

# Constructing 3D Scenes of Scientific Visualization Using CinemaScience Format

Pavel Vasev<sup>1</sup>, Sergey Porshnev<sup>1,2</sup>, Majid Forghani<sup>2</sup>, Dmitry Manakov<sup>1</sup>, Mikhail Bakhterev<sup>1</sup> and Ilya Starodubtsev<sup>1,2</sup>

<sup>1</sup>*N.N. Krasovskii Institute Russian Academy of Sciences, 16 S.Kovalevskaya str., Ekaterinburg, 620108, Russia*

<sup>2</sup>*Ural Federal University, 19 Mira str., Ekaterinburg, 620002, Russia*

## Abstract

To visualize any new entity, a visualization should be designed and programmed. Investigating approaches for programming new scientific visualizations, we come to the following idea: utilize CinemaScience format to describe 3D scenes. CinemaScience is developed for storing and visualizing supercomputer and physical modelling results, and differs with simplicity both for human and machine. It has a set of interesting features, for example it allows to specify dynamics in views dependent on parameters. However its current known applications are of 2D graphics, and in this paper we extend it for 3D. It's main idea is to treat Cinema artifacts as visual objects of explicit type. We successfully used the suggested approach in various visualization tasks, examples are presented in the paper. We developed the open-source web application that implements the suggested approach.

## Keywords

scientific visualization, declarative visualization, 3D visualization, cinemascience

## 1. Introduction

Scientific visualization is a valuable part of modeling pipeline, allowing researchers to perceive computation results. Sometimes existing visualization methods are not enough, especially if a novel research is performed. In that case, a new visualization should be created.

To construct a new visualization, a developer has to perform various steps [1], including:

1. Design a view, which implies specification of visual objects, their attributes, their placement; scene dynamics, if any; and user interaction.
2. Design a mapping between investigated entities (processes, objects, phenomenons) and the view.

The designing of a visualization is a very creative process. But its implementation is a very time-consuming process, and that is a problem. Visualization systems suggest a solution for that problem in the form of visual languages, e.g. GUI, for implementing visualizations. In addition,

---


*GraphiCon 2021: 31st International Conference on Computer Graphics and Vision, September 27–30, 2021, Nizhny Novgorod, Russia*

✉ vasev@imm.uran.ru (P. Vasev)

🆔 0000-0003-3854-0670 (P. Vasev); 0000-0001-6884-9033 (S. Porshnev); 0000-0002-9443-3610 (M. Forghani); 0000-0001-6852-8096 (D. Manakov); 0000-0001-8016-9946 (M. Bakhterev); 0000-0002-3494-4611 (I. Starodubtsev)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

such systems offer scripting and/or API for the same purpose. Also, specialized languages are developed.

As an example of using GUI for visualization implementation we point out Kepler.gl. This is a specialized geospatial visualization system. It provides visual language that allows importing data, constructing a view as a combination of basic views, and connecting structural parts of imported data with the view's inputs.

Another example, a set of systems like ParaView, Visit, 3dMax, and Blender, provide both visual language (in the form of GUI with menus, buttons, and other graphical elements) and scripting. Besides, platforms like Unity, Unreal, Roblox, Xclu.dev, and Viewzavr (which is based on ideas of our team), belong to this group.

There are systems that use special models for visualization programming. For example KNIME platform (uses data flow diagrams for defining scientific visualization pipelines) and SciVi platform (uses ontologies for declaring visual objects and semantic filters of data, see [2]).

Moreover, there exist specialized programming languages as Vega, A-Frame, and frameworks like Plotly, which allow creating visualizations primarily by scripting.

Finally, various 3D graphical formats like OBJ, STL, VRML, GLTF, so on have been developed with application for describing static scenes or scenes with limited dynamics.

**Our contribution.** We suggest a method of constructing 3D visualizations of some class. It allows to easily describe a set of 3D visual objects and configure a kind of scene dynamics and user interaction. The approach uses the CinemaScience format with extra semantics. As a result, with relatively small efforts an interactive 3D visualizations might be achieved.

## 2. Suggested approach

### 2.1. CinemaScience format

CinemaScience is a modern format developed to store massive data sets. In this format, data is stored in a directory called the Cinema database. This directory should contain an index information file named data.csv in the CSV text format. This file contains a table with columns for parameters and columns for data artifacts. Artifacts can be images, grids, CSV files; i.e., any type of data that can be written to disk. Thus this table defines a mapping between parameters and artifacts. The interpretation of these mapping is free and determined by the scope of an application which uses the format [3], [4].

