IGScript: An Interaction Grammar for Scientific Data Presentation

Richen Liu
School of CEI/AI, Nanjing Normal University
Nanjing, P.R.China
richen@pku.edu.cn

Min Gao
School of CEI/AI, Nanjing Normal University
P.R.China
19170326@stu.njnu.edu.cn

Shunlong Ye
School of CEI/AI, Nanjing Normal University
P.R.China
19180306@stu.njnu.edu.cn

Jiang Zhang
School of Electronics Engineering and Computer Science, Peking University
Beijing, P.R.China

ABSTRACT

Most of the existing scientific visualizations toward interpretive grammar aim to enhance customizability in either the computation stage or the rendering stage or both, while few approaches focus on the data presentation stage. Besides, most of these approaches leverage the existing components from the general-purpose programming languages (GPLs) instead of developing a standalone compiler, which pose a great challenge about learning curves for the domain experts who have limited knowledge about programming. In this paper, we propose IGScript, a novel script-based interaction grammar tool, to help build scientific data presentation animations for communication. We design a dual-space interface and a compiler which converts natural language-like grammar statements or scripts into a data story animation to make an interactive customization on script-driven data presentations, and then develop a code generator (decompiler) to translate the interactive data exploration animations back into script codes to achieve statement parameters. IGScript makes the presentation animations editable, e.g., it allows to cut, copy, paste, append, or even delete some animation clips. We demonstrate the usability, customizability, and flexibility of IGScript by a user study, four case studies conducted by using four types of commonly-used scientific data, and performance evaluations.

CCS CONCEPTS

• **Human-centered computing** → **Scientific visualization**; **Geographic visualization**; **Systems and tools for interaction design**.

CHI '21, May 8–13, 2021, Yokohama, Japan

ACM ISBN 978-1-4503-8096-6/21/05…$15.00
https://doi.org/10.1145/3411764.3445535

KEYWORDS

data presentation, interaction grammar, scientific visualization

ACM Reference Format:
Richen Liu, Min Gao, Shunlong Ye, and Jiang Zhang. 2021. IGScript: An Interaction Grammar for Scientific Data Presentation. In *CHI Conference on Human Factors in Computing Systems (CHI '21), May 8–13, 2021, Yokohama, Japan*. ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3411764.3445535

1 INTRODUCTION

Interpretive grammars are widely used in the academia and industry to solve different aspects of problems in academic research and industrial engineering [27, 28]. To mitigate the programming difficulties of GPLs while maintaining precise expression in some exact applications, interpretive grammar that presents a higher level of abstraction than low-level interfaces are used. These tools trade generality for ease of use and conciseness, while providing much more flexibility than general interfaces [45]. Interpretive grammar tools consist of interpretive libraries and interpretive languages (or scripts). The distinctions between the languages and libraries are sometimes fuzzy [34]. For example, Protovis [10] is built as libraries, but sometimes described as languages. The interpretive libraries are usually embedded in a host language, i.e., an existing GPL, to leverage existing application program interfaces (APIs). The syntactic features from the original GPL and corresponding language infrastructure could be reused for concrete realization, which might cause some platform dependence issues. The development of an interpretive language could be considered as the creation of a brand-new language [40]. It allows the developers to define their own lexical analyzer and semantic analyzer [53], which is widely acknowledged that the development is tedious due to its independence on existing GPLs.

Most existing scientific visualizations toward interpretive grammars enhance the customizability in either the computation stage [17, 45], the rendering stage [12, 41, 47], or their combination [34]. However, few approaches focus on the stages of interactive data presentations or data story building. We further narrow down the definition of data presentation in this paper to be a series of controls over virtual

cameras. Thus the term "data presentation" in this paper consists of three aspects: (a) defining what important data feature the cameras can see; (b) changing cameras viewpoints to view data from different perspectives and scales; (c) selecting important viewports to present data.

Building data stories and maintaining precise expressions on the presentations are important because of the following reasons. First, achieving precise control when generating gentle and elegant transitions is difficult for people in human-computer interactions. For example, rotating the Earth manually by 10 longitudes per second evenly to generate a data story through recording a demo video for a science talk is highly impossible in a global map-based data visualization. A slight transition in shot changes between regions of interest (ROIs), similar to the style in film photography, is also difficult to achieve. Second, it often takes a long time in trial-and-error explorations which are of little use to get significant cases when generating a scientific storytelling demo through traditional graphical user interfaces (GUIs). Third, supporting some complex application-specific presentations by using traditional GUI interfaces is challenging. We take an example in 3-D box queries in diffusion tensor imaging (DTI) fiber data. Suppose that four query boxes A, B, C, and D could be defined as follows: A is specified to query the ocean pathline clusters (or DTI fiber tracts) passing through the eastern Pacific (or inner capsule), B is for those that pass through Hawaii (or corpus callosum), C is for the northern Pacific (or brain stem), and D is for the southern Pacific (or outer capsule). Given a query such as $(A \cap B) \cup (C \cup D) \cup (\bar{A} \cup C)$, it is difficult to define this query by the traditional interfaces unless enumerating all the box entities and operators by widgets on the traditional GUIs. If we add more query boxes, such as E, F, and G, then the new widgets should be added to the GUI in run time. However, the flexible, legible, and unambiguous expression that defines arbitrary logic combinations of queries is simple to use and edit.

