Development of a virtual road traffic generation module

Alexey Zalata[1], *Aleksandr* Leksashov[1], *Marina* Bolsunovskaya[1], *Svetlana* Shirokova[1,*], and *Oleg* Tarasov[1]

[1]Peter the Great St. Petersburg Polytechnic University, Saint Petersburg, Russia

Abstract. In this paper a module for generating virtual road traffic was developed. There were carried out studies of the main software products for generating road traffic and current development tools for creating the necessary module. Based on these researches the development environment, programming language, and third - party modules were selected to simplify and speed up development. After the analysis and selection of the development environment, a customizable vehicle movement system was developed and implemented. The system includes a driving algorithm, the ability to customize the configuration and type of vehicle, as well as several road signs. This module has been integrated with the virtual urban environment generation module to test connectivity with other projects. Functional and load testing was performed. It helped to identify some shortcomings. These shortcomings were eliminated.

1 Introduction

Currently, the number of vehicles on the roads is growing quite quickly. Every year, the problem of traffic management becomes more complex. To analyze different types of traffic or other tasks related to the movement of vehicles, real or generated data is required. Real data can be obtained from surveillance cameras, and it takes time to get access to recordings from video cameras that monitor traffic on the roads. Also, despite the fact that the cameras are located in a large number of places, despite this, they may not be on the required road section. This does not give full access to the desired data. Generated traffic flows can help solve this problem.

Computer modeling is becoming a common tool for analyzing and creating various complex systems. The modern market requires a system for generating virtual road traffic, with the help of which it will be possible not only to analyze traffic, but also to use this module in various projects to simulate road traffic [1].

This paper is devoted to the analysis of the subject area and a review of similar technologies for generating road traffic. The directions of research are defined and the task of developing a software module is set [2,3].

To solve this problem, it is necessary to review existing solutions and consider the functionality of tools designed for creating traffic flows.

* Corresponding author: swchirokov@mail.ru

In addition, it is necessary to test in practice various tools that make it easier to develop a module for generating virtual road traffic.

2 Materials and Methods

Nowadays, 3D object visualization is used for a large number of projects, such as creating commercials, various interior layouts, movies, and computer games. Visualization is indispensable where the use of a real object is impossible or unprofitable, and then there is a need to create a virtual copy of this object.

3D modeling is the process of creating a three-dimensional object. The object itself can be either real or fictional. The possibilities of this technology are quite large – you can change the shape and size of objects, experiment with different colors, apply textures and animate the created objects. The main advantage of three-dimensional visualization is that it allows you to create layouts of impressive images, the creation of which in real life would require a large investment of effort or would be impossible at all.

Because of the above qualities, 3D modeling is often used to create a layout of new types of products, or when the available two-dimensional image of the object is insufficient to achieve the set goals.

Emulating a 3D transport model can be necessary in many different areas, such as creating interference in driving training programs, using such vehicles in movies, or when training and testing neural networks that can later be used to recognize cars on the roads.

Transport emulation also plays an important role in creating environments in various applications and beyond. If you need to generate one or two units of transport when generating the environment, then this task is not particularly difficult, but if you need to generate a large number of vehicles, then placing it manually is not very convenient and can take quite a long time. To simplify this task, you can use algorithms that will generate different types of transport, for example, procedural generation [4].

The main examples of using procedural generation (hereinafter referred to as PCG) are: the generation of the surrounding world, without human participation, which receives a new world at each launch; a system that creates new types of objects depending on the player's actions; the creation of a complete and balanced board game with its own rules; a system that fills the world with different types of plants (trees, bushes, grass, etc.); a tool that allows you to create maps of the area and has access to various parameters that affect the fullness of the map with different objects or allow you to make the map more balanced [5].

The main purpose of using PCG can be to create any objects without human intervention, which can be both less expensive and significantly speed up the process of developing a software product. The content created with the help of procedural generation must meet various conditions, and thus help in solving the tasks set. The following properties are usually considered as general requirements:

- Speed: depending on the task, the requirements vary from milliseconds to months, but in general, the content must be created in time to meet the needs of the software product;
- Reliability: Some generators create a large number of elements, while others can guarantee that the specified criteria are met;
- Controllability: the ability to control the generated content based on the situation and give the game designer the appropriate freedom;
- Diversity: create content that is as different as possible on different launches;
- Creativity and verisimilitude: generating content that looks like it was created by a person, not a generator.

