDC}
Windows Coordinates	P	x^P

Most of points we handle in GTA already are in world coordinates.

But some points, like GAMEPLAY::GET_MODEL_DIMENSIONS

$= \begin{pmatrix} x^O_{max} & y^O_{max} & z^O_{max} \end{pmatrix} \begin{pmatrix} x^O_{min} & y^O_{min} & z^O_{min} \end{pmatrix}$ output, are in object coordinates. Transitions between adjacent coordinate systems will be demonstrated on model dimensions because it is on the few vectors which are obtained in Object Coordinates and there is need to project them into Window Coordinates.

3.6.1. Object to World Coordinates

To get world coordinates of model dimensions, we use traditional rigid body transformation based on ENTITY::GET_ENTITY_ROTATION= $\begin{pmatrix} \alpha & \beta & \gamma \end{pmatrix}$ Euler angles,

and ENTITY::GET_ENTITY_COORDS= $\begin{pmatrix} x^W & y^W & z^W \end{pmatrix}$.

Because all coordinates will be homogeneous coordinates, the above-mentioned model dimensions vectors will be transformed to following form $\begin{pmatrix} x^O_{max} & y^O_{max} & z^O_{max} & 1 \end{pmatrix} \begin{pmatrix} x^O_{min} & y^O_{min} & z^O_{min} & 1 \end{pmatrix}$.

The transition is represented by model matrix

$$M = \begin{bmatrix} 1 & 0 & 0 & x^W \\ 0 & 1 & 0 & y^W \\ 0 & 0 & 1 & z^W \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\beta)\cos(\gamma) & -\cos(\beta)\sin(\gamma) & \sin(\beta) & x^W \\ \sin(\alpha)\sin(\beta)\cos(\gamma) + \cos(\alpha)\sin(\gamma) & \cos(\alpha)\cos(\gamma) - \sin(\alpha)\sin(\beta)\sin(\gamma) & -\sin(\alpha)\cos(\beta) & y^W \\ \sin(\alpha)\sin(\gamma) - \cos(\alpha)\sin(\beta)\cos(\gamma) & \cos(\alpha)\sin(\beta)\sin(\gamma) + \sin(\alpha)\cos(\gamma) & \cos(\alpha)\cos(\beta) & z^W \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and whole transformation is, as expected

$$M \begin{bmatrix} x^O_{max} & x^O_{min} \\ y^O_{max} & y^O_{min} \\ z^O_{max} & z^O_{min} \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} x^W_{max} & x^W_{min} \\ y^W_{max} & y^W_{min} \\ z^W_{max} & z^W_{min} \\ w^W_{max} & w^W_{min} \end{bmatrix}$$

3.6.2. World to Camera Coordinates

The transformation from world coordinates is principally the same, but counterintuitive in definition of used rotation matrices. It also is rigid body transformation, but rotation is defined differently than we are usually used to in computer graphics. The rotation matrices were reverse engineered as part of this thesis from camera position, rotation and resulting view matrix, this coordinate system is nowhere else documented. The camera position is CAM::GET_CAM_COORD= $\begin{pmatrix} x^W & y^W & z^W \end{pmatrix}$ and the camera rotation is CAM::GET_CAM_ROT= $\begin{pmatrix} \alpha & \beta & \gamma \end{pmatrix}$.

The transformation is represented by view matrix

$$V = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\gamma) & \sin(\gamma) & 0 & 0 \\ \sin(\gamma) & -\cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & x^W \\ 0 & 1 & 0 & y^W \\ 0 & 0 & 1 & z^W \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