Most of these existing approaches leverage the existing components from the general-purpose programming languages (GPLs) instead of developing a standalone compiler, which pose a great challenge about the steep learning curves for the domain experts who have limited knowledge about programming. According to the feedback from the domain experts, they think most of the current GPL codes are difficult for them to write and edit. They also suggest that it would be better if the identifiers and parameters can be redefined using their technical terms or closer to natural languages. However, a standalone compiler is flexible enough to allow us to customize the identifiers, statement structures and even code styles for different domains. Therefore, we propose **IGScript**, a novel script-based interaction grammar tool based on a light-weight compiler, to help build scientific data presentation for communication. We abstract and summarize the data presentation functions into script grammars. Then, we develop a special-purpose compiler to build natural language-like script codes. This compiler enables to build narrative story for 2-D scalar field data, 3-D scalar field data, vector field data, and DTI data, which are four frequently-used general data types in scientific visualization. In this paper, we mainly address the following issues in the design of IGScript:

Code generator and parameter assignments via a dual-space view: We develop a script code generator (or a decompiler) to translate the interactive scientific data explorations into script codes to achieve statement parameters, and design a compiler to convert natural language-like scripts into presentation story to make a precise customization. We also design a dual-space linked view, which consists of a visualization space view and a coding space view, to provide visual steering and visual feedback for code generations. IGScript enables to cut, copy, paste, append script codes through code editing in coding space, or even delete some trial-and-error codes which are of little use to get new discoveries or significant cases.

General-purpose design: The flexible, legible, and unambiguous script grammar of IGScript supports most general-purpose data presentations, which are summarized and abstracted into four types of functions, i.e., versatile data loading, object transformations, animation controls, and scene switching of the camera. The transitional scene switching across ROIs follows the scheme of script-driven film photography.

Application-specific design: We also design grammars to solve some application-specific data presentation issues in scientific data visualization. It involves clustering-driven camera splitting/merging to support overview-to-details vector field data exploration, origin-destination (OD) [29, 31, 54, 55] query-driven clustering to support the group tracking explorations for vector field data and DTI fiber data, 3-D box queries [1, 14, 33, 52] and their set operations (intersection, union, complement) to query DTI fiber tracts. The application-specific cases in this paper are collected from the above-mentioned literature and the domain requirements.

2 BACKGROUND AND RELATED WORK

Many scientific visualizations toward interpretive grammars focus on the customizations or function definitions in either the rendering stage or the computation stage or both. For example, many of them focus on the rendering stage, and most of which are implemented based on OpenGL Shading Language (GLSL) [12, 41, 47]. Specifically, an octree-based visual composer [44], a network-based visual selector [41], and a shader block combiner [47] are designed to select GLSL shader components at the rendering stage to support volume rendering, respectively. Besides, there are some interpretive grammar approaches to enhance the customization in the computation stage. For example, a novel domain-specific tool named ViSlang [45] is presented to perform volume data processing and querying. A framework named Vivaldi [17] is proposed to build codes from volume rendering to image segmentation by parallel computation. Another tool named Diderot [34] is proposed to bridge the semantic gap between the mathematical computation and how they look in source

codes. Some novel grammar tools like Reactive Vega [49], Vega-Lite [48] and GoTree [35] are designed to enable rapid specification of information visualizations (bar charts, line charts, scalar plots and tree) by using JSON (JavaScript Object Notation) syntax. Besides, we notice that there is a series of approaches using Shape Grammar [32, 37] to define the quadric surface and cuboid shapes of drawn objects (e.g., oil wells, chimneys, buildings, etc.) in order to make a balance between standalone grammars and nested libraries. Overall, few scientific visualizations focus in interactive customizations in data presentation stage.

Furthermore, we have summarized the related interpretive grammar work from different viewpoints according to a recent survey [51]: interpretive grammars can be classified into external interpretive grammars and internal interpretive grammars based on whether the grammar depends on another host language. Many papers classified by the three criteria overlap each other to some extent. We, therefore, review the related work according to the latter two classification criteria briefly. The second classification criterion is to categorize the interpretive grammars into textual interpretive grammars (Tex) [4, 8] and graphical interpretive grammars (Gra) [36, 42] according to the symbols of the libraries/scripts. The third classification criterion classifies the interpretive grammars according to the exact functions like modeling or visual analytics, e.g., DSVL [2, 26], DSML [20, 50], and DSEL [5, 21]. IGScript is more like a DSVL work while it focuses on grammars for interactive data presentation instead of visualization. It enables users to customize data story animations in a dual-view on important data features, data queries, and visual tracking.

2.1 External Interpretive Grammars

External interpretive grammars are mainly implemented on standalone languages, thus most of the external interpretive grammars are script languages instead of libraries. Few external grammar approaches aim to solve visualization problems toward complex platform development due to its independence on existing GPLs [40]. They need to build their own lexical analyzer and semantic analyzer [53]. However, external interpretive grammars allow design freedom [11, 51]. Barringer et al. develop several external interpretive grammars/languages for trace analysis, e.g., EAGLE [6], HAWK [19], RULER [9], and LOGSCOPE [7]. To alleviate the independence problem, Cosentino et al. propose a prototype-tool named DSLit [18]. We can find that almost all of the existing external grammar methods aim to solve the domain-specific engineering problems or satisfy the industry requirements.