Here is an example of a data.csv file (spaces between values are not necessary):

```
alfa, beta, FILE_image
0, 0, i1.png
0, 1, i2.png
1, 0, i3.png
```

This file describes a dataset which consist of two parameters, `alfa` and `beta`, and a one image artifact. The semantics of this definition may be different; for example, the application may allow users to select ranges of parameters and show images that correspond to the selected subset.

Other examples are provided at the project website <https://cinemascience.github.io>.

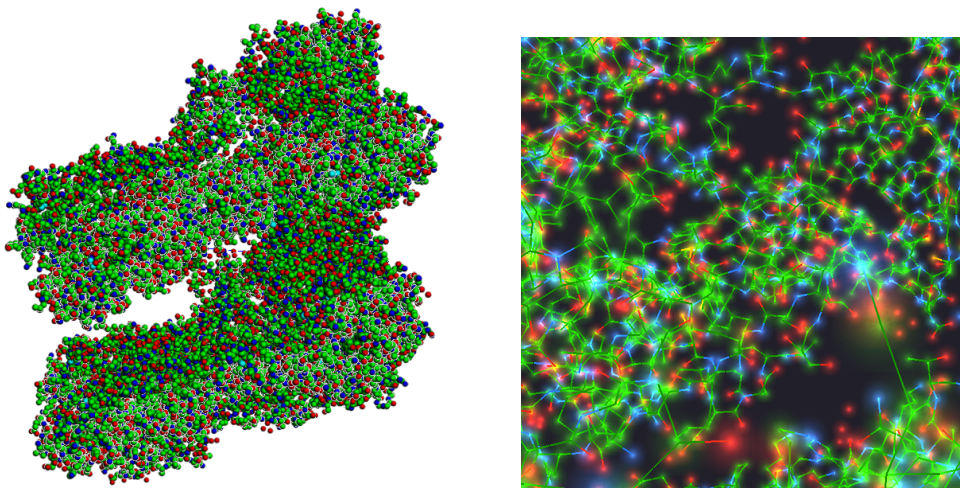
## 2.2. 3D artifacts

We specialize CinemaScience for programming 3D visualizations by the following.

Artifacts may be 3D visual objects. Thus a list of artifacts in a `data.csv` file specifies a list of visual objects. All these objects are displayed in a common 3D space. The type of particular visual object is determined by artifact name, in second underscores-surrounded word. For example, "FILE\_points\_atoms" artifact name specifies "points" object type.

The list of supported types is considered extensible. Our practice required us to implement points, lines (segments), triangles, spheres, vrml (for VRML files), obj (for OBJ files), vtkpoints (for VTK files). Currently, the type strictly determines not only the visual object look and behavior, but also the data format of its input files.

In figure 1, images for the influenza virus hemagglutinin are presented. These are generated by 3D scene computed by our team [5] from the data obtained from [6].



**Figure 1:** Influenza virus hemagglutinin 3D scene. The visualization task is to show specific atoms and their connections, which is used for modeling the viral evolution. All positions and colors are precomputed and saved in files. Atoms and their connections are depicted with points and lines. The view configuration of the scene is provided in CinemaScience format using the approach discussed in the paper. Camera position and rendering settings can interactively be configured by a user. On the left image, an overall picture is shown. On the right image a camera is zoomed inside.

The visualization program for this scene is the following "code" placed in a `data.csv` file:

```
T, FILE_points_atoms, FILE_lines_connections
0, w-coords.csv, w-lines.csv
1, w-coords-t1.csv, w-lines.csv
2, w-coords-t2.csv, w-lines.csv
```

Here, one parameter (T) and two artifacts (FILE\_points\_atoms, FILE\_lines\_connections) are defined. The parameter T represents the time. However, it is indifferent on the CinemaScience level because it operates parameter names just as identifiers. But the interpretation of artifact names is different.

For the first artifact, `FILE_points_atoms`, the `points` type is specified (by its name); that means loading and visualizing 3D points, coordinates, and colors for which are specified in artifact file. In the current scene it is used to depict atoms. For the second artifact, the `lines` type is specified; it is used in the scene to depict connections between atoms.

