Scientific Visualization, 2021, volume 13, number 4, pages 127 - 143, DOI: 10.26583/sv.13.4.10

An Experience of Using Cinemascience Format for 3D Scientific Visualization

Pavel Vasev[1,A], Sergey Porshnev[2,A,B], Majid Forghani[3,B], Dmitry Manakov[4,A], Mikhail Bakhterev[5,A], Ilya Starodubtsev[6,A,B]

[A] N.N. Krasovskii Institute of Mathematics and Mechanics of Russian Academy of Sciences, Ekaterinburg, Russia
[B] Ural Federal University, Ekaterinburg, Russia

[1] ORCID: 0000-0003-3854-0670, vasev@imm.uran.ru
[2] ORCID: 0000-0001-6884-9033, s.v.porshnev@urfu.ru
[3] ORCID: 0000-0002-9443-3610, majid.forqani@gmail.com
[4] ORCID: 0000-0001-6852-8096, manakov@imm.uran.ru
[5] ORCID: 0000-0001-8016-9946, mob@k.imm.uran.ru
[6] ORCID: 0000-0002-3494-4611, starodubtsevis@robolab.tk

Abstract

To visualize any new entity, a visualization algorithm should be designed and programmed. Investigating approaches for programming new scientific visualizations, we come with the following technique: 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, the technique currently known applications are 2D graphics, and in this paper we extend it for 3D case. The main feature of the approach is treating of 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. It is open-source and its main features are also described in the paper.

Keywords: scientific visualization, declarative visualization, 3d visualization, cinemascience.

1. Introduction

Scientific visualization is a valuable part of the modeling pipeline, that allows researchers perceiving computation results. Sometimes existing visualization methods are insufficient, 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. However, its implementation is very time-consuming. Visualization systems suggest a solution for that problem in the form of visual languages, e.g. GUI, for implementing visualizations. In addition, 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 introduce 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, quads, spheres, vrml (for VRML files), obj (for OBJ files), vtkpoints (for VTK files), gltf (for GLTF 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].

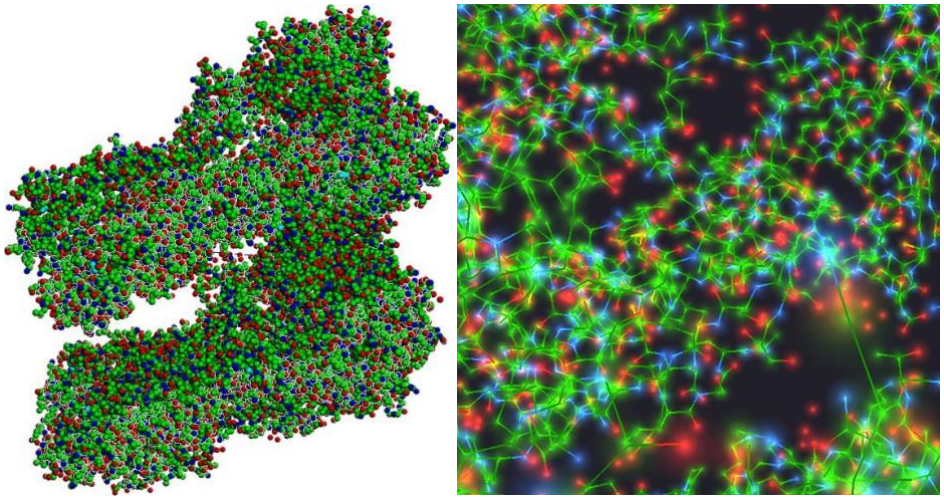


Figure 1: Influenza virus hemagglutinin 3D scene. The visualization task is to show specific atoms and their connections, which is used for modeling the viral evolution. All positions and colors are precomputed and saved in files. Atoms and their connections are depicted with points and lines. The view 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

For each parameter column, a graphical control is generated by the program (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 line in

a data.csv file and thus a 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 slider graphical control 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

Let us consider a requirement to draw 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.