2.2 Internal Interpretive Grammars

Internal interpretive grammars are often designed to embed in a host language (i.e., an existing GPL), and they are often implemented as libraries in order to employ the existing functions defined by the GPLs [40]. Most of the existing visualization approaches toward interpretive grammars can be categorized into internal interpretive grammars because they can leverage the existing components from the GPLs. However, selecting an appropriate host language is significant [46] in internal grammar designs. GPLs offer more flexibility for the construction of internal interpretive grammars [25].

The high-level abstractions of grammars [39] are designed by reusing the host language features. The languages like Python [17], Haskell [22], Ruby [28], and C/C++ [3, 27] are seemingly ideal for the host languages of internal interpretive codes. In addition, scripting languages and dynamic languages are also good options [8, 25] due to their syntactic flexibility and straightforward adaption [51].

2.3 Relationships

The relationships between the existing interpretive grammars and the proposed IGScript can be summarized as follows:

First, IGScript exerts the advantages of standalone compiler and script-like grammar. It is independent on the host languages and thus decreases the learning curves of them. The IGScript grammars are simple and can be easy to follow the style of natural language-like grammars, which is of significance to the domain experts who have limited knowledge about programming. Many existing visualization approaches toward interpretive grammars leverage the existing components from the GPLs or host languages instead of developing a standalone compiler.

Second, most of the existing visualization approaches toward interpretive grammars enhance the customizability in either the computation stage or the rendering stage or both of them, few focusing on interactive data presentation stage because most of them focus on defining codes in parallel computation or rendering. However, it is of great significance to customize the interactive data presentations and maintain the precise expression. The design of dual-space view enables users easily to define ROIs and get visual feedback to customize the presentation animations. Besides, the general-purpose design enables users to generate script-driven visual traversals across multiple ROIs and the presentation animation on camera splitting/merging.

Third, the carefully-designed grammars of IGScript make all presentation animations (demo videos) editable because all the exploration steps will be bound and linked to different script codes in the dual-space linked views. It enables users to cut, copy, paste, append the animation videos through script editing in coding space, or even delete some trial-and-error explorations which are of little use to get new discoveries or significant cases.

3 SCRIPT CODE GENERATOR AND GRAMMAR DESIGN

We describe the design goals, design considerations and the design details of IGScript in this section. The design details consist of the designs of a light-weight compiler, a linked dual-space view, general-purpose grammars and application-specific grammars.

3.1 Design Goals and Considerations

The design goals of IGScript are listed as follows:

- G1: help users define ROIs via coarse-grained and fine-grained interactions
- G2: form a presentation animation by recording visual traversals or visual tracking across ROIs
- G3: enable users to edit the animation clips of a data story in a semantic space

First, the ROIs can be defined through a multi-scale strategy, i.e., in a coarse-grained scale and a fine-grained scale (G1). We design a dual-space view, i.e., a visualization space view and a coding space view, to provide visual steering and visual feedback for ROI definitions. Users are allowed to place and move an ROI box in the visualization space by coarse-grained tuning and then adjust its position slightly (fine-grained) in the coding space.

Second, the presentation animations are significant for domain experts to conduct domain experts' discussion, which can be recorded by visual traversals or visual tracking across ROIs (G2). We develop a script code generator to translate the interactive explorations into script codes to achieve statement parameters, and design a compiler to convert natural language-like grammar statements (scripts) into scientific data explorations to make a precise control on interactive explorations.

Third, the grammars should be designed intuitively and easily to be edited by users who have limited experience in programming. All animation clips of the data presentation story can be edited by users in an intuitive interface (G3). The grammars consist of general-purpose grammars (G2) and application-specific grammars (G2).

We focus on four frequently-used general types of scientific data, i.e., 2-D scalar field data, 3-D scalar field data, vector field data, and DTI data. According to the book "The Visualization Handbook" by Charles Hansen and Chris R. Johnson [30] and the tensor-based data type definitions by Tim McGraw [38], the categories of data type (type of attribute data) include 2-D/3D scalar field data (0th order tensors), vector field data (1st order tensors) and tensor field data including DTI data (higher order tensors).

3.2 Script Code Generating Based on Dual-Space Linked Views

We develop a script code generator which consists of a parameter assignment module to translate the interactive data presentations into script codes and then achieve statement parameters (G1). All general data presentations like rotation, translation, scaling, etc., can be translated into script codes by using quaternion matrices, thus the code generator seems more like a decompiler. For the demonstration about the code generator, please refer to the supplementary video of the paper.

Furthermore, in order to generate a data presentation animation with more close-ups to ROIs, where would be some tissues or lesions the users are interested in, it requires to achieve the statement parameters bout ROIs. The parameter assignment module of the code generator consists of linked dual-space views, i.e., an interactive visualization space and a coding space. Actually, parameter assignment module can also compile the generated codes into presentation animations by calling the functions of the compiler because the dual-space views are linked in real-time. The visualization space provides visual steering and visual feedback in real-time, and the script codes in the coding space will also be compiled in real-time. It means all the interactive parameter assignments steered by an interaction device (e.g., mouse) in the visualization space can be automatically translated into script codes in real time according to the grammar, and all the codes edited in the coding space can be realized in the visualization space by the compiler of IGScript in real-time (G3). In this way, users are allowed to place and move an ROI boundary box in the visualization space by coarse-grained tuning and then adjust its position slightly in the coding space.