to fit the matrix into page, let us propose following substitutions

$$\cos(\alpha) = c_\alpha, sin(\alpha) = s_\alpha$$

$$\cos(\beta) = c_\beta, sin(\beta) = s_\beta$$

$$\cos(\gamma) = c_\gamma, sin(\gamma) = s_\gamma$$

$$V = \begin{bmatrix} c_\beta c_\gamma & c_\beta s_\gamma & -s_\beta & 0 \\ c_\alpha s_\beta c_\gamma + s_\alpha s_\gamma & c_\alpha s_\beta s_\gamma - s_\alpha c_\gamma & c_\alpha c_\beta & 0 \\ c_\alpha s_\gamma - s_\alpha s_\beta c_\gamma & -s_\alpha s_\beta s_\gamma - c_\alpha c_\gamma & -s_\alpha c_\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & x^W \\ 0 & 1 & 0 & y^W \\ 0 & 0 & 1 & z^W \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c_\beta c_\gamma & c_\beta s_\gamma & -s_\beta & x^W c_\beta c_\gamma + y^W c_\beta s_\gamma - z^W s_\beta \\ c_\alpha s_\beta c_\gamma + s_\alpha s_\gamma & c_\alpha s_\beta s_\gamma - s_\alpha c_\gamma & c_\alpha c_\beta & x^W \left(c_\alpha s_\beta c_\gamma + s_\alpha s_\gamma\right) + y^W \left(c_\alpha s_\beta s_\gamma - s_\alpha c_\gamma\right) + z^W c_\alpha c_\beta \\ c_\alpha s_\gamma - s_\alpha s_\beta c_\gamma & -s_\alpha s_\beta s_\gamma - c_\alpha c_\gamma & -s_\alpha c_\beta & x^W \left(c_\alpha s_\gamma - s_\alpha s_\beta c_\gamma\right) + y^W \left(-s_\alpha s_\beta s_\gamma - c_\alpha c_\gamma\right) - z^W s_\alpha c_\beta \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and whole transformation is, as expected

$$V \begin{bmatrix} x^W_{max} & x^W_{min} \\ y^W_{max} & y^W_{min} \\ z^W_{max} & z^W_{min} \\ w^W_{max} & w^W_{min} \end{bmatrix} = \begin{bmatrix} x^C_{max} & x^C_{min} \\ y^C_{max} & y^C_{min} \\ z^C_{max} & z^C_{min} \\ w^C_{max} & w^C_{min} \end{bmatrix}$$

From definition of rotation axes in the rotation matrices, following observation can be made. z^C represents distance from camera in direction of camera heading, and $x^C$and y^C represent horizontal and vertical position of point relative to camera, respectively. But the view frustum of camera is in opposite direction than z^Caxis, which means the camera "is looking" into negative z^C coordinates.

3.6.3. Camera to NDC

This is the first transformation which is not rigid-body transformation. Because camera sees only frustum, this transformation prepresents transition from frustum to cuboid in Normalized Device Coordinates. The frustum being projected is specified by near clip, far clip, field of view and screen resolution width and height. Usually, none of these parameters are changing during the game, so the projection matrix is usually the same for multiple scenes during data gathering session. Although all of these parameters can be changed programatically if needed.

The near clip and far clip of camera can be obtained by CAM::GET_CAM_NEAR_CLIP= n_c and CAM::GET_CAM_FAR_CLIP= f_c. Width and height of screen resolution are obtained by GRAPHICS::_GET_ACTIVE_SCREEN_RESOLUTION= $\begin{pmatrix} W & H \end{pmatrix}$ and field of view of camera by CAM::GET_CAM_FOV= φ_{VD} in degrees. φ_{VD} in radians will be denoted as φ_{VR}.

The near clip and far clip define planes between which the content is being rendered. Nothing before the near clip and behind the far clip is rendered.

The field of view φ_{VD} is only vertical. Horizontal field of view can be calculated from W and H ratio, but currently we don't need it.

There is important observation, the far clip f_c does not figure in the projection matrix at all. In the projection matrix, only n_c is used. Far clip used in projection matrix is non-changing value which can not be obtained through Camera native function. By reverse-engineering I calculated the value of this new far clip to be 10003.815, details of this calculation are covered in experiments4.0.1.