### 2.3. Parameters interpretation

The following logic is used: for each parameter column, a graphical control is generated (slider for numeric parameter and combobox for string parameter). A user manipulates these controls and thus specifies the combination of parameter values. This determines the current set of input files corresponding to visual objects.

In the scene above, there is the parameter `T`. A user is able to select a value of `T` using GUI and observe visual change in the scene. For example, by selecting `T=1` the file `w-coords-t1.csv` is displayed, whereas for `T=2` the file `w-coords-t2.csv` is displayed, both together with `w-lines.csv`.

## 3. Examples

### 3.1. Simple point cloud

Imagine there is a requirement to paint point cloud, e.g. a set of points in 3D space. Additionally, let this point cloud changes according to some parameter `theta` (it may have the meaning of time or any other meaning). Let the following dynamics in a scene be required: when a user selects `theta` value, a point cloud corresponding to that value should be displayed.

The solution using the suggested approach is as follows:

1. Put coordinates of points in a series of files, each file corresponding to some particular `theta` value. An example list of such files is shown in the Table 1a.
2. Each file should contain points coordinates in CSV format. An example content of such file is shown in the Table 1b. Each data line in it represents some point of the point cloud.
3. Create a Cinema database file `data.csv` with following content (Table 1c):

**Table 1**

The solution of the stated visualization task.

<code>pts_1.csv</code>	X, Y, Z	<code>theta,</code> <code>FILE_points_p</code>
<code>pts_2.csv</code>	0, 0, 0	0.1, <code>pts_1.csv</code>
<code>pts_3.csv</code>	0, 0, 1	0.2, <code>pts_2.csv</code>
...	2, 1, 0.15	...
<code>pts_10000.csv</code>	...	1000, <code>pts_10000.csv</code>

(a) List of data files

(b) Example of data file content

(c) Content of `data.csv` file

Thus we defined:

1. A list of data files, each containing coordinates of some point cloud instance (in `pts_*.csv` files). Each one is allowed to have its own amount of points.



2. A list of visual objects in a scene (artifact `FILE_points_p`) and their types (points).
3. A list of parameters in a scene (`theta`).
4. A relation between parameter values and input files for visual objects.

This information is enough for painting the required. Currently, our approach has a built-in algorithm that implements the required dynamics of a scene, as described in section 2.3. Parameter `theta` will be shown as graphical element. When a user will change value of that parameter, a related file with point cloud will be loaded by visual object and presented on a screen.

### 3.2. Fluid flow

In this study, the spreading of a highly viscous incompressible fluid was considered. The Navier-Stokes flow equation was considered in the form [7]:

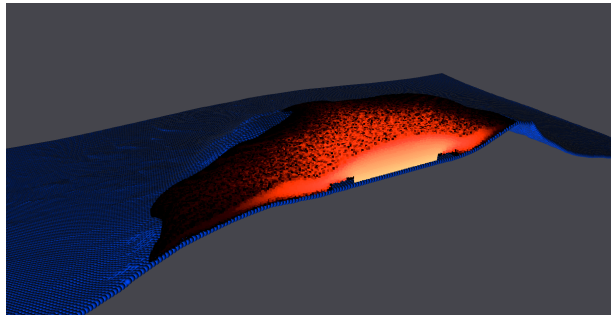
$$\frac{\partial \rho v}{\partial t} + \langle v, \nabla \rangle (\rho v) - \nabla \left( \mu (\nabla v + (\nabla v)^T) \right) = -\nabla p - \rho g \quad (1)$$

where  $\rho$ ,  $v$ ,  $\mu$ ,  $p$ ,  $g$  is a volumetric mass density, velocity vector, dynamic viscosity, pressure and gravitational acceleration respectively. The viscosity here is not a constant but is a function

