Scientific Visualization, 2025, volume 17, number 5, pages 109 - 122, DOI: 10.26583/sv.17.5.12

Problems and Peculiarities of Visualization of Traffic Flows

O.P. Bobrovskaya[1,A,B], T.V. Gavrilenko[2,A,B], V.A. Galkin[3,A,B]

[A] Surgut State University
[B] Surgut Branch of Federal State Institute "Scientific Research Institute for System Analysis of the Russian Academy of Sciences"

[1] ORCID: 0000-0001-7045-9085, o-bobrovskaya@mail.ru
[2] ORCID: 0000-0002-3243-2751, taras.gavrilenko@gmail.com
[3] ORCID: 0000-0002-9721-4026, val-gal@yandex.ru

Abstract

Visualization of transport movement on the plane and in 3-dimensional space is presented. Examples of implementation using frameworks and libraries (WF, WPF, OpenVDB+Blender3D, Manim). A circle consisting of lanes in two-dimensional version and corridors in three-dimensional version was considered as a motion trajectory. The method of setting an arbitrary trajectory for the movement of agents in the model of transportation flow based on the action potential is considered. The formulas allowing to take into account the displacements of both the corridors of the trajectory and the positions of the agents in them are given. This allows us to correctly handle the rearrangements of agents between trajectories, as well as to visualize the simulation results.

Keywords: transport, data visualization, transport flow, rotation quaternion, OpenVDB.

1. Introduction

Modeling any process is only one component of the study. The key problem is the process of presenting the results for their subsequent evaluation and comparison.

For visualization, various methods can be used, from static to animation, depending on the needs. As the number of objects under consideration increases during the study, the following difficulties arise [1]:

- The resource costs for their processing and display on the screen increase;
- The volume of stored objects increases;
- The information content of the visual representation of the model, overloaded with objects, decreases;
- The quality of the resulting visualization, which accommodates a larger volume under consideration, deteriorates;
- The complexity of the implementation, providing scaling and interaction, increases.

This study examines traffic flow modeling, which allows solving problems of traffic flow management, selecting the optimal configuration [2], and planning the travel time. As noted in [3], computer modeling can significantly reduce the time of data analysis, and visualization can provide a visual representation of the ongoing process. Let us consider approaches to solving the problem of visual representation of the model.

On a plane, the visualization task is to clearly represent the parameters and properties of traffic flows. There are no problems when depicting cars: objects cannot intersect. Difficulties can arise with a large number of agents, since the image becomes overloaded for perception and difficult for the computer to process. In this case, it is possible to move from the level of micromodeling to meso- or macromodeling [4], and depict as objects not individual cars, but a set of them, or to abandon this image altogether, replacing it with numerical characteristics.

The increasing complexity of the transport network has led to the fact that the transport infrastructure is expanding not only on the plane, but also in the 3rd dimension, which leads to the complexity of the image of tunnels and multi-level junctions [5], since parts of the road will overlap each other. To solve this, you can introduce translucency of the upper level or select the current image level.

The problem of the relationship between realism and the performance of the model calculation, which occurs with a large number of agents, can be solved in different ways. In the work [6], graphic primitives are used to depict cars on the road - circles of a given color, since this is sufficient for the task of visually demonstrating the movement of cars along the network. In the freely distributed application for modeling SUMO, the user is allowed to choose the degree of detail of the agent image depending on the current task [7]: whether they will be primitive triangles moving along a schematically designated road on a white background, rectangles closer to reality, or almost realistic figures of cars against the background of textures of the surrounding world.

In 3D space, it is much more difficult to represent the flow and clearly depict the movements of individual vehicles in it, since a significant portion of the agents are overlapped by those that are closer to the observer's position. In addition, as noted in [8], with an increase in the number of attributes/dimensions, in this case, the dimensionality of the space in which the agents move, the accuracy of the observer's judgment about each of the available attributes separately decreases.

Even depicting a single object and its movement in 3D space is difficult. Usually, they limit themselves to depicting coordinate systems (stationary and associated with an aircraft), the trajectory of a single object or their projections onto a plane. It is more difficult to convey the interaction of several objects in space, especially in the form of a still image. It is easier to understand the spatial arrangement of objects and their movement in several successive frames.

A relatively small number of agents can be depicted in detail as is, and at the same time understand how they are located by rotating the image around several axes. However, an increase in the number of agents will lead to an increase in the cognitive load and complicate the task of perception and analysis. Research on visual attention shows that even perceiving and assessing the number of dots in an image takes time, and this time increases with the number of dots [9]. Thus, if there are at least more than 1000 agents, it will be problematic to perceive them and their trajectories separately.

One solution to the problem of overlapping agents, if there are enough of them, can be to depict agents as a flow (liquid or gas cloud). If the liquid is opaque, then only objects located on the outer boundary can be observed. But more often the liquid is partially transparent and is depicted as a set of small primitive objects (cube, sphere). When storing in memory and subsequently depicting each object directly, to process their large volume, you can use specialized tools, such as the OpenVDB library, which works with sparse objects (initially smoke and flame).