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

(a) List of data files (b) Example 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 sufficient for the required drawing. Currently, our approach has a builtin 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. 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 [7], [8]. 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 2.

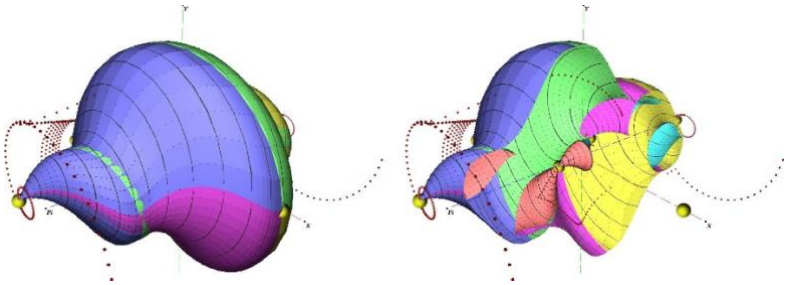


Figure 2: A reachable set of Dubins car at instant. The set is represented by its boundary and shown as a colored surface. A user might specify current instant T and the appropriate set is loaded and displayed. In addition, some extreme motions of interest are shown by red dots. Yellow dots represent 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 [9].

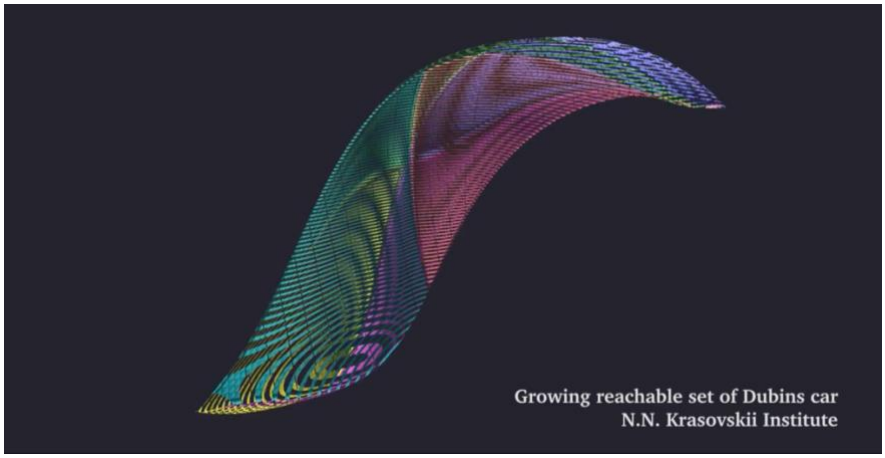
For this figure, a scene data.csv file has the following content:

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

The file contains the parameter T that 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 an additional column named N from the 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.

The visualization presented in this section required us to develop a lot of visual effects (filters), applied at the scope of visualization software (see [9]). This visualization is available online at https://viewlang.ru/dubins page and on Video 1.



Video 1: An example of achieved animation of a growing reachable set of Dubins car.

3.3. Wave fronts

In N. N. Krasovskii Institute of Mathematics and Mechanics, the problem of the propagation of wave fronts Φ in three-dimensional space is investigated by Dynamic systems department. The case of a spherical vectogram of velocities of a unit radius centered at the origin is considered. As the initial set, we consider a parametrically given curve Γ:

$$\begin{cases} x = 2\cos t \\ y = \sin t \\ z = \cos^2 t \end{cases}$$
$$t \in [0, 2\pi] \qquad (1)$$

The visualization task here is to show sections of growing wave fronts, see Figure 3.