The grammar of the statement ***defineROI*** in the parameter assignment module can be designed as follows,

```
defineROI(roiID=1, roiName="ROI#01", at(0,0,0),
    size=(0.2,0.2,0.2));
```

The initial coarse-grained position (*at*) and *size* of an ROI can be assigned by mouse dragging and mouse wheel zooming its 3-D boundary box in the visualization space, and the corresponding values in coding space will be changed accordingly (G1). Then, a fine-grained step can be conducted in the coding space to slightly adjust the 3-D position values because it is difficult to use mouse to perform fine-tuning and it just works better in 2-D space (G1). For more details about the parameter assignments such as ROI definition, please refer to the supplementary video of the paper. Once an ROI has been defined in the parameter assignment module, it can be used by the whole IGScript system because it is serialized into a local configuration file. Therefore, ROIs for each scientific data just need to be defined only once.

3.3 General-purpose Grammars

The datasets currently supported by IGScript are some traditional scientific data, including map-based 2-D scalar, 3-D scalar, vector, or diffusion tensor imaging data. The general-purpose interaction grammars of IGScript are summarized and abstracted into four types of data presentation functions, i.e., versatile data loading, object transformations, animation controls, and scene switching of the camera. We describe them in details below (G2).

First, the statement ***load*** is designed like a C++ overloading function to support multiple data types. The four types of scientific data that are currently supported can be loaded by invoking different run-time data loading functions. The overloading design makes people who are not familiar with programming relatively easy to load different types of data. It should be noticed that the sample script codes with all types can be selected from a list before the script editing.

```
load {
    data(string dataFile); map(string mapFile);
```

```
3
4     volumeData(string dataFile);
5     rgbaScheme(string tf);// transfer function
6
7     vectorFieldData(string dataFile);
8
9     dtiData(string dataFile);
10};
```

Second, the object transformations are encapsulated by three transformation identifiers, i.e., ***rotate***, ***scale***, and ***translate*** in order to change the objects when generating presentation animations. The overloading function ***rotate*** and the parameters could be legible and semantic identifiers, such as the *X*-, *Y*- or *Z-axes*, the global *angle* (i.e., 360 degrees), or even a 3-D vector in a general form.

Third, the animation control grammars are designed to control the production of data presentation animations. For example, the statement ***animate*** can be exploited to set a global animation speed when generating a presentation animation. In addition, all the basic interactive presentation animations can be conducted simultaneously by using the code block ***parallel***. All the statements in the code block can be specified by individual duration time ranges.

Fourth, ROI-driven scene switching of the camera is also vital in the general-purpose grammar design, where ROIs are the important feature the users are interested in, e.g. the regions of tissues or the lesions in a medical volume data. The ROI positions can be interactively specified by the statement ***defineROI*** in a dual-space. Then, the defined ROIs can be sequentially viewed in the presentation animation by the statement ***locate***, where the stay time (***duration***) for each ROI and the switch time (***interval***) between two successive ROIs can be further customized. IGScript follows the shot change style in film (documentary) photography. The stay time and switch time can be changed to make transitional scene switching between ROIs as smooth as possible.

```
1rotate {axis=string, angle=string, duration=float
      seconds};
2rotate {axis=(float,float,float), angle=float
      degrees, duration=float seconds};
3translate {to(float,float,float), duration=float
      seconds};
4scale {factor=(float,float,float), duration=float
      seconds};
5animate {speed=string}; // [low, moderate, high]
6parallel { // executed concurrently
7     rotate {axis=(float,float,float), angle=float
      degrees, duration=float seconds};
8     scale {factor=(float,float,float), duration=
      float seconds};
9     translate {to(float,float,float), duration=
      float seconds};
10};
11
12defineROI(roiID=int, roiName=string, at(float,
      float,float), size=(float,float,float));
13locate {
14     roiArray=string[roiName#1,roiName#2,roiName
      #3,...],
15     foreach {
16          duration=float seconds,
17          interval=float seconds
18     }
19};
```

The parameters of most statements in IGScript can be visually assigned in a dual space with two linked views. Examples include the position parameters ***to*** of ***translate***, the position parameters ***at*** of ***defineROI***, the scaling parameters ***factor*** of ***scale***, etc. For more details about the parameter assignment module in the code generator, please refer to Section 3.2.

3.4 Application-specific Grammars

We design some application-specific grammars to solve some classic data presentation issues in vector field data visualization and DTI data visualization (G2). Examples include clustering analysis and OD query [29, 31, 54, 55] for vector field visualization, 3-D box queries and their arbitrary logic combinations for DTI fiber visualization.