Or you can depict only the macroparameters of the traffic flow, that is, use three-dimensional visualization of scalar fields (in our case, these are fields representing the density or modulus of the velocity of objects). The following methods are common:

- isosurfaces, level surfaces - surfaces of constant value of a scalar field. To extract a level surface from a field, it is necessary to calculate points with a given constant value of the field in the cells of the scalar grid [10].
- volume rendering [11] – a set of methods that represent a three-dimensional data set as a two-dimensional image (for example, the maximum intensity projection method).

2. The used traffic flow model

There is a traffic flow model based on the action potential [12], in which cars are represented by particles that attract or repel each other depending on the distance between them. Below we will consider how to visually represent the data obtained during the operation of

this simple model, as well as its generalizations to 3-dimensional space. The trajectory for the movement of agents was strictly defined by a circle with several corridors in which lane changes are possible.

The aim of the study is to generalize the model by adding the ability to specify an arbitrary trajectory by a set of break points of a certain broken line.

The solution to the problem will expand the capabilities of the traffic flow model, which is relevant in view of the current state of the transport network and the need to find solutions to increase the capacity of roads, and, accordingly, to model the traffic flow. In addition, the development of unmanned transport poses the task of analyzing ways to introduce new types of cars into the existing transport system, which is also solved using modeling.

3. Visualization approaches

Depending on the purpose of visualization, several directions can be distinguished.

If it is necessary to visually represent the movement of a traffic flow on a two-dimensional plane, it can be depicted as a sequence of images of the position of all cars on the space model changing with a given frequency. Any software tool that allows you to display graphic primitives and save them as an image is suitable for this.

In the case of a flow image in 3-dimensional space, it is preferable to use programs that allow you to work with volumetric objects, both statically and dynamically.

OpenVDB allows you to create a Grid of points with a given density and save a file with the extension ".vdb", which opens in the Blender application. Another Grid of the same points allows you to set the color for these points in Blender. In this way, you can save the speed of agents, and then display traffic jams in colors, as on the M. Treiber website (https://traffic-simulation.de/ring.html). Blender render engines: 1) Eevee (faster); 2) Cycles (more objects can be shown).

Manim is a convenient tool for creating high-quality animation using Python code. Allows you to depict 3D objects.

If we do not have time to monitor the movement of many objects, then it is convenient to depict their generalized, aggregated characteristics on graphs:

1) dependence of the position on a road segment on time (see Fig. 1);

2) fundamental diagram;

3) generalized flow characteristics (speed, number of lane changes) depending on time.

4) thermodynamic characteristics, in particular, temperature, as in [13].

There are also ways to visualize geospatial data for a more general depiction of the movement of transport flows associated with logistics.

4. Possible implementation methods

As already mentioned, there is a mathematical model of the traffic flow that needs to be visualized. So far, the trajectory of the flow is a closed line without intersections or with one intersection (one-dimensional manifold). The rules of behavior are set for vehicles, according to which they interact with each other. Therefore, we will consider visualization suitable for microscopic modeling (images of individual agents).

4.1. Windows Forms (C#).

A tool for developing a graphical interface, which is still supported, but has ceased to be developed due to the release of WPF. It is possible to output graphic primitives to a certain area, display graphs, and use controls to execute user commands in real time. The following was implemented using the described capabilities (Fig. 1):

- The field on the left simulates the movement of cars, represented as colored circles.
- The model parameters were defined that are available for the user to change (number of cars, track diameter, speed limit, number of lanes (one or two), track shape (circle or

eight), number of autopilots among the total number of cars, repulsion and attraction coefficients).

- The graphs display the dependence of the coordinate on the track on time and the fundamental diagram in real time.

A timer was used to visualize the movement of cars. Every 0.001 s, a new position of the cars is calculated, the old image is deleted and a new one is displayed in its place.

In the implementation of the model under consideration, most of the previously described problems are not observed due to the small scale of modeling and the number of road configurations.

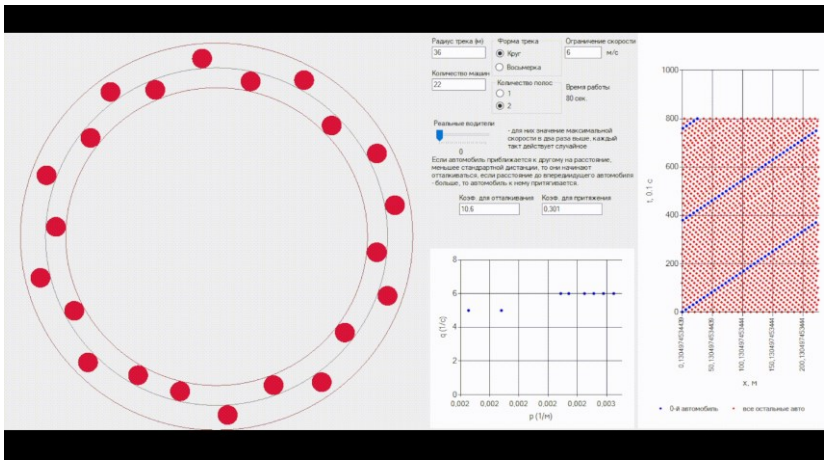


Fig. 1. Interface of the application created in WF

When the track diameter increases (due to a large number of cars), the scale decreases, so starting from a certain diameter, the cars are displayed small. In addition, the application response time increases with a large number of agents or long-term modeling (since old values are not removed from the graphs).