One of the main areas of research and development of modern cybernetics is the field of computer vision, which consists in machine recognition of various images. In corporate and everyday life, technologies are beginning to be introduced that gradually remove the line

between real and virtual space, which requires a new qualitative level of universally implemented image recognition technologies, whose scope of application has recently grown quite strongly: previously considered quite complex, recognition tasks are now quite easily solved by budget mobile devices. Due to the fact that the pace of development of information society technologies is increasing, the development of the Internet of Things and various artificial intelligence systems determine this area a special place in modern scientific knowledge.

The effectiveness and correctness of the recognition of various images depends not only on the training algorithms of the neural network and the quality of the network, but also, of course, on the quality of the material on which the neural network is trained. Sometimes, even with seemingly successful training, the network does not always learn exactly what the developers wanted it to. Quite often, you can find cases when a neural network trained on the same type of data or data taken from the same angle cannot recognize the desired image. As a result, the larger the amount of data and the more detailed its quality, the better the recognition of the neural network will be.

Testing the neural network is also very important. It helps to identify inaccuracies and shortcomings in training, which can later lead to various errors and failures in the work of a ready-made program that uses this neural network. When testing a neural network, it is necessary to take into account not only the usual pattern recognition, but also to conduct testing with various interferences that prevent direct object recognition. Thus, the created virtual traffic generation module can be used for testing neural networks or for traffic generation tasks in other software products. There are solutions on the market for transport generation tasks:

1. PTV Vissim (microscopic multimodal software package for traffic flow simulation). This software product is used to simulate a model of traffic flows, the data from which is later used to develop various solutions for the organization of pedestrian and road traffic. [6] Each object here (person, train, car) of reality is modeled individually.
2. Simulation of Urban Mobility (SUMO-Simulation of urban mobility). This is a package of portable and continuous open source traffic simulation tools designed to work with large road networks. [7]

It is also possible to pair SUMO with Unity to get a 3D simulation [8].

3. AimSun Live is a simulation-based traffic forecasting solution developed by AimSun. Traffic management centers use AimSun Live to make decisions about managing the road network in real time. It is used to dynamically predict future traffic conditions based on the current state of the network, and it is also used to evaluate incident response or create traffic management strategies [4].

AimSun Live continuously processes operational field data, simulating the movement of vehicles within a road network of any size, from a single road corridor to an entire major global city. By combining these real-time traffic data streams and high-speed simulations with emulation of congestion mitigation strategies, AimSun Live can accurately predict future network flow patterns that will result from a specific traffic management or information delivery strategy.

AimSun Live analyzes real-time inputs from a variety of information sources, such as field dispatchers, detectors, incident reports, and live data feeds from key intersections.

This step involves dynamic simulation of one or more scenarios in real time. Each of the scenarios is modeled on a dedicated computer. The simulation allows you to obtain dynamic forecasts of the state of traffic at a high detailed local level for the near future from thirty to sixty minutes. [4]

Comparison of the above software tools by the main generation criteria (see Table 1).

Table 1. Comparison of the above software tools by the main generation criteria.

	PTV Vissim	SUMO	Aim Sun
Customizability of generation (0..1)	1	0.6	0.9
Generating the environment	+	+	+
Support for specific vehicles	-	+	-
Generating pedestrians	+	+	+
Real-time analysis	-	-	+

These software tools are used to generate traffic on a pre-modeled section of the city, which does not allow using these products in a fairly short time. In addition, all the data of the program analyzes the traffic. This action may be unnecessary if you only need to generate a transport stream. After analyzing the results, we can conclude that it is necessary to create our own software solution for the task of generating virtual road traffic.

The paper examines the functionality of the tools needed to create a generation module. The analysis and selection of the development environment used is carried out. The choice was made between Unity and Unreal Engine.

Unreal Engine features include support for destructible objects, optimized and realistic handling of collisions between a high number of complex-shaped objects, dynamic lighting, multiple post-processing effects, and an artificial intelligence system.[9]

The main advantages of Unity are the presence of a visual development environment, cross-platform support, and a modular component system.

To select the development environment, we compared these software products (see Table 2).

Table 2. Comparison of software products.