The transformation is represented by projection matrix

$$P = \begin{bmatrix} \frac{H}{W \cdot \tan\left(\frac{\varphi_{VR}}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\varphi_{VR}}{2}\right)} & 0 & 0 \\ 0 & 0 & \frac{-10003.815}{n_c - 10003.815} & \frac{-10003.815 \cdot n_c}{n_c - 10003.815} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

So the projection to Clip Coordinates is

$$P \begin{bmatrix} x^C_{max} & x^C_{min} \\ y^C_{max} & y^C_{min} \\ z^C_{max} & z^C_{min} \\ w^C_{max} & w^C_{min} \end{bmatrix} = \begin{bmatrix} x^L_{max} & x^L_{min} \\ y^L_{max} & y^L_{min} \\ z^L_{max} & z^L_{min} \\ w^L_{max} & w^L_{min} \end{bmatrix}$$

The transition between Clip Coordinates and NDC is only division by width, so it is

$$\begin{bmatrix} x^L_{max} & x^L_{min} \\ y^L_{max} & y^L_{min} \\ z^L_{max} & z^L_{min} \\ w^L_{max} & w^L_{min} \end{bmatrix} \circ \begin{bmatrix} \frac{1}{w^L_{max}} & \frac{1}{w^L_{min}} \\ \frac{1}{w^L_{max}} & \frac{1}{w^L_{min}} \\ \frac{1}{w^L_{max}} & \frac{1}{w^L_{min}} \\ \frac{1}{w^L_{max}} & \frac{1}{w^L_{min}} \end{bmatrix} = \begin{bmatrix} x^{NDC}_{max} & x^{NDC}_{min} \\ y^{NDC}_{max} & y^{NDC}_{min} \\ z^{NDC}_{max} & z^{NDC}_{min} \\ 1 & 1 \end{bmatrix}$$

where $\circ$ is Hadamard product, also known as entry-wise product or element-wise matrix multiplication.

Let us have vector $\boldsymbol{x} = \begin{bmatrix} x & y & z & w \end{bmatrix}^T$ in both coordinate systems, $\boldsymbol{x}^L = \begin{bmatrix} x^L & y^L & z^L & w^L \end{bmatrix}^T$, $\boldsymbol{x}^{NDC} = \begin{bmatrix} x^{NDC} & y^{NDC} & z^{NDC} & w^{NDC} \end{bmatrix}^T$. Then, the relation between Clip Coordinates and

NDC can also be expressed by following relationship

$$\boldsymbol{x}^L = \begin{bmatrix} x^L \\ y^L \\ z^L \\ w^L \end{bmatrix} = \begin{bmatrix} x^{NDC} w^L \\ y^{NDC} w^L \\ z^{NDC} w^L \\ w^L \end{bmatrix} = w^L \begin{bmatrix} x^{NDC} \\ y^{NDC} \\ z^{NDC} \\ 1 \end{bmatrix} = w^L \boldsymbol{x}^{NDC}$$

The view frustum was now transformed into NDC cuboid. The NDC cuboid has dimensions $x \in [-1, 1], y \in [-1, 1], z \in [0, 1]$. The x and y coordinates are intuitive, but the z-axis is reverted, so near clip is being mapped to 1 and far clip is being mapped to 0. The NDC is important because it is coordinate space in which GPU operates and depth is gathered from GPU in NDC. The value of $z^{NDC} = 0$ usually belongs to sky.

3.6.4. NDC to Window Coordinates

This is the last transformation of the rendering pipeline and only in this transformation the dimension reduction happen. So far points have been kept in homogeneous coordinates, but window coordinates are only 2D, expressing x and y coordinates of pixel where point will be rendered. Here, we need only GRAPHICS::_GET_ACTIVE_SCREEN_RESOLUTION= $\begin{pmatrix} W & H \end{pmatrix}$ because this transformation depends only on screen with and height.