A disadvantage of WF is the lack of 3D display capability.

4.2. NET Core WPF.

A more modern and multifunctional tool allows you to implement the above application interface. At the same time, it provides the ability to depict a three-dimensional space and its projection onto the camera plane, which is necessary for further expansion of the traffic flow model to the 3rd dimension.

The features of representing volumetric objects make working with this tool difficult. Fig. 2 shows agents moving in space along one corridor from different viewing angles.

Fig. 2. Movement of agents (tetrahedrons) along a circle in WPF

Tetrahedrons are, in this case, graphic primitives in WPF, which were used due to the lack of spheres. Accordingly, the problem of orienting agents along the route appears, which is not necessary for circles and spheres and has not been solved previously, therefore, it requires modifications to the algorithm.

Visualization of the required movement of agents can only be implemented using rotations relative to the axis passing through the centers of the corridor circles, which is an ad hoc solution and will hinder future expansion of the model. Therefore, this tool will not be used in the future.

4.3. OpenVDB.

An open source library written in C++ allows you to save spatial positions of objects. Files saved in this way can then be imported into Blender3D. The transport flow model was implemented in python, for which the OpenVDB library is also available.

It was not possible to configure OpenVDB to work in Windows, so the work was carried out in Debian, deployed on WSL (Windows Subsystem for Linux). Installation according to [https://github.com/AcademySoftwareFoundation/openvdb] and building the library for Python3 (OPENVDB_BUILD_PYTHON_MODULE=ON [https://www.openvdb.org/documentation/doxygen/build.html]). Then, a simple import of the pyopenvdb library allows you to use its capabilities in Python.

```
import pyopenvdb as vdb

for j in range(1200): # simulation time 2 min
    ...
    grid1 = vdb.FloatGrid() # creating a grid
    grid2 = vdb.FloatGrid()
    grid1.name = 'density' # assigning a name to a grid
    grid2.name = 'temperature'
    acc1 = grid1.getAccessor() # getting a pointer to a grid, allowing voxels to be modified
    acc2 = grid2.getAccessor()
    for i in range(N): # loop through all objects
        ijk = (int(x), int(y), int(z)) # tuple of object coordinates
        acc1.setValueOn(ijk, 0.95) # assigning the desired value to a voxel of a grid at
        given coordinates
        acc2.setValueOn(ijk, (v/speed_limit))
    vdb.write(f'../../../../mnt/c/Users/user/project_openvdb/track/mygrids{j}.vdb',
    grids=[grid1, grid2]) # saving grid to file
```

During the calculation of new agent positions, a set of their coordinates was saved to files with the extension ".vdb". The second grid, saved in parallel, contains the values of agent velocities. After completing the calculations of a 2-minute simulation, the files (1200 in total) were imported into the Blender application as a new object on the scene. To display agent velocities as the color of spheres, nodes were added in the shader editor. Then the video was rendered (it is possible to render a set of individual images - Fig. 3, with subsequent assembly) and saved.

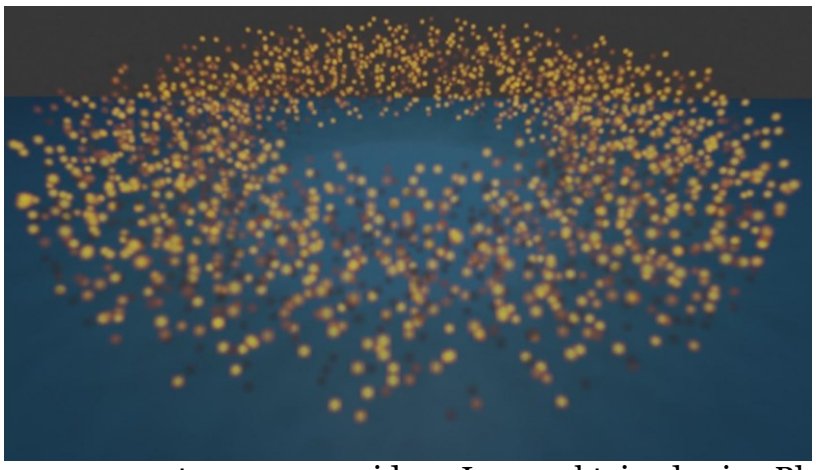


Fig. 3. 4000 agents on 100 corridors. Image obtained using Blender

After getting familiar with OpenVDB, the solution to the problem of preserving the position of a set of particles is not difficult (it requires a couple of lines of code). Using Blender3D for color mapping and visualization of particle movement is also not difficult (there is enough information on the Internet). The difficulty is in improving the image quality.