№	Functional requirement	Unity, %	Unreal Engine, %
1	Working with textures	80	50
2	Working with materials	70	85
3	External libraries	80	70
4	Performance of large-scale scenes	80	85
5	Ability to optimize the code	60	80
6	Working with animation	70	80
7	Documentation	85	50
8	Working with 3D models	90	40
9	Graphics	75	90
10	External generation modules	65	50

Since this module will be used to develop a video stream emulator, it is necessary to compare the convenience of the development environment for all modules of this emulator and choose the most suitable one (see Table 3).

Table 3. Comparison of the module development environment.

№	Task	Unity, %	Unreal Engine, %
1	Generating a city	73.7	55.7
2	Generating transport	76	68
3	Generating people	75.9	62.3

Based on the results of the comparison, Unity was chosen as the development environment, since it better corresponds to the task at hand.

Unity supports the two main scripting programming languages Javascript and C#. Between these two languages, C# was chosen, since this programming language (abbreviated

YAP) has a more convenient object-oriented programming (abbreviated OOP). In addition, most of the examples in the Unity documentation are written in C#.

Unity contains a large number of components to facilitate the development of a software product. [8] These are the Transform component (responsible for determining the coordinates, rotation, and scale of an object on the scene); Animation windows (you can create and edit animations inside Unity itself without using third-party 3D animation programs); Raycast/Spherecast components, and other components.

In general, Unity has a large number of built-in tools that simplify and speed up development. In addition, thanks to the Asset Store platform, you can install and use third-party modules developed by other Unity users. Of the two possible development languages, C# was chosen, due to the fact that its support by the creators of Unity is getting better, and JavaScript support is getting worse.

3 Results

The multi-layer architecture was chosen as the application architecture. In this case, the functions of storing the representation and processing the data are separated. In this case, a three-level architecture is presented (the data storage layer, the presentation layer, and the business logic layer).

When the software product is divided into levels, it is possible to make changes to a certain level. No need to change the entire app.

A three-layer solution can be deployed on a single layer, (personal workstation).

In information system architectures that are logically divided into layers, the following three layers are most common:

- View layer (UI layer, user interface, view layer in a multi-level architecture)
- Business logic layer (domain layer)
- Data access layer (data storage layer)

The software product under development must meet the following requirements: Generation of vehicles (Passenger cars).

- Generate cars with custom parameters.
- Types of passenger cars (station wagon, sedan, hatchback, coupe).
- Car coloring (using textures).
- For all modes of transport, the following configurable parameters are available:
- Moving parts of the vehicle
  - Wheels
- Lighting
  - Far away
  - Near
- Numbers
  - Russian
  - Different regions
- Motion algorithm:
- Movement by points
- Cornering speed control
- Stop at traffic lights
- Speed adjustment with signs
- Braking when approaching another vehicle [10]
- Road intersections:
- The number of branches is limited to 8
- Ability to adjust the probability of rotation

- Traffic Lights:
  - Using colliders to synchronize with other modules
  - Using colliders to synchronize with other modules
  - Ability to adjust the duration of the signals
    - 5-120 seconds
- Signs:
  - Using colliders to synchronize with other modules
  - Ability to adjust the maximum speed of the vehicle
    - 40-130 km / h
  - The ability to adjust the allowed direction of movement.
  - The ability to stop the vehicle.

An algorithm for creating a license plate has been developed. If you need to have a specific license plate, you can initially set the number or region manually when creating the vehicle, which avoids randomness where it is not necessary.

As the basis of the motion algorithm, we chose point motion, since in the future it is necessary to combine this module with the urban environment generation module, and this method of motion is the most optimal for implementing this requirement [11].

The scripts were written to manage and set traffic points, as well as to calculate the route.

A 3D model was developed to simulate the vehicle. This model is transferred to the scene and then configured by attaching various scripts to this object. Important parameters to adjust are the wheel mass, wheel radius, force point, and suspension settings.

The basic driving algorithm is as follows: until the speed of the vehicle has reached the maximum set speed, the force that was set in the settings at startup is applied to the wheels. When approaching a point, if the length of the special beam, which is calculated using the formula (1), is greater than the distance to the point itself, then the speed update method is called [5].

$$zVelocity * 2 + 1 \tag{1}$$

where z Velocity is the increment of the vehicle speed, along the local Z axis.