The transformation matrix is

$$T = \begin{bmatrix} \frac{W}{2} & 0 & 0 & \frac{W}{2} \\ 0 & \frac{-H}{2} & 0 & \frac{H}{2} \end{bmatrix}$$

so the NDC to screen transformation is

$$T \begin{bmatrix} x^{NDC}_{max} & x^{NDC}_{min} \\ y^{NDC}_{max} & y^{NDC}_{min} \\ z^{NDC}_{max} & z^{NDC}_{min} \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} x^P_{max} & x^P_{min} \\ y^P_{max} & y^P_{min} \end{bmatrix}$$

Due to the division by width, the pipeline unfortunately can not be expressed as matrix multiplication by matrix constant for all points in one scene.

CHAPTER
FOUR

EXPERIMENTS

4.0.1. Reverse engineering the true Far Clip

For reverse engineering the far clip, I gathered 33293 screenshots with parameters for projection matrix reconstruction and projection matrices. Because during whole data gathering none of the parameters used to reconstruct projection matrix, was changed, the projection matrix should be same for all records. As mentioned in 3.6.3 parameters for reconstructing the Projection matrix are near clip, far clip, screen width, screen height and field of view.

The screenshot contain both RGB images and depth buffer from GPU.

The projection matrix transforms frustum into cuboid. For approximate estimation of projection matrix parameters, I used DirectX projection matrix[6] as a starting point for analysis, because GTA V requires DirectX, so I assumed it is underlying framework of RAGE.

The DirectX projection matrix is

$$P = \begin{bmatrix} \frac{2n}{r-l} & 0 & -\frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & -\frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{f}{f-n} & -\frac{fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

where n is near clip, f is far clip, l and r determine distance between left and right planes of the frustum and t and b determine distance between top and bottom planes.

The view frustum is symmetric, so $r = -l$ and $t = -b$ [6]. In that case, the projection matrix is simplified to form

$$P = \begin{bmatrix} \frac{2n}{r+r} & 0 & -\frac{r-r}{r+r} & 0 \\ 0 & \frac{2n}{t+t} & -\frac{t-t}{t+t} & 0 \\ 0 & 0 & \frac{f}{f-n} & -\frac{fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} \frac{n}{r} & 0 & 0 & 0 \\ 0 & \frac{n}{t} & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & -\frac{fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

The DirectX maps near clip to 0 and far clip to 1, but from data, where obviously4.2 nearer pixels had higher value in depth buffer than pixel more far from camera, I concluded that near and far clip are being mapped to 1 and 0, respectively. The far clip being mapped to 0 can also be deduced by pixels for sky having 0 value.

The example in 4.2 does not have actual depth buffer values, but instead, it is rescled visualization. Since the depth buffer pixels are in range $[0, 1]$ and png images take unsinged 8bit integer,

Figure 4.1.: Example of RGB image

Figure 4.2.: Example of depth buffer

this image is mapped linearly from $[0, 1]$ to $[0, 255]$. Since even the nearest pixels were distant from near clip and real range of pixels in this image was $[0, 19]$, I rescaled it 10 times to range $[0, 190]$, so the depth is visible.

At first, I assumed the camera near clip and far clip obtained by native calls 3.6.3 and the projection matrix is same as in DirectX.

The near clip and far clip calculation can be demonstrated on image4.1. By calling CAM::GET_CAM_NEAR_CLI n_c and CAM::GET_CAM_FAR_CLIP= f_c I obtained values $n_c = 0.15$ and $f_c = 800$. The obtained projection matrix is.

CHAPTER
FIVE

FUTURE WORK

In GTA V reverse-engineering, many stencil values semantics remain to be discovered.

BIBLIOGRAPHY

[1] Gta 5 mods, . URL https://www.gta5-mods.com/.

[2] Gta5 mods forum, . URL https://forums.gta5-mods.com/category/5/general-modding-discussion.

[3] Gta forums, . URL http://gtaforums.com/forum/370-gta-v/.

[4] . URL http://gtaforums.com/topic/793247-cameras/.

[5] . URL http://www.ign.com/wikis/gta-5/Vehicles.