4.4. Manim.

A free open source python library (https://www.manim.community/) for creating animation in the field of mathematics provides the ability to work with objects that have 3 dimensions. For this, there is a special 3D scene class and objects that can be placed on it. After a fairly convenient interactive camera movement in blender, it can be difficult to control the camera using two angles and without the ability to set the distance from the camera to the origin. On the other hand, Manim has the ability to place objects not only in integer coordinates, but also on any real ones.

A program written using this library allows you to use pre-prepared data (in this case, the coordinates of agents saved in a text file during the calculation of the model) for visualization (Fig. 4):

```
class Agents3D(ThreeDScene): # declaration of a class inherited from 3D scene
    def construct(self):
        self.set_camera_orientation(phi=50*DEGREES, theta=-45*DEGREES) # determining the position of the camera
        ...
        for i in range(N): # adding all particles to the list and to the scene
            dots.append(Dot3D().shift(UP*x+LEFT*y+OUT*z).set_color(BLUE).scale(0.5))
        for j in range(1199*N): # cycle for each object at each moment of time (2 min)
            temp_coords.append(dots[itr%N].animate.move_to(UP*x+LEFT*y+OUT*z)) # adding movement animations to an object and saving them to a list
            if j % N == N-1: # if all objects for a given time point are processed
                self.play(*temp_coords, run_time=0.05) # display all movement animations on the screen
                temp_coords.clear()
        self.wait(1)
```

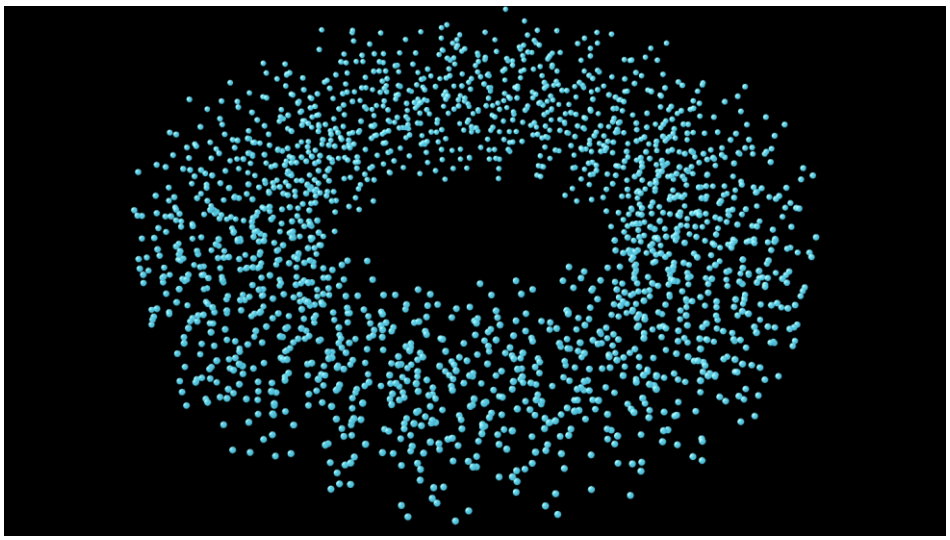


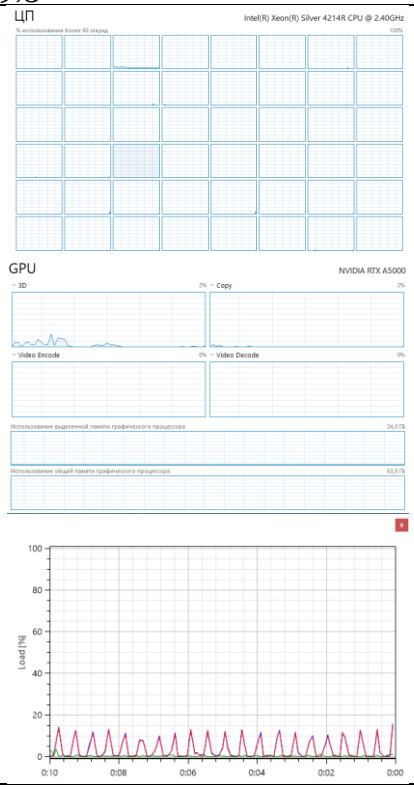
Fig. 4. 2000 agents on 100 corridors. Image obtained with Manim