Streamline, pathline, and streakline are three traditional vector field visualization approaches. Clustering analysis and OD query that can be used to reduce visual clutter and support further interactive explorations are two frequently-used techniques exploited in line-based visualizations. These techniques can be used to group lines, reduce visual clutter or conduct source destination analysis. If we take pathline as an example, IGScript enables to generate presentation animations for pathlines in an overview-to-details way. Specifically, IGScript enables to split the tracking camera into N lenses to generate a group of comparative tracking animations for the major clusters by ***trace***. There are several ***mode***s for group tracking. In ***realtime*** mode, the number of tracking cameras equals to the number of clusters output by clustering algorithm. In both ***origin*** and ***destination*** mode, the number of tracking cameras N equals to the number of clusters corresponding to the top N eigenvalues of the clustering algorithm. It can be initially specified by users via the statement ***colorOfCenterPathline*** because different clusters will be rendered in different colors. The two modes mean to cluster the original seed positions and the destination positions of all traced pathlines, respectively. It should be noted that the clustering algorithms and dimension reduction algorithms can be customized in the IGScript codes according to different datasets and different application scenarios by ***clusteringAlg***. The ***lifeTime*** of all the pathlines could be assigned to restrict the length of the pathlines. Furthermore, the rendering style of all lines in IGScript can be assigned to be ***lineStyle*** or ***tubeStyle***.

Alternatively, users can specify the camera splitting/merging scheme manually by ***halfMergeSplit*** in an "Overview + Details" scheme, after getting representative pathlines and OD query clusters. It enables users to split cameras into two halves to see more details of the clusters, or merge the two halves into one to see the overview, where the splitting/merging strategy can be customized by the parameters of ***timeSlots***. The data presentation animations can be customized as overview-to-details, overview-to-details-to-overview-..., cyclically. For example, if there are N ($N = 4$)

representative pathlines are recommended, the parameter list "[overview3months,5,9,0,3]" is meant to generate a presentation animation starting from an overview of all the pathlines for their first 3 months' pathline tracing. In the following steps, one camera is split into two for the next 5 months' tracing, two split into four for the next 9 months' tracing, four merge into two for the next 0 month's tracing due to the maximum camera number is restricted by $N = 4$ (skip this one), and finally merge into one (overview) for the final 3 months' tracing.

```
trace {
    mode=string, //[destination, origin, realtime]
    clusteringAlg=[dbscan, kmeans, pca, ...]
    lifeTime=float,//lifetime of traced fieldlines
    colorOfCenterPathline=[c1, c2, ...]
};
halfMergeSplit { // overview-to-details
    timeSlots=[overview,t1,timeUnits, t2, t3, ...]
}
lineStyle {color=colorL, width=float};
tubeStyle {color=colorT, thickness=float};
```

In DTI fiber data visualization, 3-D box queries and their arbitrary logic combinations (e.g., intersection, union, and complement) are significant to explore DTI data to satisfy users' kinds of complex query requirements [14, 16]. The fiber tracts passing through the 3-D query boxes can be highlighted [14]. Logic combinations of query boxes aim to reduce visual clutter and help obtain a deep insight into the analysis of DTI fiber data [16]. The statement ***placeANewBox*** is also designed to be an overloading function. The position of the newly placed box can be assigned according to the ROIs defined by ***defineROI*** or specified by the parameter assignment module of code generator (refer to Section 3.2).

In addition, all 3-D query boxes can be moved by ***moveABox***, and their sizes can be changed by ***scaleABox***. The animation speeds specified by the ***duration*** time of the two statements can be increased to obtain additional details when an ROI box is moved or its size is changed. However, this step is tedious and time-consuming in the traditional method [14] because it requires at least three control widgets (e.g., sliders designed by Chen et al. [14]) on GUI to change the 3-D position, if users want to obtain precise control. The parameters can be determined by the above-mentioned parameter assignment module of the code generator.

More importantly, arbitrary logic combinations of multiple query boxes can be flexibly customized by the query expression in ***show (queryExpr)*** due to the legible and unambiguous script grammar. Specifying all the logic operators between the query boxes by the traditional GUI controls or the traditional interaction devices is difficult for people because the number of the query boxes is undetermined and the size of the combinatorial space (logic combinations of intersection, union, and complement) increases greatly as the number of query boxes increases. For example, if A, B, and C represent three ROIs in white matter, grey matter, and corpus callosum, respectively, then simple and legible query expressions, namely, ***show*****(A∩(B∪C))**, ***show*****((A∪B)∩C)** and ***show*****(A∩not(B∪C)**, are easy to edit in IGScript. However, designing GUI controls or interaction to customize an arbitrary logic combination over A, B, and C through the traditional GUI designs [14, 16] is difficult. Furthermore, a new control widget should be added in the GUI in runtime if users add a new box D, which is unscalable for the GUI designs and the combinatorial number is greatly increased. Additionally, users can use the code block ***with*** to set a local value of color and opacity within its statement scope.

For more detailed information about the compiler structure and the application-specific algorithms, please refer to **Appendix A**.

```
// overloading "placeANewBox": assigned by an ROI
placeANewBox {boxID(string), at(roiName=string),
    color=c, alpha=float};
// overloading "placeANewBox"
placeANewBox {boxID(string), at(float,float,float)
    , size=(float,float,float), color=c, alpha=
    float};
moveABox {boxID(string), to(float,float,float),
    duration=float seconds};
scaleABox {boxID(string), factor(float,float,float
    ), duration=float seconds};

// code block "with": set local color and opacity
with(color=c, alpha=float) {
    show (queryExpr); //e.g., show(not(A∩B)∪(C∩D))
    pause (float seconds);
    show (queryExpr); //e.g., show((A∩C)∪(B∩C))
};
```

4 IMPLEMENTATION

Syntax checking of the compiler, expression-based DTI fiber queries, clustering-driven camera tracking, and OD queries for vector field visualization pose several implementation challenges. The rendering component of IGScript is developed on GPU rendering and OpenGL libraries. For example, the rendering parts of the general-purpose data presentations are mainly implemented on OpenGL libraries. However, the volume rendering and the lighting effect are implemented based on GPU rendering, which makes the presentation animation smooth. The rendering component is built upon compiler component. The compiled results will be transferred to the rendering component to generate presentation animations. The rendering system is designed similar to a library design, which can be changed and added more rendering functions as libraries.