This method accesses a dictionary that contains a point that stores the speed at which you need to enter a turn. The arguments of the method are the previous and next points.

It is assumed that the vehicle has reached the point of movement when the length of the vector from the car to the point itself becomes less than (2).

$$zVelocity/6 + 0.5 \tag{2}$$

After that, the Destination Reached method is called from the Waypoint Navigator, which configures the next path. When the next points are set up, the wheel rotation method is triggered, which turns the front wheel collider to the next point [12].

In order to avoid collisions, special beams are used, which originate from the points configured at launch, at a length equal to (3).

$$zVelocity * 6 + 1.5 \tag{3}$$

At the ends of the beams, spheres with a radius of 2 are used. A method was also needed to brake the vehicle for a certain distance. For its implementation, coroutines were used. A coroutine is a function that can pause its execution with yield until the statement completes. The essence of the method is that until the speed of the car is equal to 0, the braking force is applied to all the wheels. The force required for braking is calculated using the formula (4).

$$F_{breaking} = a_{breaking} * m \tag{4}$$

where $a_{breaking}$ acceleration of braking, and m is the mass of the vehicle. The braking acceleration is calculated using the formula (5)

$$a_{breaking} = \frac{-v^2}{2x} \quad (5)$$

Where v is the speed of the vehicle, and x is the distance for which it is necessary to brake [5].

Braking takes place over the same distance, regardless of the speed of movement at the start of braking. Otherwise, the braking algorithm will become much more complicated, since you will have to determine in advance the presence of a traffic light along the way.

When creating the traffic light, colliders were used, which allow detecting interaction with foreign objects. Each traffic light has a zone that is directly in front of it. When entering this zone, the vehicle slows down and stops.

To create a speed limit sign, you also need to use colliders. When entering the area that belongs to this sign, the vehicle's maximum permissible speed changes.

The integration of this module with the virtual urban environment generation module was also implemented [13].

As a result, the algorithm for creating license plates is implemented, the main scripts and functions that were used to create a traffic system and create a vehicle are developed and described. A variant of combining the virtual road traffic generation module with the virtual urban environment generation module is considered.

4 Discussion

This development can have various applications. To verify the implementation of all functions, you need to perform functional testing. Since the module generates a large number of objects, it is necessary to perform load testing. It will help you identify peak loads. Thus, functional testing was carried out, during which compliance with the functional requirements for the software product was revealed. Functional testing is implemented for the following tasks:

1. License plate generation, including the function of manual number assignment and automatic generation.
2. The movement of the wheels taking into account the rotation, as well as testing the maximum angle of rotation.
3. Lighting, taking into account the ability to turn on and off the lighting when moving.
4. Driving algorithm with simulation of different turning angles for the vehicle, including generation with multiple vehicles to test braking [11].

The ability to adjust the branching probability was tested on boundary values. The result was the same as expected.

When testing traffic lights, different signal durations were set and checked. In addition, it was checked that the vehicle stops at a traffic light at a red signal.

To test the speed limit sign, it was set to different values in the settings before launching the program and after the vehicle passed through the zone of this sign, the maximum speed of the vehicle was checked.

The turn sign was tested using a movement point with many possible branches. Initially, all the turn options were available, then the turn sign script was used and the settings were alternately set that only the directions should remain straight, then left, and then right.

To test the stop sign, the vehicle was set different maximum speeds and checked whether it would have time to slow down. At speeds up to 80 km / h, the vehicle brakes properly, but at speeds greater than the specified speed, the vehicle may roll over or not have time to brake for the distance indicated by the sign. This is due to the physics of wheel collider objects and the inability to brake at a very high speed over a short distance. The results obtained during testing coincided with the expected results.

Load testing is important. During testing, the program was run alternately with a different number of movement points and the time required to load the scene was measured. After that, this scene was run together with the vehicle, where the number of vehicles was equal to the number of points on the scene, for a more convenient display of the result in the table. Then, in addition to vehicles, license plates were also generated. The loading time was measured in seconds. The results obtained are shown in the table 4.

Table 4. Load testing.