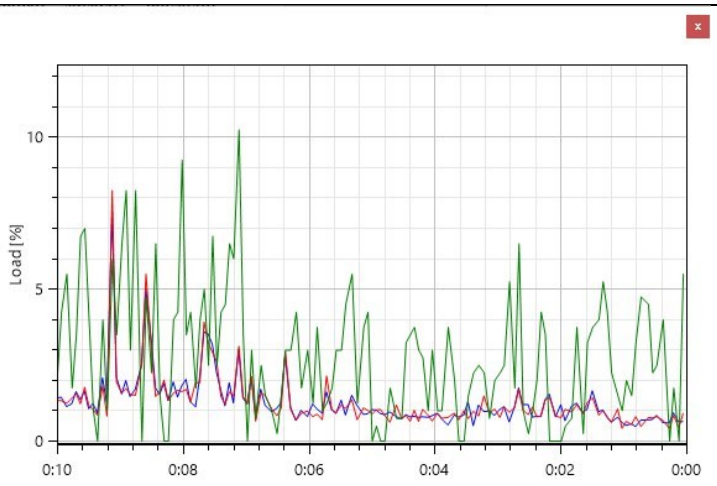
Visually, the agents seem sharper, but also less realistic than in Blender.

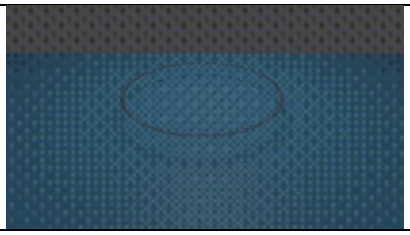
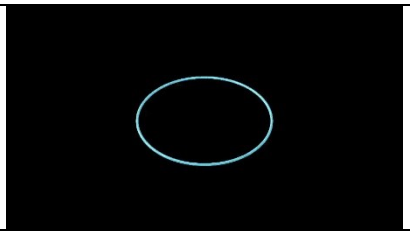
4.5. Comparison

To compare the last two options in terms of time and resources, the following experiment was launched. 400 objects are initially located on one lane and have an initial speed of 8 m/s. The track radius is 36 m. The number of available lanes is 16, located in a 4x4 square. Simulation time is 2 min. The results are shown in Table 1.

Table 1. Comparison of OpenVDB and Manim

	OpenVDB+Blender3D	Manim
Rendering time	6,4 h	9,3 h
Resource costs for the process		
Code execution in wsl:		



Resulting file size	14,9 MB	16,0 MB
The size of the file storing the implemented coordinates of the points	Folder with 1200 vdb files, size from 87 to 101 KB Total 119 MB	txt 13.9 MB
Code size	Only application controls were used	20 lines for processing data saved in txt
Result		

Manim subjectively shows higher quality with a small number of agents due to its good default settings. However, as the number of agents increases, Manim significantly increases rendering time and degrades quality, which can only be improved by significantly more processing time.

5. Arbitrary trajectory

To ensure the ability to move agents along an arbitrary trajectory, it is necessary to specify a generator, a receiver, and a trajectory connecting them. For the generator (source) and receiver (sink), the intensity is specified - the number of agents created and removed, respectively, per unit of time.

The trajectory can be specified as a broken line, specifying the breakpoints. To process agent movements, the distance traveled from the start of the movement (generator) is stored.

The trajectories of agents located in corridors, except for the zero one, differ from the trajectory of the zero corridor in length (which was not taken into account on the circle): at turns, the distance traveled along a segment of the trajectory increases or decreases. To reflect this, the model has implemented functions that calculate the extension and compression of individual corridors, depending on their distance from the zero corridor.

Since the displacements of the corridors depend on the displacements on the previous segment of the path, the calculation will be made at the beginning, when creating the trajectory. To do this, it is necessary to determine the offsets for the first segment, and for all subsequent ones, a rotation will be performed that coincides with the rotation of the direction vector of the straight line segment.

The offsets along the relative axes y (y_offset) and z (z_offset) for the first corridor are calculated as follows (we want to implement the offset along y in the XOY plane).

They are orthogonal to the direction vector of the corridor line, therefore:

$$\bar{e} \cdot (x - x_0\ y - y_0\ z - z_0) = 0$$

where $\bar{e}$ is the unit direction vector of the line on which the trajectory segment lies.

$$y - y_0 = \frac{e_z z_0 - e_x(x - x_0)}{e_y} + y_0$$

$$x - x_0 = 1$$

$$y - y_0 = \frac{e_z z_0 - e_x}{e_y}$$

$$y_{offset} = \frac{y - y_0}{|y - y_0|}$$

$$z_{offset} = e \times y_{offset}$$

For subsequent corridors, the offset directions depend on the offset directions of the previous corridor and are calculated differently.

The rotation quaternion is calculated as:

$$\Lambda = 1 + (\overline{n_0}, \overline{n_1}) + [\overline{n_0}, \overline{n_1}],$$

where $\overline{n_0}, \overline{n_1}$ are two given vectors between which the angle of rotation must be found [14].

$$\Lambda_{norm.} = \frac{\Lambda}{|\Lambda|}$$

This quaternion uniquely defines the rotation along the shortest path that the direction vector of a trajectory segment has made to become the direction vector of the next trajectory segment.