4.1 Special-purpose Compiler and Expression-Based Logical Combination Query

The statements and their parameter values are interpreted by the carefully designed compiler of IGScript. In the compiler, a trie tree is exploited to perform lexical analysis, the input of which are the identifiers and keywords extracted from the script codes, as shown in Figure 1. Fail pointer is further employed to improve the trie node construction and accelerate the parsing process. Then, the lexical spelling is checked,

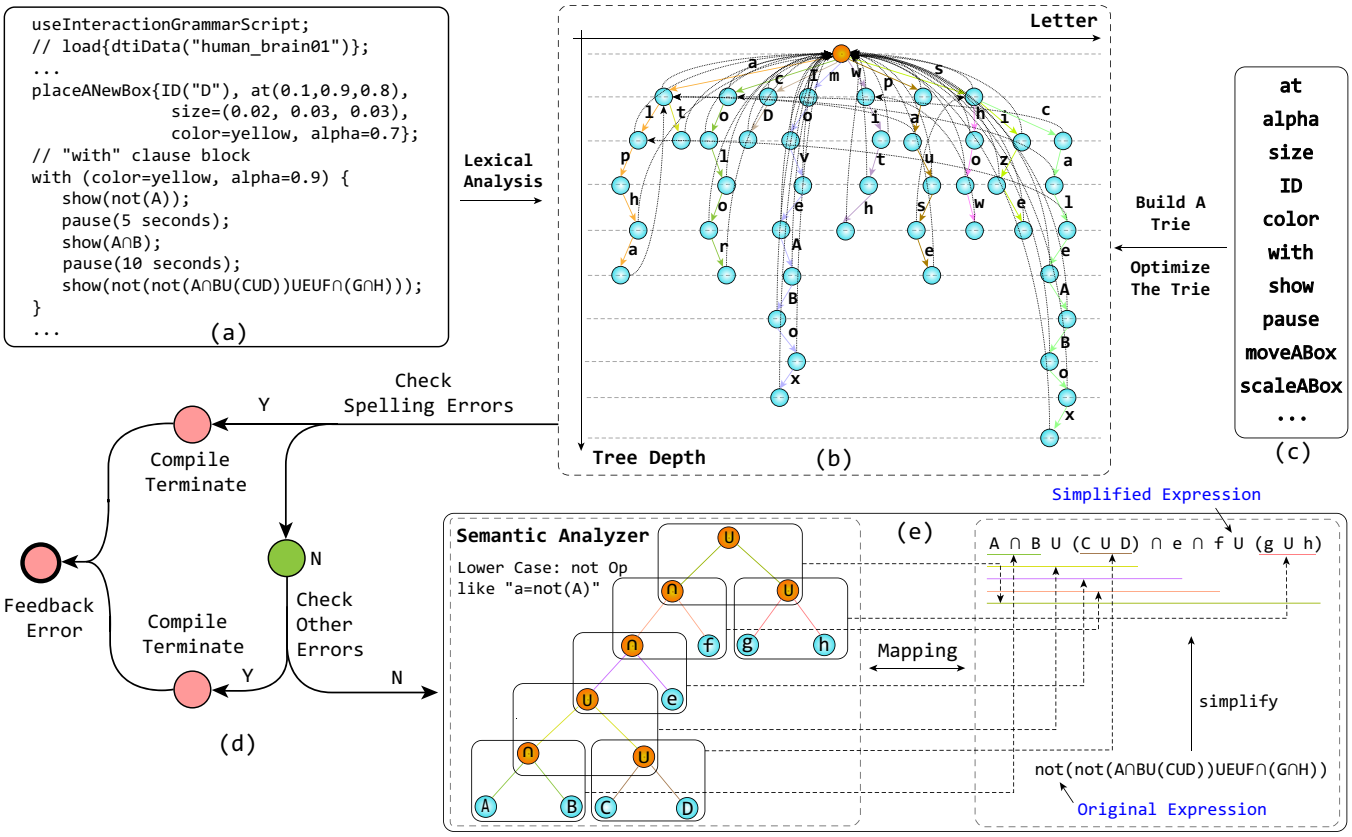


Figure 1: A technical design diagram of the compiler of IGScript. (a) Script codes of the presentation animation generation for DTI fiber data. (b) A trie tree is constructed for lexical analysis. (c) The extracted identifiers and keywords to be analyzed by the trie tree. (d) The state diagram of the spelling checking parser. (e) A binary tree is built to analyze the simplified query expression semantically.

which involves parenthesis checking, keyword checking, identifier checking, variable checking, and function checking, and the error messages are output to help fix bugs. Otherwise, in the next step, semantic analysis is performed. We take DTI fiber data visualization as an example. One of the most challenging issues in semantic analysis is to parse the query expression. A binary tree is built to parse the logical combination query expression. The fiber tracts passing through the query box A (i.e., a 3-D ROI box) are saved into a data structure set and highlighted in the visualization space when the expression is **show(A)**. Subsequently, the original expression is simplified if it includes a complement operator (**not**). The lower case character indicates the complement operator over a set, e.g., **a=not(A)**. Thus, the binary tree is constructed according to the simplified expression. The output of the semantic analyzer is a set of fiber tracts marked as **true** after binary tree analysis. The time complexity of the compiler is $O(M)$, where M is the code length, including the number of keywords, identifiers, variables, and values.