	Test 1	**Test 2**	**Test 3**	**Test 4**	**Test 5**
Number of objects	0	20	50	100	500
Time to load only with movement points. sec	2.55	2.6	2.75	3	3.6
Time to load from the vehicle. sec	2.55	3	3.3	3.6	5.8
Loading time with vehicle and license plates. sec	2.55	3	3.4	3.75	6.3

The second test was conducted on a stage with a generated city. In this case, the number of frames per second was checked. The results obtained are shown in Table 5.

Table 5. Test results.

	Test 1	**Test 2**	**Test 3**	**Test 4**	**Test 5**
Number of objects	0	20	50	100	500
Number of frames with movement points only, fps	50	50	50	50	49
Number of frames from the vehicle, fps	50	44	37	30	10
Number of frames with vehicles and license plates, fps	50	44	37	29	9

The virtual traffic generation module is tested in various modes. There are different test scenarios in which the software product is run with different parameters. Important is load testing, which shows the dependence of the number of frames per second and loading time on the number of objects that need to be placed on the scene.

5 Conclusion

The paper considers the problem of developing a module for generating virtual road traffic. In the course of the work, an analysis of the subject area and an overview of road traffic generation technologies were carried out. Based on the analysis, the cross-platform development environment Unity was selected for use in the project [14,15].

The analysis of existing solutions and their functionality allowed us to identify the shortcomings of these solutions for the task. This allowed us to avoid some mistakes during development. We also explored the functionality of various built-in and third-party Unity tools. These tools made it possible to simplify the development of the generation module. In addition to studying the theoretical information, the tools were tested in practice. This allowed us to choose the most suitable ones.

Several types of customizable vehicles were created and a license plate generator was created. When creating the motion algorithm, several options were tried. The algorithm of movement on points using the wheel collider component was chosen. This made it possible to use the physics engine built into Unity to calculate the braking path, the angle of rotation of the wheels, the number of wheel rotations per minute, and other values necessary for

development. All this made it possible to achieve a realistic movement of the vehicle. In addition, tools have been developed to manage the entire system as a whole [11,14].

This module can be expanded, complicating the traffic algorithm, adding new vehicles and signs. Thus, you can implement a module that will fully simulate traffic flows and then connect it to the urban environment generation module, obtaining convenient solutions for testing neural networks in certain conditions or, for example, for creating an environment in another project.

References

1. *Computer simulation*, dic.academic.ru, https://dic.academic.ru/dic.nsf/ruwiki/91889
2. *Official source of assets*, assetstore.unity.com, https://assetstore.unity.com/
3. A. Thorne, *The Art of Scripting in Unity* (DMK, SPb, 2016)
4. *Aimsun website*, aimsun.com, https://www.aimsun.com/
5. A. Cerny, *Physical driving behavior of vehicles on different ground surfaces and the implementation in Unity3D*, alexander-cerny.at, http://www.alexander-cerny.at/portfolio/wp-content/uploads/2015/12/BA1_CERNY_Alexander.pdf
6. *PTV Group website*, ptvgroup.com, https://www.ptvgroup.com/en/solutions/products/ptv-vissim/
7. *SUMO website*, sumo.dlr.de, https://sumo.dlr.de/index.html
8. *Unity documentation*, docs.unity3d.com, https://docs.unity3d.com/Manual/index.html
9. *Unreal Engine Official Website*, unrealengine.com, https://www.unrealengine.com/en-US/features
10. M. Bolsunovskaya, A. Leksashov, S. Shirokova, V. Tsygan, E3S Web of Conferences **244**, 07007 (2021)
11. Abramov, N., Rakova, V., Barinov, D., et al., IOP Conference Series: Materials Science and Engineering, 618(1), 012010 (2019)
12. M. Barinov, M. Bolsunovskaya, S. Shirokova, Transportation Research Procedia **54**, 692–698 (2021)
13. M.V. Bolsunovskaya, A.V. Leksashov, A.V. Loginova, S.V. Shirokova, E3S Web of Conferences **91**, 07011 (2019)
14. M. Bolsunovskaya, A. Gintciak, K. Beliaevskii, et al., *Proceedings of the 3rd World Conference on Smart Trends in Systems, Security and Sustainability, WorldS4 2019* **8903929**, 37–42 (2019)
15. A. Gintciak, K. Beliaevskii, M. Fomina, et al., *Proceedings of the 33rd International Business Information Management Association Conference, IBIMA 2019: Education Excellence and Innovation Management through Vision 2020*, 8852–8859 (2019)