[6] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008. ISBN 987-1-56881-424-7.

[7] Alexander Blade. Native db. URL http://www.dev-c.com/nativedb/.

[8] Alexander Blade. Native db script/native documentation and research, June 2014. URL http://gtaforums.com/topic/717612-v-scriptnative-documentation-and-research/.

[9] Alexander Blade. Script hook v, April 2015. URL http://gtaforums.com/topic/788343-script-hook-v/.

[10] Gabriel J. Brostow, Julien Fauqueur, and Roberto Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recogn. Lett.*, 30(2):88–97, January 2009. ISSN 0167-8655. doi: 10.1016/j.patrec.2008.04.005. URL http://dx.doi.org/10.1016/j.patrec.2008.04.005.

[11] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *CoRR*, abs/1604.01685, 2016. URL http://arxiv.org/abs/1604.01685.

[12] Crosire. Community script hook v .net, April 2015. URL https://github.com/crosire/scripthookvdotnet.

[13] Cyron43. Vautodrive, July 2015. URL https://www.gta5-mods.com/scripts/vautodrive.

[14] Artur Filipowicz. Driving school ii video games for autonomous driving. resreport, Princeton University, May 2016. URL http://orfe.princeton.edu/~alaink/SmartDrivingCars/Visitors/FilipowiczVideoGamesforAutonomousDriving.pdf.

[15] MICHAEL FRENCH. Inside rockstar north - part 2: The studio, October 2013. URL https://www.mcvuk.com/development/inside-rockstar-north-part-2-the-studio.

[16] Rockstar Games. Grand theft auto v is coming 9.17.2013, January 2013. URL https://www.rockstargames.com/newswire/article/48591/grand-theft-auto-v-is-coming-9172013.html.

[17] Rockstar Games. Grand theft auto v release dates and exclusive content details for playstation 4, xbox one and pc, September 2014. URL https://www.rockstargames.com/newswire/article/52308/grand-theft-auto-v-release-dates-and-exclusive-content.

[18] Guad. Native ui, July 2015. URL https://github.com/Guad/NativeUI.

[19] Matt Hill. Grand theft auto v: meet dan houser, architect of a gaming phenomenon, September 2013. URL https://www.theguardian.com/technology/2013/sep/07/grand-theft-auto-dan-houser.

[20] M. Johnson-Roberson, Charles Barto, Rounak Mehta, Sharath Nittur Sridhar, Karl Rosaen, and Ram Vasudevan. Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks? In *IEEE International Conference on Robotics and Automation*, pages 1–8, 2017.

[21] Baldur Karlsson. Renderdoc. URL https://renderdoc.org/.

[22] Thomas Morgan. Face-off: Grand theft auto 5, September 2013. URL https://www.eurogamer.net/articles/digitalfoundry-grand-theft-auto-5-face-off.

[23] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. URL http://arxiv.org/abs/1506.01497.

[24] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *European Conference on Computer Vision (ECCV)*, volume 9906 of *LNCS*, pages 102–118. Springer International Publishing, 2016.

[25] Balázs Tukora and Tibor Szalay. High performance computing on graphics processing units. *Pollack Periodica*, 3:27–34, 08 2008. URL https://www.researchgate.net/publication/242065578_High_performance_computing_on_graphics_processing_units.

[26] V4D3R. Gta-mod-dev-tutorial, August 2016. URL https://forums.gta5-mods.com/topic/1451/

tutorial-grand-theft-auto-v-modding-a-few-things-you-should-know/2.

[27] Jun Xie, Martin Kiefel, Ming-Ting Sun, and Andreas Geiger. Semantic instance annotation of street scenes by 3d to 2d label transfer. *CoRR*, abs/1511.03240, 2015. URL http://arxiv.org/abs/1511.03240.

[28] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122, 2015. URL http://arxiv.org/abs/1511.07122.

APPENDIX
A

CONTENTS OF THE ENCLOSED CD

appendix content