4.2 Clustering-driven Camera Splitting and Merging

The second challenging issue in the implementation includes clustering-driven camera tracking for vector field data presentations and DTI fiber data presentations. Clustering analysis and OD query are two frequently used techniques in line-based visualization for vector field data. IGScript supports clustering-driven camera splitting and merging to generate a tracking animation for the line data presentations. If we take a pathline as an example, the count (N) of tracking camera lenses can be assigned as the count (C) of clusters or specified by the script (i.e., the color count) subject to $N \leq C$, indicating that the N cameras track the top N clusters from all C clusters. It means we should recommend N representative pathlines for the top N clusters.

The major goal of the representative pathline recommendations is to make their inter-distances as large as possible, which ensure that as many pathline points (with current time-step) as possible to appear in any tracking camera, if the number N has been provided in the script by ***colorOfCenterPathline***. The major steps (Figure 2) of the camera splitting and merging scheme can be summarized as follows.

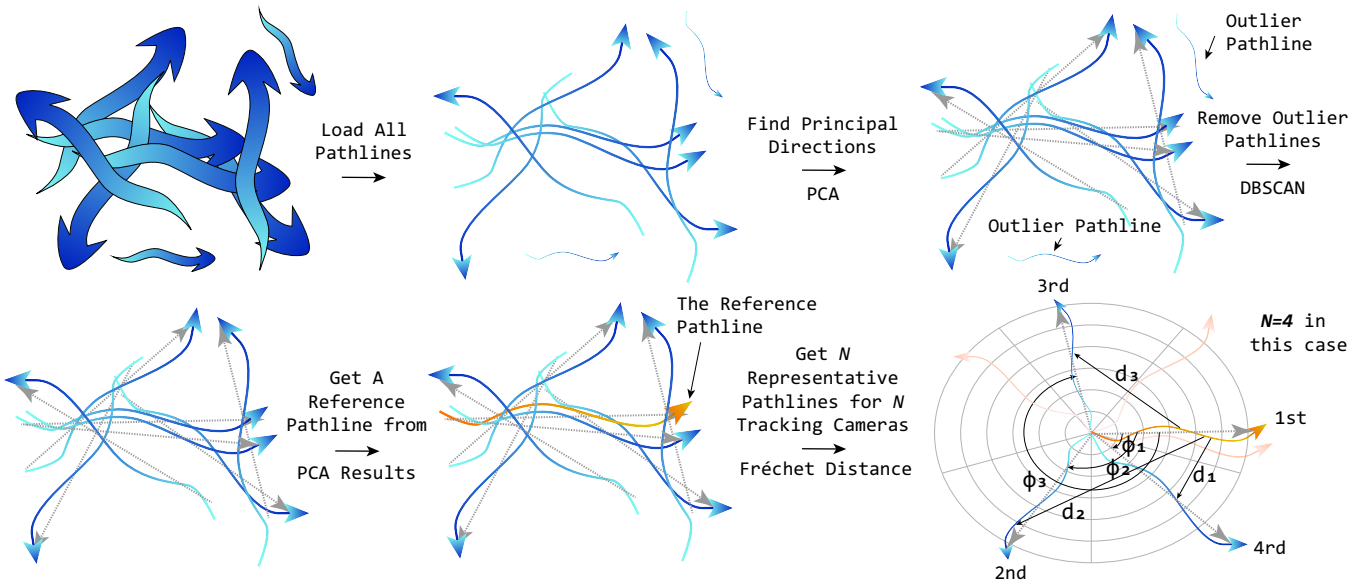


Figure 2: The major steps of clustering-driven camera tracking and OD query-based camera tracking. The principal directions of all loaded pathlines are found by PCA [43]. The outlier pathlines can be removed by DBSCAN [23], and a reference pathline is selected, then the Frechet distances [24] between the reference pathline and all the candidate pathlines are computed to recommend N (N=4 in this case) representative pathlines to make the inter-distances of clusters as large as possible.

Step 01: Use principal components analysis (PCA) [43] to find C principal directions from all pathlines. All the pathlines are discretized and sampled to form a point cloud set (P). A longer pathline indicates that more points are sampled into (P).

Step 02: Remove the outlier pathlines. Density-based spatial clustering of applications with noise (DBSCAN [23]) is exploited to detect the outlier pathlines from the point cloud P. DBSCAN is often used to cluster point data in accordance with their spatial densities, which can detect the outlier points in a large-scale point cloud.