A special case is a 180-degree rotation, for which $\overline{n_0} = -\overline{n_1}$. In this case, the value Λ=0, which is processed as described below (one of the many suitable vectors is chosen arbitrarily).

The quaternion describing the rotation is calculated as:

$$\Lambda = \begin{cases} \dfrac{\Lambda_{non-norm.}}{|\Lambda_{non-norm.}|}, & |\Lambda_{non-norm.}| \neq 0 \\ \dfrac{(0; -n_{0z}; n_{0y})^T}{\sqrt{{n_{0z}}^2 + {n_{0y}}^2}}, & |\Lambda_{non-norm.}| = 0, \quad |n_{0x}| < 0.58 \\ \dfrac{(n_{0z}; 0; -n_{0x})^T}{\sqrt{{n_{0z}}^2 + {n_{0x}}^2}}, & |\Lambda_{non-norm.}| = 0, \quad |n_{0x}| \geq 0.58 \end{cases}$$

Next, the Rotation.from_quat($\Lambda_{norm.}$) function of the scipy library (python3) is used to rotate the corridor displacement vectors by the same angle (in scalar-last (x, y, z, w) format).

As a result, the displacement vectors along the y and z axes are rotated by the same angles as the direction vector of the line when moving from one trajectory segment to another.

The stretching (compression) of the corridors will be done as follows. The trajectory segment is divided in half, and each half is deformed in accordance with the junction of the adjacent trajectory segment. Direction vectors are constructed, the intersection point is found, and the displacement for the agent coordinate is determined.

The location of the agent in the zero corridor is determined by the formula:

$$p = e(l - L_i) + T_{r\_p},$$

where T is a sequential set of vertices of all trajectory break points; L_i is the cumulative length of the path traveled to the trajectory segment for the i-th agent; r_p is the index of the right boundary of the route segment found using binary search.

Let us define the multiplier γ, which characterizes the expansion/compression of the corridor depending on the intersection point with the adjacent corridor.

$$\bar{p} + \gamma\bar{e} = \overline{p'} + \gamma'\overline{e'}$$

where p is the location of the agent, p' is a point lying on the line of the adjacent corridor, e is the direction vector of the line of the considered corridor, e' is the direction vector for the adjacent corridor.

$$\gamma = \frac{e'_y p_x - p'_x e'_y + p'_y e'_x - p_y e'_x}{e_y e'_x - e_x e'_y}$$

In this case, x and y are conditional and can be replaced by another pair of coordinates if one of the direction vectors of the lines has a zero value.

The same is used to calculate the changed lengths of the shifted corridors. This allows us to correctly take into account the location of agents in adjacent corridors when changing lanes.

The result is shown in Fig. 5-8.

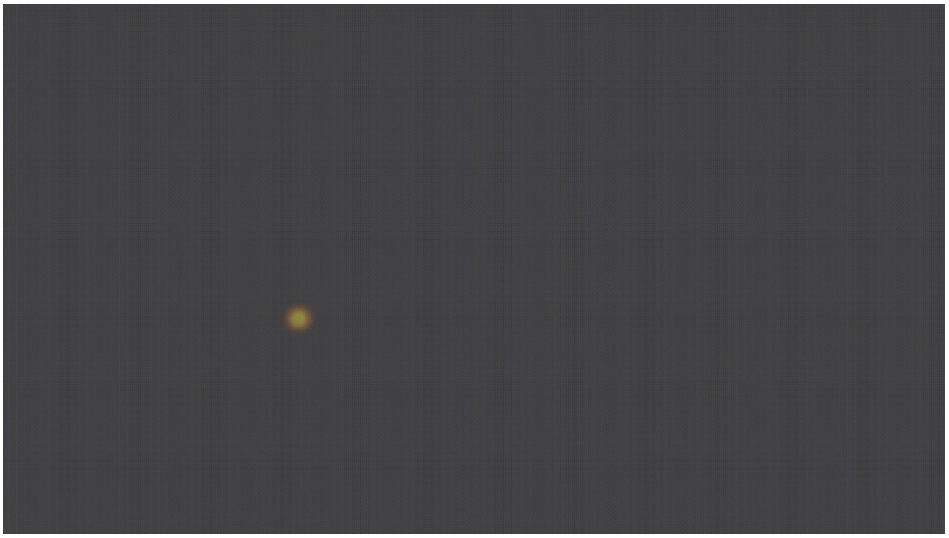


Fig. 5. Example of arbitrary trajectory 1.