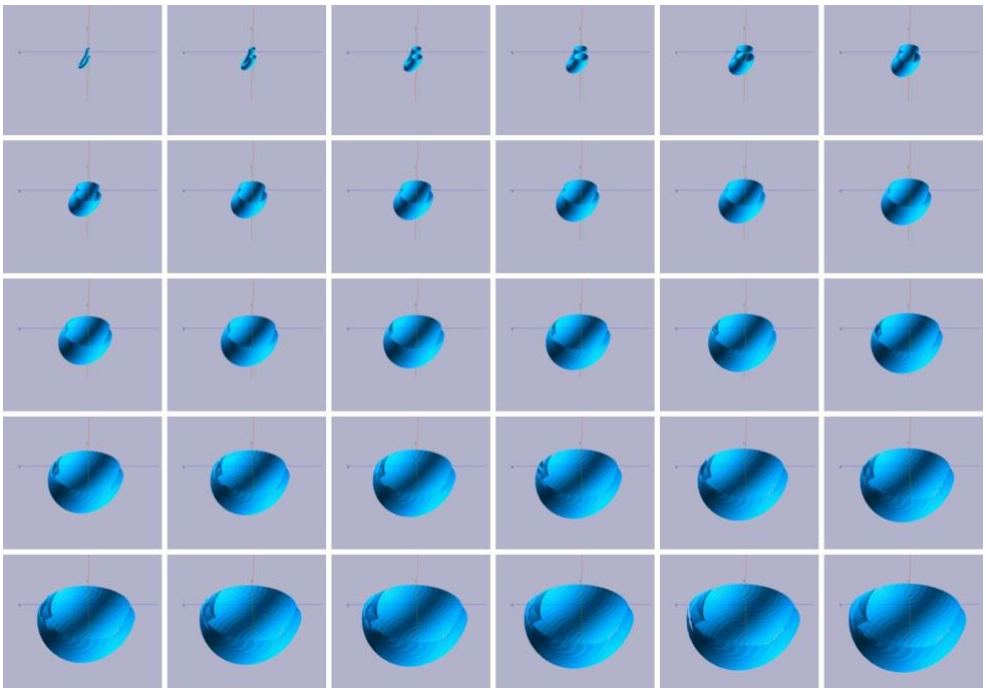


Figure 3: The figure shows 30 sections of wave fronts by the yOz plane (more precisely, the parts of the fronts located in the half-space x <= 0). The step along the propagation radius of the wave fronts Φ was chosen h=0.2. The basis for constructing the approximations of the fronts by triangles was formed by the algorithms proposed in the article [10]. The surfaces Φ have kinks caused by the presence of a singular set at those points at which waves come from 2 or more elements of the curve Γ.

To visualize this, the data.csv file of following content was created:

```
n,    FILE_triangles_surf
1,    surface/1.txt
2,    surface/2.txt
3,    surface/3.txt
                ...
30,   surface/30.txt
```

That is, we declared one parameter n and one visual object of type triangles (FILE_triangles_surf). The files in subdirectory surface (1.txt ... 30.txt) represent sections of wave fronts and have the following form of content:

X,	Y,	Z,	X2,	Y2,	Z2,	X3,	Y3,	Z3
-1.858,	-0.0137,	0.8582,	-1.8535,	-0.0425,	0.8554,	-1.8332,	-0.0432,	0.8793
-1.858,	-0.0137,	0.8582,	-1.8332,	-0.0432,	0.8793,	-1.8376,	-0.0139,	0.8821
				...				

These files define coordinates for triangles that form the surface of given wave front. With data.csv they provide enough information for creating visualization. Then a mathematician that runs visualization is able to choose value of parameter n and see section of corresponding wave front.

The visualization presented in this section is available online at https://github.com/viewzavr/vr-cinema/tree/main/examples/wafe-fronts page.

3.4. 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 [11]:

$$\frac{\partial \rho v}{\partial \mathrm{t}} + \langle v, \nabla \rangle (\rho v) - \nabla(\mu(\nabla v + (\nabla v)T)) = -\nabla p - \rho g \quad (2)$$

where ρ, v, μ, p, g is a volumetric mass density, velocity vector, dynamic viscosity, pressure and gravitational acceleration respectively. It is also assumed that the viscosity is not a constant but is a function

$$\mu_i(t) = viscoStep * lifeTime_i + \mu_i(0) \quad (3)$$

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 [12, 13, 14]. Figure 4 shows the results of spreading a viscous fluid along a slope. Also, in Figure 4 the surface of the slope is shown in the form of particles. This is employed to understand the method of approximating the boundary conditions [15]. For a numerical experiment, a volcano surface with a crater was randomly generated in the World Machine package.