$$\mu_i(t) = viscoStep * lifeTime_i + \mu_i(0) \quad (2)$$

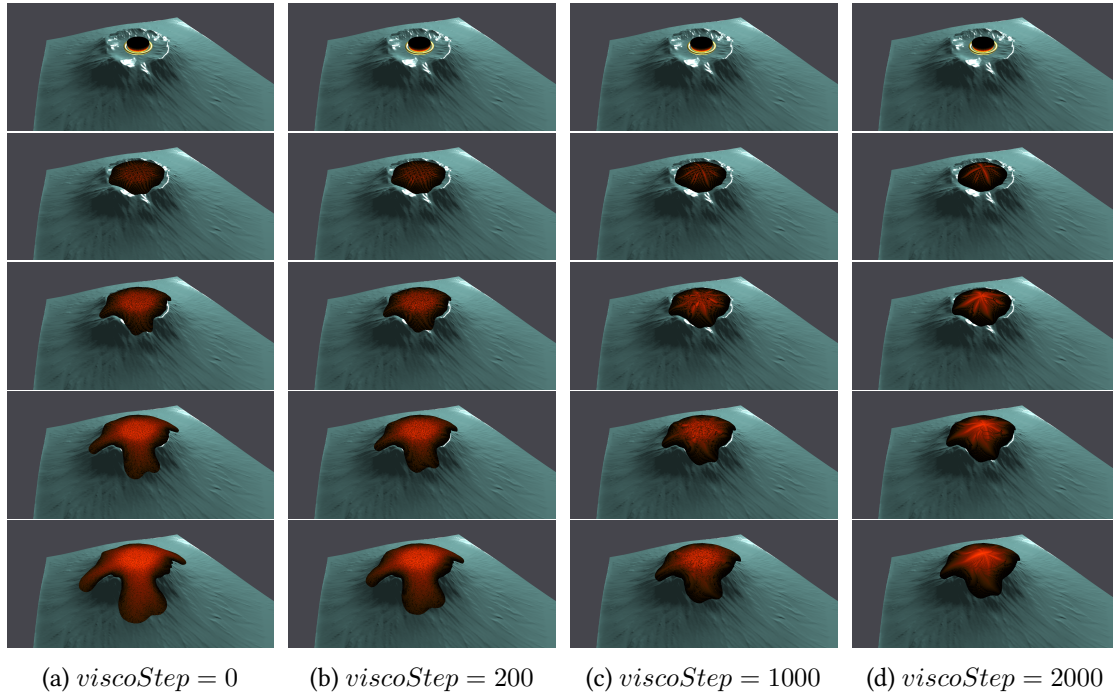
where  $\mu_i(0)$  is an initial viscosity (constant for all fluids);  $lifeTime_i$  is the lifetime of a particle after leaving the volcano;  $viscoStep$  is the constant responsible for the rate of change in viscosity. This form of viscosity allows an approximate description of the processes of change in viscosity associated with the formation of crystals in lava.

Numerical calculations were carried out using Smoothed particle hydrodynamics (SPH) methods. More details can be found, for example, in [8, 9, 10]. Figure 2 shows the results of spreading a viscous fluid along a slope. Also, in Figure 2 the surface of the slope is shown in the form of particles. This is employed to understand the method of approximating the boundary conditions [11]. For a numerical experiment, a volcano surface with a crater was randomly generated in the World Machine package.



**Figure 2:** The result of numerical simulation of the spreading of a viscous fluid along a slope. On the cut, the color indicates the viscosity of liquid particles, from yellow (low viscosity,  $10^5 Pa \cdot s$ ) to black (high viscosity,  $3 * 10^7 Pa \cdot s$ ).

Figure 3 shows the dynamics of spreading of a viscous fluid under various conditions of viscosity change.



**Figure 3:** Dynamics of spreading of a viscous fluid along a slope at different rates of viscosity change. (a) - there is no change in viscosity, it corresponds to spreading of a viscous liquid with constant viscosity. (b) - changes in viscosity are equal to half of the initial viscosity. (c) - changes in viscosity per conditional second are equal to the initial viscosity. (d) - changes in viscosity per conditional second are twice then the initial viscosity.

This visualization was achieved using Cinema database data.csv file of the following content:

T,	ViscoStep,	FILE_vtkpoints_lava,	FILE_obj_vulkan,	FILE_obj_emmitter
4,	0,	0/ParticleData_4.vtk,	vulk.obj,	emit.obj
5,	0,	0/ParticleData_5.vtk,	vulk.obj,	emit.obj
		...		
4,	1,	200/ParticleData_4.vtk,	vulk.obj,	emit.obj
5,	1,	200/ParticleData_5.vtk,	vulk.obj,	emit.obj
		...		
4,	2,	1000/ParticleData_4.vtk,	vulk.obj,	emit.obj
		...		
252,	3,	2000/ParticleData_252.vtk,	vulk.obj,	emit.obj

Let's denote interesting parts in contrast to the example presented in previous section:

1. There are two parameters defined,  $T$  and  $ViscoStep$ . While the first one is a time, the second one is the computational parameter representing viscosity changes. Graphical controls will be displayed in GUI for both parameters. Their values, selected by a user,

will determine current input files for visual objects. Thus a user might move his focus among  $T$  or among *ViscoStep*. This gives a researcher the opportunity to view results of numerical simulation with different parameters at the same modeling time.