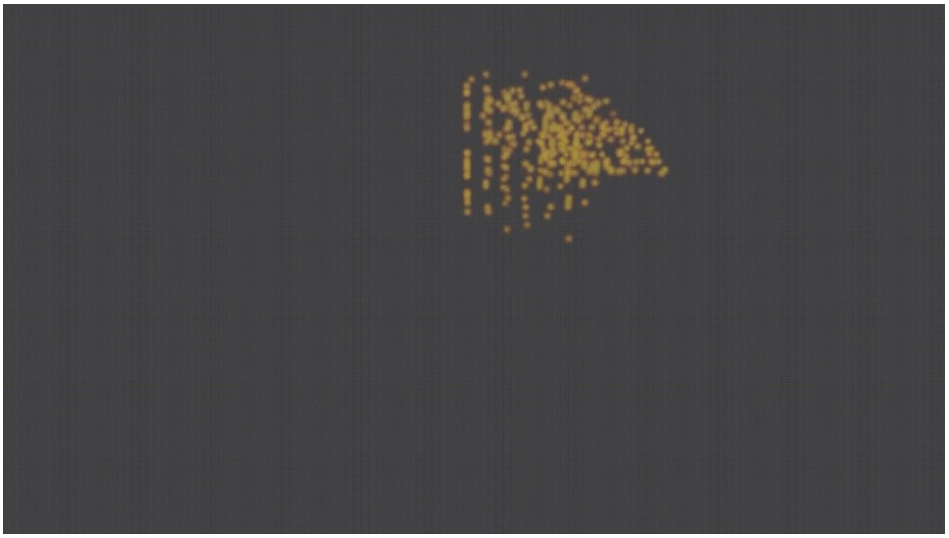


Fig. 6. Example of arbitrary trajectory 2.

Fig. 7. Example of arbitrary trajectory 3.

Fig. 8. Example of arbitrary trajectory 4.

Fig. 9 shows a closed arbitrary trajectory. The operating principle is similar to that provided on the circle: upon reaching the conditional "end" of the trajectory, the agent moves to the beginning of the trajectory.

Fig. 9. Closed arbitrary trajectory.

Figure 10 demonstrates the possibility of selecting an individual agent in a flow.

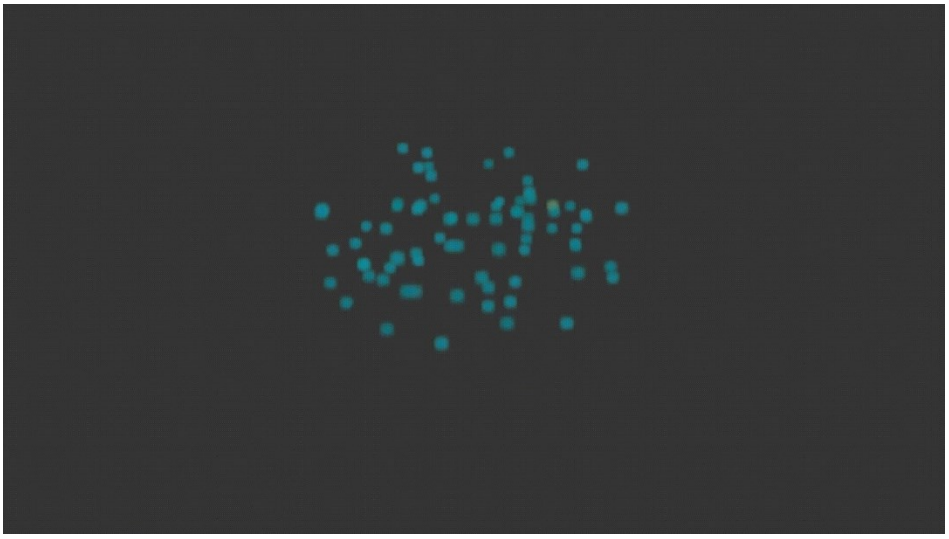


Fig. 10. Agent selected in the flow.

This is achieved by creating two additional Grid meshes and setting them to a different color in Blender.

Conclusion

Several methods of modeling traffic flows were considered. WinForms and C# can be suitable for a small model of two-dimensional movement. WPF, OpenVDB and Manim are suitable for modeling three-dimensional traffic flows. Of the latter, WPF is intended more for static placement of 3D objects and their simple transformations than for implementing dynamics. OpenVDB works with both C++ and Python3, the positions of objects on the scene saved with it can be imported into Blender3D for visualization. Manim allows you to visualize the movement of three-dimensional objects using Python code. On small models, Manim wins in terms of memory usage and resource costs. But with an increase in the number of modeled elements, Manim rapidly increases costs and loses to Blender.

In the future, the OpenVDB + Blender bundle will be used, allowing you to conveniently perform intermediate editing of the visual appearance and work with large models.

The modeling of an arbitrary trajectory in space is considered: formulas that provide for the extension and compression of the outer corridors of the trajectory during turns, and quaternions that allow preserving the relative location of agents in the corridors during turns.

The existing traffic flow model was expanded by the ability to specify an arbitrary trajectory. The issues of ensuring the correct relative location of the corridors of agent movement in space and calculating their location were resolved. The capabilities of the traffic flow model were significantly expanded, ensuring comparative experiments with a large number of trajectory configurations.