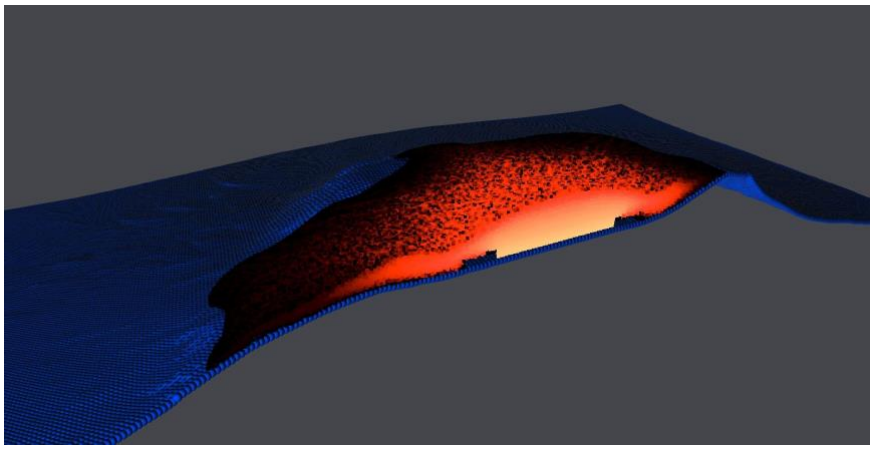
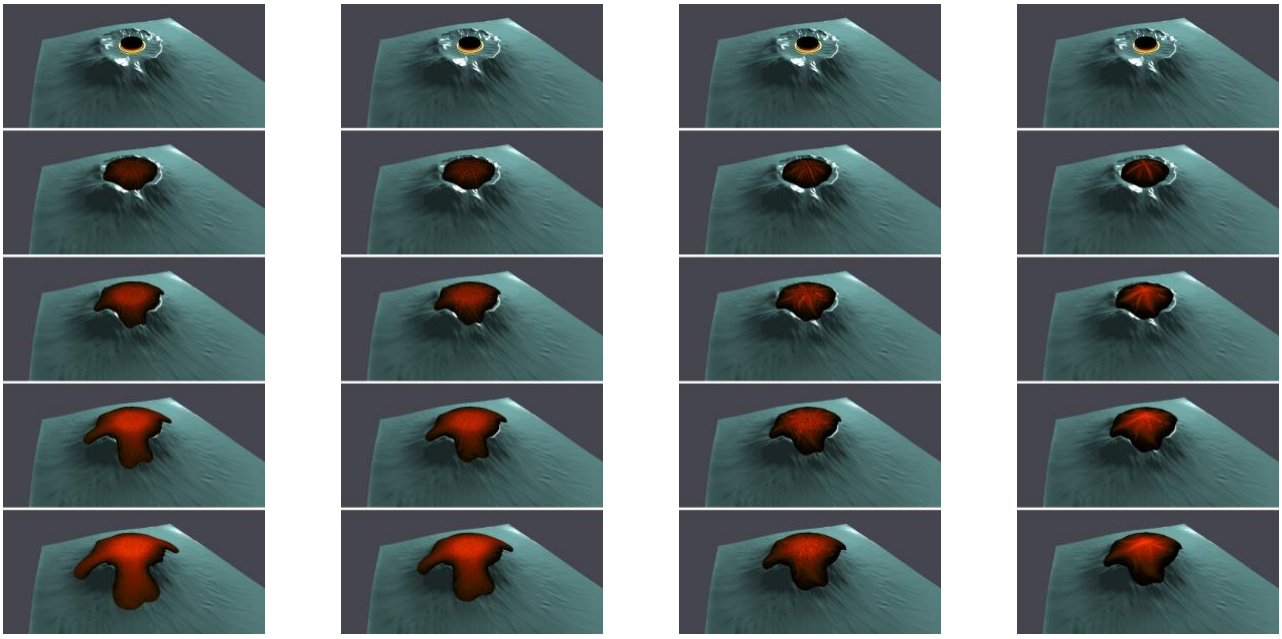