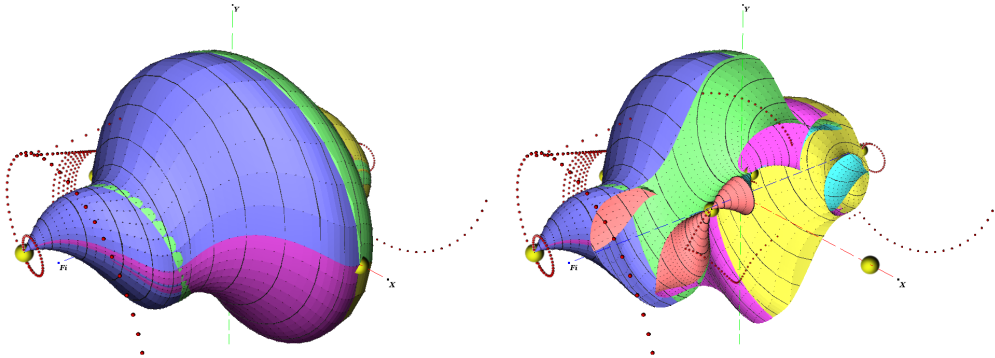
2. There are three visual objects defined, *FILE\_vtkpoints\_lava*, *FILE\_obj\_vulkan* and *FILE\_obj\_emitter*. The first one have type "vtkpoints", and the latter two have type "obj".
3. The vtkpoints type corresponds to VTK file format whereas the loader of that type was implemented especially for the visualization task being described and now made publicly available. The obj type corresponds to the OBJ file format. It was implemented earlier in another project.

The total size of data files (e.g. all vtk and obj files) of the scene presented in Figure 3 is about 4.3 Gb. Because it is only files corresponding to current parameters values are required at any given moment, we successfully performed visualization of this and similar scenes on a typical laptop.

### 3.3. Reachable set of Dubins car

In N. N. Krasovskii Institute of Mathematics and Mechanics, V. S. Patsko's working group performs long-term research on Dubins car model [12], [13]. This model is widely used to describe various motions: an airplane in a horizontal plane, a car, etc.

The approach suggested in this paper (of using CinemaScience for 3D scenes) was practically born in response to visualization required by that research. An example resulting image is presented in Figure 4.



**Figure 4:** A reachable set of Dubins car at instant. The set is represented by its boundary and shown as a colored surface. A user might specify current instant  $T$  and the appropriate set is loaded and displayed. In addition, some extreme motions of interest are shown by red dots. Yellow dots represent points of that extreme motions which correspond to the current instant  $T$ . The right image represents the same reachable set with `clip X` filter applied. Details of this study are covered in [14].

For this figure, a scene data.csv file has the following typical content:

T,	FILE_vrml_rset,	FILE_treki_all,	FILE_treki_currentT
1.00,	Symm-20.vrml,	treki.csv,	pts_1.00.treki.csv
1.05,	Symm-21.vrml,	treki.csv,	pts_1.05.treki.csv
1.10,	Symm-22.vrml,	treki.csv,	pts_1.10.treki.csv
		...	
4.00,	Symm-80.vrml,	treki.csv,	pts_4.00.treki.csv

Here are one parameter `T` which denotes time instant and three visual objects. The first one represents the reachable set surface; it is of type `vrml` which corresponds to VRML file format. The latter two of type `treki`, and this type was created especially for the current task.

It was necessary to show a set of points which denotes some extreme motions, e.g., tracks, interesting to a mathematician. Additionally, it was necessary to filter them, e.g. to show only particular tracks. For that purpose, we used `points` data type implementation and extended it to load additional column named `N` from input CSV file. A custom GUI control, which allows specifying values of interesting `N`, was added, and data filtering was implemented as a built-in ability of `treki` type. Thus the stated necessity was achieved.