Acknowledgments

The work was carried out within the framework of the government contract of the National Research Center "Kurchatov Institute" - NIISI on topic No. FNEF-2024-0001 " Development and Deployment of Trusted AI Systems based on New Mathematical and Algorithmic Approaches and Fast Computing Models Compatible with Domestic Computer Hardware" (1023032100070-3-1.2.1).

1. Ali S.M., Gupta N., Nayak G.K., Lenka R.K. Big data visualization: Tools and challenges // 2016 2nd International Conference on Contemporary Computing and Informatics (IC3I), Greater Noida, India, 2016, pp. 656-660. (doi: 10.1109/IC3I.2016.7918044).
2. Li A.V., Repkin V.S., Semenov G.Yu., Sermavkin N.I., Faerman V.A. Komp'yuternoye modelirovaniye problemnogo perekrestka v srede AnyLogic [Computer simulation of a problem crossroad in the anylogic environment] // Bulletin of the Tomsk Polytechnic University. Industrial cybernetics, Vol. 1, №2, 2023, pp. 1–10, (doi: 10.18799/29495407/2023/2/16) [in Russian].
3. Marukhlenko P.G. Trekhmernaya vizualizatsiya dvizheniya transportnykh potokov na avariynoopasnom uchastke dorogi [Three-dimensional visualization of traffic flow on the accident-prone road section] // Mining informational and analytical bulletin, №11, 2016, pp. 408-417.
4. Adekunle A. A Review of the Difference Among Macroscopic, Microscopic and Mesoscopic Traffic Models, 2017. (doi: 10.13140/RG.2.2.11508.65929).
5. Котиков Ю.Г., Савченко К.А. 3D-моделирование многоуровневых транспортных развязок на базе платформы ArcGIS // ГИС в меняющемся мире. 2010. №4 (55). URL: https://arcreview.esri-cis.ru/2010/10/12/3d-modeling/.
6. Tishkin V.F., Trapeznikova M.A., Chechina A.A., Churbanova N.G. Modelirovaniye transportnykh potokov na osnove kvazigazodinamicheskogo podkhoda i teorii kletochnykh avtomatov s ispol'zovaniyem superkomp'yuterov [Simulation of traffic flows based on the quasi-gasdynamic approach and the cellular automata theory using supercomputers] // Computer Research and Modeling, Vol. 16, №1, 2024, pp. 175–194 (doi:10.20537/2076-7633-2024-16-1-175-194) (https://www.mathnet.ru/rus/crm1157) [in Russian].
7. Panovski D. Simulation, optimization and visualization of transportation data // Networking and Internet Architecture [cs.NI]. Institut Polytechnique de Paris, 2020.
8. Miller G.A. The magical number seven, plus or minus two: Some limits on our capacity for processing information // Psychological Review, Vol. 63, № 2, 1956, pp. 81–97. (doi: 10.1037/h0043158)
9. Kaufman E.L., Lord M.W., Reese T.W., Volkmann J. The discrimination of visual number // The American Journal of Psychology, Vol. 62, 1949, pp. 498–525.(doi: 10.2307/1418556)
10. Timokhin P.Y., Mikhaylyuk M.V. Metod izvlecheniya poverkhnostey urovnya na GPU s pomoshch'yu programmiruyemoy tesselyatsii [The Method to Extract Isosurfaces on the GPU by Means of Programmable Tessellation] // Programmirovaniye, № 3, 2020, pp. 66-72 (doi: 10.31857/S0132347420030103).
11. Xu C., Sun G., Liang R. A survey of volume visualization techniques for feature enhancement // Visual Informatics, Vol. 5, № 3, 2021, pp. 70-81. (doi: 10.1016/j.visinf.2021.08.001).
12. Bobrovskaya O.P., Gavrilenko T.V., Galkin V.A. Model' transportnogo potoka, osnovannaya na vzaimodeystvii chastits s potentsialom deystviya [Transport flow model based on interaction of particles with action potential] // Vestnik KRAUNC. Fiz.-mat. nauki, Vol. 40, № 3, 2022, pp. 72–87 (doi: 10.26117/2079-6641-2022-40-3-72-87) [in Russian].
13. Bobrovskaya O.P., Gavrilenko T.V., Galkin V.A. O primenenii ponyatiya temperatury dlya opisaniya transportnogo potoka [The Traffic Flow Temperature Concept] // Russian Journal of Cybernetics, Vol. 5, № 2, 2024, pp. 26-34 (doi: 10.51790/2712-9942-2024-5-2-03) [in Russian].
14. nabbla1. "Likbez po kvaternionam, chast' 6: povorot po kratchajshemu puti" [A tutorial on quaternions, part 6: shortest path turning] // LiveJournal : Russian-owned social networking service. 22 february 2018 (https://nabbla1.livejournal.com/187019.html).