Figure 4: The result of numerical simulation of the spreading of a viscous fluid along a slope. On the cut, the color indicates the viscosity of liquid particles, from yellow (low viscosity, $10^5\ P\ a \cdot s$) to black (high viscosity, $3 * 10^7\ P\ a \cdot s$)

Figure 5 shows the dynamics of spreading of a viscous fluid under various conditions of viscosity change.



(a) $viscoStep = 0$ (b) $viscoStep = 200$ (c) $viscoStep = 1000$ (d) $viscoStep = 2000$

Figure 5: 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_emmiter
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 the previous section:

4. 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.

5. 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”.

6. 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 5 is about 4.3 Gb. Because it is only a subset of files corresponding to current parameters values are required at any given moment, we successfully performed visualization of this and similar scenes on a laptop with average performance characteristics.

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 a vector of artifact values. There might be other algorithms. For example, the CinemaScience authors use, apart from the presented one, an algorithm based on multiple artifact values specified by parameter range restrictions using parallel coordinates [16].

Also, as demonstrated in [17], 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 [18] it is stated that the expressiveness of a programming language is based on three elements:

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

One of the important questions in this research 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 research and optimization analysis problems solving [19]. 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 [20]. Thus, it is convenient and justified to use the interactive visualization in high-performance computing area, including application for optimization analysis problems solving.

In the considered in this work case of the use of visual filtering (filtering alternatives based on their visual images) in [21], the main focus is 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) [22].

Unlike the research in work [23], where parametric analysis is used for solving a combination of visualization and multi-criteria optimization problems, we do not create new views that depend on a parameter, but consider visualization as a process that one can control through parameters. We draw a parallel between visualization and linguistics, and say that we are not creating words but we are creating phrases. We can introduce an abstraction of the visualization metaphor (continuous map from source domain into target domain). We abstract from the semantics of the sign and 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 assume 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 [24]. 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.

With the development of specialized visualization systems, we are movign from theory to practice. The theoretical research of visualization was also performed in the computer visualization laboratory of V. L. Averbukh, see [25]. 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 [20]. 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. The advantages of using parameters in the development of specialized visualization systems are:

1. Approach can be 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) [26], in particular - an animation.
4. A dynamic visualization can be implemented.

We emphasize that dynamic visualization should be treated as not a set of words (signs) but a whole 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 a sleep effect. It

means that a person loses ”trust” not only in the information being perceived but also in 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. Implementation: VR-Cinema

We develop a visualization tool named VR-Cinema that implements the stated approach. The tool is implemented as a web application and executes in a web browser. It is based on Viewzavr framework, which uses WebGL for 3D graphics with the help of ThreeJS library. For demonstration purpose, the tool is presented online at https://viewzavr.com/apps/vr-cinema page. Source codes are available at https://github.com/viewzavr/vr-cinema repository. The overall current look of the tool is presented on Figure 6.