## 4. Reflection

We see that the presented approach of using CinemaScience for 3D scenes is just a way to program; it is a programming language with semantics built for the description of relations between entities in a dynamic visual scene.

This semantics reflects the following. A Cinema database defines a list of parameters and separately their associated values, a list of artifacts, and a relation between parameters values and artifacts. The specified relation is further interpreted by some algorithm built into visualization software. In our approach, as stated in section 2.3, it is about choosing vector of artifact values.

There might be other algorithms. For example, CinemaScience authors use, among with the presented one, an algorithm based on multiple artifact values specified by parameter range restrictions using parallel coordinates [15].

Also, as demonstrated in [16], this relation might be considered in a wider sense as a relation of parameter values with particular attributes of artifacts. For example it is possible to specify those attributes by columns like `FILE_obj_vulkan.rotateY` where attribute name `rotateY` is encoded after a dot.

In [17] it is stated that the expressiveness of a programming language is based on:

1. Primitive expressions;
2. Means of combination;
3. Means of abstraction.

One of the questions of our interest is how far the suggested approach is able to move towards expressiveness according to that statement. In the current case, the primitive expressions are artifacts (and probably their relations with parameters). Then, they might be combined by listing them in a Cinema database. And finally, a Cinema database, theoretically, might be used itself as an artifact in other Cinema databases, which resembles some form of abstraction. E.g. one may use them by denoting in a common-like `FILE_somecinemadb_item`. This is the subject for future research.

## 5. Theoretical foundation

The modern development of high-performance computing opens up a number of new possibilities for solving mathematical modeling problems, including large-scale parametric researching and solving optimization analysis problems [18]. Firstly, one can consider these problems from the point of view of multi-criteria optimization for the solution of which, along with the lexicographic method or the weighted sum method, an interactive approach one can apply. Secondly, visual analytics solutions provide technology that combines the strengths of human and electronic data processing [19]. Thus, the use of interactive visualization in the field of high-performance computing is justified, including solving optimization analysis problems.

If it is proposed to use visual filtering (filtering alternatives based on their visual images) in [20], then this work is focused on justifying the applying and considering the advantages of controlling the parameters of both the visual program and the parallel program. To continue, it aims at the formalization of this approach, which from the point of view of programming one may call as *visual abstract parameters*. These are a special case of abstract data types, the function range of which is a dynamic visual image. Abstracting parameters are considered as the subset implementation of data abstraction models (generalizations that allow abstracting from the source and ontology of data during visual analysis) [21].

Unlike work [22], where parametric analysis is used to solving visualization problem and multi-criteria optimization combined, we do not create new views that depend on a parameter, but consider visualization as a process that one can control through parameters. If we draw a parallel between visualization and linguistics, we can say that we are not creating words but we are creating phrases. In principle, we can talk about abstraction of the visualization metaphor (continuous map from source domain into target domain). We abstract from the semantics of the sign, consider only the syntax, whereas the pragmatics in this case is generalization or formalization.

Thus, we have defined the task of abstracting parameters for the solution of which we have applied the extension of the CinemaScience format.

We suppose that it is beneficial to turn the development of a new specialized visualization system from a complex project (that includes many routine technical tasks) into a process of adaptation, usage and expansion of the capabilities of the visualization systems constructor. It is done for example in [23]. The authors have developed a series of visualization constructors. Their creation is not an end in itself. The goal is the optimal (in terms of time, cognitive effort) implementation of specialized visualization systems for visual analysis of applied problems.

In the development of specialized visualization systems, we continue to move from theory to practice. The theoretical research of visualization was also performed in the computer visualization laboratory of V. L. Averbukh, see [24]. Let us briefly dwell on visual analytics, the advantages of using parameters and dynamic visualization.

Visual analytics solutions provide technology that combines the strengths of human and electronic data processing. Visualization becomes the medium of a semi-automated analytical process, where humans and machines cooperate using their respective distinct capabilities for the most effective results. The goal of visual analytics is to make our way of processing data and information transparent for an analytic discourse [19]. Therefore, visual analytics is the science of analytical reasoning simplified by interactive visual interfaces.