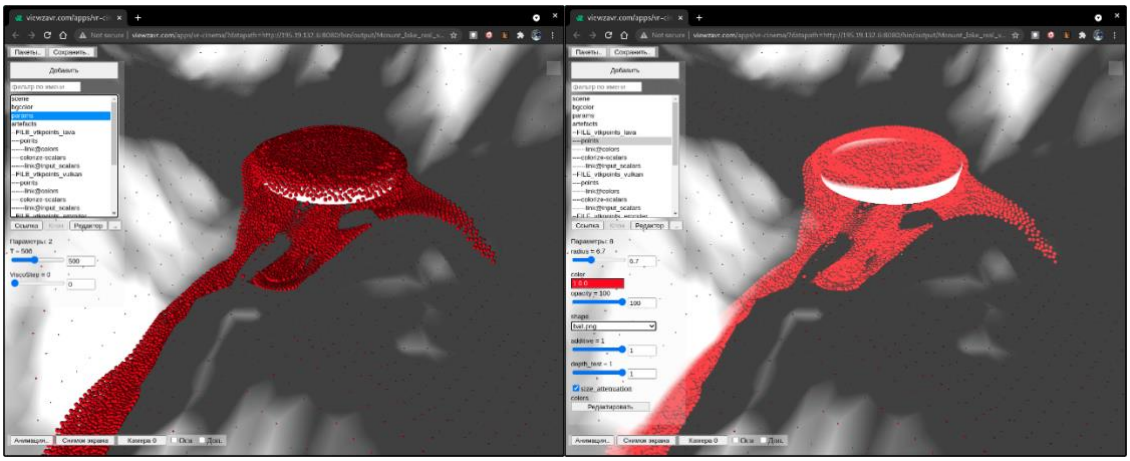


Figure 6: The VR-Cinema tool. Scene from “Fluid flow“ example is loaded. Both images organized as follows. First, a list of objects in the application environment is listed (a list box at the top-left). A user can select the current object and adjust it’s parameters (a set of GUI controls at the left-middle). Left image – a user defines values of Cinema parameters. Right image – a user interactively selects points visual object and performs visual tuning, setting it’s “additive” attribute to 1, which leads to additive mixing of points color over other objects.

6.1. Structure

The entry point of the application is index.html file. After startup, a Cinema database is loaded by URL specified in *datapath* query parameter. The URL can also be specified manually by a user.

The application determines which Cinema database should be visualized, loads *data.csv* file content and parses it. Parameters and artefacts are determined. Also additional artifact attributes are determined, see “Other features” section below.

Based on a list of detected Cinema parameters, a graphical user interface (GUI) is generated. For each Cinema parameter, there are three cases:

1. The type of parameter is string.
2. The type of parameter is numeric.
3. The type of parameter is numeric and interpolation is enabled.

According to the parameter’s case, an appropriate GUI type is selected.

For each artifact in Cinema database it’s corresponding visual object type is determined

(see “3D artifacts” section above which explain this). An object of that type is created in an application environment. The object internally creates structures for ThreeJS library, required for object operation. The object provides a message handler to setup a path to the input file which this object should visualize. The object receives a message with a new input file path, fetches the file content, parses it, and setups attributes of ThreeJS/WebGL structures from the file content.

As stated above, the GUI generated for parameters is used for selecting one value for each parameter. When user selects parameter values, a vector consisting of parameter values for all Cinema parameters is achieved. Application determines a line in a Cinema database with values nearest to selected vector, and thus determines paths of input files for visual objects. As objects are already created and placed in the environment, the application sends for each of them a message with a new input file path. That implements the algorithm of application operation.

6.2. Visual object types

During our practice like in Examples section, we detect the list of visual object types required for solving incoming visualization tasks. Following visual object types are currently implemented:

1. Points.
2. Spheres.
3. Lines.
4. Triangles.
5. Quads.
6. OBJ files.
7. VRML files.
8. VTKpoints files.
9. GLTF files.
10. CinemaScience sub-databases.

Detailed information about visual object types and their input files is specified at https://github.com/viewzavr/vr-cinema/blob/main/format.md web page.

6.3. Visual tuning

The practical application has demonstrated that supplying coordinates for visual objects is insufficient. For a user the possibility for fine-tuning of visual objects is essential. For example, to specify a triangle’s surface colors and shininess, point’s radius and sprite image, rotation of a model from GLTF file. On one hand, these parameters might be stored in input files of visual objects on per-primitive basis (e.g. per-point, per-triangle, etc.). On the other hand, it is comfortable to perform tuning of these parameters visually, in GUI, together for all primitives of a visual object.

Thus application provides GUI for tuning parameters of visual objects. The list of such parameters for tuning particular visual object is determined in the scope of the object.

Values of tuning parameters, together with camera settings (position and target), and current selected values of Cinema parameters, forms a state of the visualization program. This state is repeatedly get stored to the web page address, in its hash part, in a serialized JSON form. The JSON format was selected because it nicely serializes/deserializes to string. This technique allows to (re)load web page and receive the same program state and a view as before.

Beside that, when working in local server mode, the program’s state is repeatedly saved to settings file, also in JSON format. This is explained in details in section below.

The presented approach, named visual tuning, has benefits and drawbacks. The benefit

is higher user performance and comfort, because operating visually in GUI is faster than regenerating cinema database. The drawback is program state: it is out of the scope of Cinema approach.

However keeping such state in software-specific format allows providing additional features, not covered by CinemaScience standard. This has a positive effect: visualization software and CinemaScience 3D extension are able to evolve independently.

6.4. Local file server

While web-based approach used for VR-Cinema application has a lot of advantages, it also has some restrictions. For example, web applications have restricted access to file system of user's computer. This complicates one popular visualization scenario when files to be visualized are located locally, on the user's computer.

To solve this complication, we developed a tool for running local file server with ease. It only requires one-time installation of NodeJS framework on a user's machine. Then, to perform a visualization, user have to run following command in command line:

npx vr-cinema

This command instantly starts visualization of Cinema database from the current directory. Internally it starts a local web server, and opens a web browser with VR-Cinema application and with url pointed to the created server.

We used *npx* ecosystem for providing access to the tool. Actually, when *npx vr-cinema* command is issued, the *npx* loads the package named *vr-cinema* from *npm.org* internet database and passes control to it. Then it performs as described. This allows us to upgrade the tool easily, because *npx* checks and downloads new versions of the tool to user's machine automatically.

The tool might be used not only for running things locally. For example, it might be used for serving files from remote systems, for example supercomputers. In this scenario, files from high-performance computations become easily available for instant visualization.

We also implemented a rescan feature in the tool. It tracks *data.csv* file repeatedly and if it sees the file change, it notifies VR-Cinema application and the latter reloads the currently opened Cinema database. This allows visualising the results as they get computed. The notification is performed using WebSockets technology.

Additionally, a rescan-follow feature was implemented. After VR-Cinema reloads *data.csv* by rescan feature, it automatically setups it's first parameter to the maximum value from *data.csv*. This allows for visualization to "follow" running computation. However this feature is not a common case, and currently might be turned on and off by the user manually.

6.5. Interpolation

For some scenes it was found useful to compute interpolated input data for visual objects. This allows the smoothing of the transitions between visual object for synthetic parameter values that are not listed in *data.csv* but are between those values. Currently the following algorithm is used.

1. If interpolation mode is turned on, allow to specify parameter values not just from data.csv but any values from min..max range of corresponding columns.
2. Find two lines *line1* and *line2* in *data.csv* of minimal distance between vector of line's parameters values and vector of values specified in GUI by a user. Consider these distances as $d1$ and $d2$.
3. The weight w from range 0..1 for interpolation is determined by equation

$$w = d1 / (d1 + d2) \quad (4)$$

4. For each visual object, send a message with weight w and a pair of corresponding

input files extracted from *line1* and *line2*.

As seen in step 4, interpolation is performed in the scope of a visual object. Currently, for all objects that load input data from CSV files, we perform whole-file per-value linear interpolation (for numeric values; string values are taken from first file). This suits well for objects with same amount of vertices.

6.6. Other features and plans

We implemented a special visual object of type *cinema*. It's input data file is a path to *data.csv* of some other Cinema database that has to be included in a current scene. It was found useful to propagate parameters from child database to parent database. Thus a user is able to manage all parameters from all child databases in one GUI of the root database. This is a step towards abstraction, as explained in Reflection section.