Let us answer the question, what can be given in practical terms by reasoning about abstracting parameters. Advantages of using parameters in the development of specialized visualization systems are:

1. It might be used for steering of high-performance computing in-situ visualization (e.g. for on-line visualization) through passing and control of parameters.
2. The CinemaScience format structure is suitable for out-of-core algorithms implementation, such as using k-tree for multi-view visualization.
3. A continuous map (e.g. visualization) might be defined through a small parameters change (operational semantics) [25], in particular - an animation.
4. A dynamic visualization might be implemented.

We emphasize once again that dynamic visualization is not a set of words (signs) but is a phrase. We can consider it as any change of a visual image, but we must give a strict definition in terms of information theory.

For example, if the information does not change, a person achieves the sleep effect. It means that a person loses "trust" not only to the information being perceived but also to the information source.

Therefore, the rate of change of information should be considered, or mutual information should be considered. The change of information is a process; thus, we must consider data filtering, like any other visualization, as a process. Two options are possible: a human-computer interactive process and an animation.

## 6. Conclusion

The strong sides of the suggested approach for 3D scenes construction are the following. First, the CinemaScience format is extra simple to write. This allows a developer not to be overwhelmed by details like in other 3D formats. Second, this format is extra simple to read. That is, a format reader and a player algorithm may be easily implemented both as a standalone application or as a plugin to existing scientific visualization software.

Our preliminary results show that the approach is robust and relatively versatile. It is suitable for some class of scientific visualization tasks, especially related to visualization of results of computational modeling.

However, there is a nuance. The approach allows to define a set of visual objects and simple dynamics of a scene. Additional view settings like windows, clippings, transfer functions, and advanced a user interaction are out of the scope of the approach.

It is supposed that a user configures these aspects using the graphical interface of a visualization software. It then may be saved as project settings and reused in scenes of the same kind.

We develop software that implements the described approach as a web application using WebGL and WebXR technologies. It is available at <https://github.com/viewzavr/vr-cinema> page.

We would like to thank reviewers of the paper for their constructive feedback.



## References

- [1] V. Averbukh, Towards the conceptions of visualization language and visualization metaphor, in: Proceedings of IEEE Symposia on Human Centric Computing Languages and Environments: September 5-7, Stresa, Italy, 2001, pp. 390–398. doi:10.1109/HCC.2001.995296.
- [2] K. Ryabinin, S. Chuprina, Adaptive scientific visualization system for desktop computers and mobile devices, *Procedia Computer Science* 18 (2013) 722–731. doi:10.1016/j.procs.2013.05.236, 2013 International Conference on Computational Science.
- [3] D. Rogers, J. Woodring, J. Ahrens, J. Patchett, J. Lukasczyk, Cinema database specification dietrich release v1.2, in: Tech. Rep. LA-UR-17-25072, Los Alamos National Laboratory, 2018. URL: [https://github.com/cinemascience/cinema/blob/master/specs/dietrich/01/cinema\\_specD\\_v012.pdf](https://github.com/cinemascience/cinema/blob/master/specs/dietrich/01/cinema_specD_v012.pdf).
- [4] J. Ahrens, S. Jourdain, P. O’Leary, J. Patchett, D. H. Rogers, M. Petersen, An image-based approach to extreme scale in situ visualization and analysis, in: SC ’14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2014, pp. 424–434. doi:10.1109/SC.2014.40.
- [5] V. Averbukh, I. Mikhailov, M. Forghani, P. Vasev, 3d visualization to analyze multidimensional biological and medical data, in: International conference in honor of the 90th Birthday of Constantin Corduneanu, Ekaterinburg, Russia, Springer, 2018, pp. 241–251. doi:10.1007/978-3-030-42176-2\_24.
- [6] R. Xu, D. C. Ekiert, J. C. Krause, R. Hai, J. E. Crowe, I. A. Wilson, Structural basis of preexisting immunity to the 2009 h1n1 pandemic influenza virus, *Science* 328 (2010) 357–360. doi:10.1126/science.1186430.
- [7] Y. V. Starodubtseva, I. S. Starodubtsev, A. T. Ismail-Zadeh, I. A. Tsepelev, O. E. Melnik, A. I. Korotkii, A method for magma viscosity assessment by lava dome morphology, *Journal of Volcanology and Seismology* 15 (2021) 159–168. doi:10.1134/S0742046321030064.
- [8] D. Koschier, J. Bender, B. Solenthaler, M. Teschner, Smoothed Particle Hydrodynamics Techniques for the Physics Based Simulation of Fluids and Solids, in: W. Jakob, E. Puppo (Eds.), *Eurographics 2019 - Tutorials*, The Eurographics Association, 2019. doi:10.2312/egt.20191035.
- [9] D. J. Price, Smoothed particle hydrodynamics and magnetohydrodynamics, *Journal of Computational Physics* 231 (2012) 759–794. doi:10.1016/j.jcp.2010.12.011, special Issue: Computational Plasma Physics.
- [10] P. Randles, L. Libersky, Smoothed particle hydrodynamics: Some recent improvements and applications, *Computer Methods in Applied Mechanics and Engineering* 139 (1996) 375–408. doi:10.1016/S0045-7825(96)01090-0.
- [11] N. Akinci, M. Ihmsen, G. Akinci, B. Solenthaler, M. Teschner, Versatile rigid-fluid coupling for incompressible sph, *ACM Trans. Graph.* 31 (2012). doi:10.1145/2185520.2185558.
- [12] V. Patsko, S. Pyatko, A. Fedotov, Three-dimensional reachability set for a nonlinear control system, In *Journal of Computer and Systems Sciences International* 42 (2003) 320–328.
- [13] V. Patsko, A. Fedotov, Three-dimensional reachable set at instant for the Dubins car: Properties of extremal motions, 2020, pp. 1033–1049. 60th Israel Annual Conference on Aerospace Sciences, IACAS 2020.