We take advantage of managing additional artifact attributes, e.g. attributes of visual objects, right in Cinema databases. A column named *FILE_artifact->ATTR* means that values in that column specify values for attribute ATTR of visual object *FILE_artifact*. For example, *FILE_gltf_obj1->ROTATEX* will specify rotation around X axis of visual object *FILE_gltf_obj1*. The feature is under development and is connected to Visual tuning feature (see section above).

It is obvious that just showing visual objects, even with dynamics based on parameters, is not enough for successful visualization. Often filtering, transformation, and other effects on visual information are required.

We are working on a model of visual effects. Visual effect is meant to be applied to one or more visual objects. This might be for example shader-level effects like clipping, coloring, coordinates modification. The main problem here is a robust [visual] language for building such effects and managing matrix of their appliance to visual objects. One of approaches to GUI modeling that we consider in this work, is visual editing of building blocks of effects, in the form of nodes and connections between them, like in [27].

We see as very useful the feature of adding new types of visual objects. Currently, to add a new visual object, one have to place it in the source code tree of VR-Cinema project. There should be a more flexible way, for example by adding new types of visual objects in Cinema database scope or research project scope.

7. Conclusion

We emphasize the following advantages of the proposed approach for 3D scenes construction. Firstly, the CinemaScience format is extra simple to write. This allows a developer not to be overwhelmed by details like in other 3D formats. Secondly, 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 classes 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 user interactions 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 thank reviewers of the paper for their constructive feedback. We thank experts of the journal "Scientific Visualization" for preparation of the paper for publication. We also thank the mathematicians of the N.N.Krasovsky Institute for the pleasure of jointly solving the presented visualization tasks.

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.

[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] 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.

[8] V. Patsko, A. Fedotov, Three-dimensional reachable set at instant for the Dubins car: Properties of extremal motions, 2020, pp. 1033–1049. URL: http://sector3.imm.uran.ru/ stat/IACAS2020_ThL2T2-03.pdf, 60th Israel Annual Conference on Aerospace Sciences, IACAS 2020.

[9] 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.

[10] A. A. Uspenskii, P. D. Lebedev, On the structure of the singular set of solutions in one class of 3d time-optimal control problems, Vestnik Udmurtskogo Universiteta. Matematika. Mekhanika. Kompyuternye Nauki 31 (2021) 471–486. doi:10.35634/vm210309.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] CinemaScience Composable Image Set extension, https://github.com/cinemascience/cinema/blob/master/specs/dietrich/01/extensions/cis/1.0/cis_specification_v1-0.md, 2021.

[18] H. Abelson, G. J. Sussman, with Julie Sussman, Structure and Interpretation of Computer Programs, 2nd editon ed., MIT Press/McGraw-Hill, Cambridge, 1996.

[19] A. E. Bondarev, On visualization problems in a generalized computational experiment, volume 11, 2019, pp. 156–162. doi:10.26583/sv.11.2.12.

[20] 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.

[21] A. Zakharova, D. Korostelyov, O. Fedonin, Visualization algorithms for multi-criteria alternatives filtering, Scientific Visualization 11 (2019). doi:10.26583/sv.11.4.06.

[22] D. Manakov, Data abstraction models: Sampling (parallel coordinates), filtering, clustering, Scientific Visualization 11 (2019). doi:10.26583/sv.11.1.11.

[23] 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.

[24] P. Vasev, S. Kumkov, E. Shmakov, Extensible scientific visualization system, Scientific Visualization (in Russian) 4 (2012) 64–77. URL: http://sv-journal.org/2012-2/05.php.

[25] 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.

[26] D. Manakov, V. Averbukh, P. Vasev, Visual text as truth subset of the universal space, Scientific Visualization (in Russian) 8 (2016) 38–49. URL: https://sv-journal.org/2016-4/04. php.

[27] D. S. Vyatkin, Visualization of grid data using nodes editor, Ural Federal University / Qualification work for a master's degree (in Russian) (2021). URL: http://www.cv.imm. uran.ru/e/3241774.

A. Online Resources

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