- [14] P. Vasev, V. Patsko, A. Fedotov, Visualization of three-dimensional reachable set for the Dubins car, in: Proceedings of the 15th International Conference on Computer Graphics, Visualization, Computer Vision and Image Processing, 2021, pp. 119–123. URL: [http://sector3.imm.uran.ru/confers/MCCSIS2021/index\\_eng.html](http://sector3.imm.uran.ru/confers/MCCSIS2021/index_eng.html).
- [15] D. Orban, et al, *Cinema:Bandit*: a visualization application for beamline science demonstrated on XFEL shock physics experiments, Journal of Synchrotron Radiation 27 (2020) 1–10. doi:10.1107/S1600577519014322.
- [16] CinemaScience Composable Image Set extension, [https://github.com/cinemascience/cinema/blob/master/specs/dietrich/01/extensions/cis/1.0/cis\\_specification\\_v1-0.md](https://github.com/cinemascience/cinema/blob/master/specs/dietrich/01/extensions/cis/1.0/cis_specification_v1-0.md), 2021.
- [17] H. Abelson, G. J. Sussman, with Julie Sussman, Structure and Interpretation of Computer Programs, 2nd editon ed., MIT Press/McGraw-Hill, Cambridge, 1996.
- [18] A. E. Bondarev, On visualization problems in a generalized computational experiment, volume 11, 2019, pp. 156–162. doi:10.26583/sv.11.2.12.
- [19] D. Keim, G. Andrienko, J.-D. Fekete, C. Görg, J. Kohlhammer, G. Melançon, Visual Analytics: Definition, Process, and Challenges, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 154–175. doi:10.1007/978-3-540-70956-5\_7.
- [20] A. Zakharova, D. Korostelyov, O. Fedonin, Visualization algorithms for multi-criteria alternatives filtering, Scientific Visualization 11 (2019). doi:10.26583/sv.11.4.06.
- [21] D. Manakov, Data abstraction models: Sampling (parallel coordinates), filtering, clustering, Scientific Visualization 11 (2019). doi:10.26583/sv.11.1.11.
- [22] I. K. Romanova, Comparative analysis and modifications of visualization methods in parametric studies of control systems, Science and Education. Bauman Moscow State Technical University. Electronic Journal 1 (2017) 50–76.
- [23] P. Vasev, S. Kumkov, E. Shmakov, Extensible scientific visualization system, Scientific Visualization (in Russian) 4 (2012) 64–77.
- [24] V. L. Averbukh, Semiotic analysis of computer visualization, in: A. L.-V. Azcarate (Ed.), Interdisciplinary Approaches to Semiotics, IntechOpen, Rijeka, 2017. doi:10.5772/67729.
- [25] D. Manakov, V. Averbukh, P. Vasev, Visual text as truth subset of the universal space, Scientific Visualization (in Russian) 8 (2016) 38–49.

## A. Online Resources

Authors implementation of the suggested approach and example scenes are available via <https://github.com/viewzavr/vr-cinema>