Step 03: Get the reference pathline from the PCA results. The one projected to the first principal component of PCA with the longest length in the projection space is a good option to be a reference pathline because it makes the inter-distance of clusters as large as possible.

Step 04: Recommend N representative pathlines, which correspond to the N camera lenses to track the major directions of all pathlines. Then get a set of $w = \phi_i + kd_j$ and sort them into a list, where k is a constant value, ϕ_i are the polar angles between the reference pathline and all the candidate pathlines, and d_j are their discrete Frechet distances [24], as shown in Figure 2 (bottom-right). Then, the N items of equal distance in the above-mentioned sort list are the values of the corresponding N recommended pathlines. It ensures as many pathlines as possible appear in any one tracking camera. Frechet distance is a classic solution to measure the similarities between trajectories. The Frechet distance d_j can be computed as follows:

Suppose the first pathline P with its length X and the second pathline Q with its length Y. Then we use $\alpha(t)$ and $\beta(t)$ to describe the points of P and Q, respectively. Thus, $\alpha(0) = 0, \alpha(1) = X, \beta(0) = 0, \beta(0) = Y$. We define $P(\alpha(t))$ and $Q(\beta(t))$ to represent the spatial positions of the two pathlines at timestep t, respectively. We discretize Frechet distance in Equation (1) to compute the distances of discrete pathlines.

$$\delta_F(P,Q) = \min_{\alpha,\beta}\{\max_{t\in[0,1]} d(P(\alpha(t)), Q(\beta(t)))\} \tag{1}$$

Since the pathline is composed of discrete points, the above formula is considered to be extended to the discrete Frechet distance.

We discretize the above two trajectories, and assume that the curve P is composed of p locus points, the curve Q is composed of q locus points, and the length of the discretized mapping sequence is r. Use $\sigma(P)$ and $\sigma(Q)$ to set the order of the two track points, which means $\sigma(P) = (u_1, ..., u_p)$ and $\sigma(Q) = (v_1, ..., v_q)$, respectively. Meanwhile, we can get the following sequence points for L

$$\begin{gathered}(u_{a_1}, v_{b_1}), (u_{a_2}, v_{b_2}), ..., (u_{a_r}, v_{b_r}),\\ a_1 = 1, b_1 = 1, a_r = p, b_r = q\end{gathered} \tag{2}$$

The length of sequence pairs between P and Q is defined as $||L||$, which is the largest Euclidean distance in each pair.

$$||L|| = \max_{i=1,...,r} d(u_{a_i}, v_{b_i}) \tag{3}$$

The final Frechet distance of (P, Q) can be computed by Equation (4), where $||L||$ is the largest Euclidean distance for a sequence pair in P and Q.

$$\delta_{dF}(P,Q) = \min ||L|| \tag{4}$$

4.3 OD Query-Driven Camera Tracking

The third major issue in the implementation is OD query-driven camera tracking. After the N representative pathlines are recommended in the camera splitting/merging step, the next step is camera tracking. The virtual cameras of the animation track the corresponding representative pathlines. All the pathlines are classified into the N categories (one category for one representative pathline). Three classification strategies are available. The number of splitting/merging cameras is the number of clustered ***origin*** points (seed points) or ***destination*** points.

In the camera splitting/merging and camera tracking steps, the clustering algorithms applied in discovering the principal directions (PCA by default), removing the outlier pathlines (DBSCAN by default), and clustering pathlines (DBSCAN by default) **could be customized in the script codes by the identifier *clusteringAlg*** in accordance with different scenarios. The optional algorithms could be PCA, DBSCAN, or K-means, which could be encapsulated as the external dependency libraries of IGScript.

5 USER STUDY

We conduct a user study to examine if non-experts and domain experts with limited programming skills could create their desired data presentation animations by IGScript. In the user study, we aim to evaluate IGScript regarding three aspects, which are coincident with the above-mentioned design goals G1-G3: (a) whether users can define ROI easily (G1); (b) whether the generated animations are what users want (G2); (c) whether it is allowed to edit the small clips in an animation (G3). We will describe the considerations for all questionnaire questions in Section 5.3.

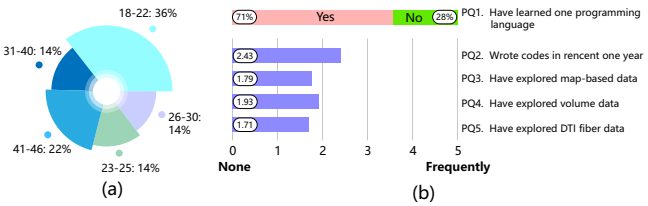


Figure 3: (a) Age distribution of participants. (b) Pre-study: domain experts have limited experience in programming but most of them have learned one coding language.

5.1 Participants and Settings

We recruit 14 participants (8 males, ages: 18 to 46, average: 29), who are 8 doctors from different hospitals and 6 non-expert novice users with different majors. We choose doctors as participants because they are highly relevant to the biomedical data (two volume datasets, two DTI fiber datasets). They are the largest potential users of IGScript. Six non-expert users have different domain knowledge and have limited knowledge in programming. Figure 3 (a) illustrates age distribution of the participants. We pre-screen the participants to ensure that most of them have ever learned one programming language, as shown in Figure 3 (b). The gifts prepared for the participants are independent of their performance. Besides, they are not the co-authors of the paper.

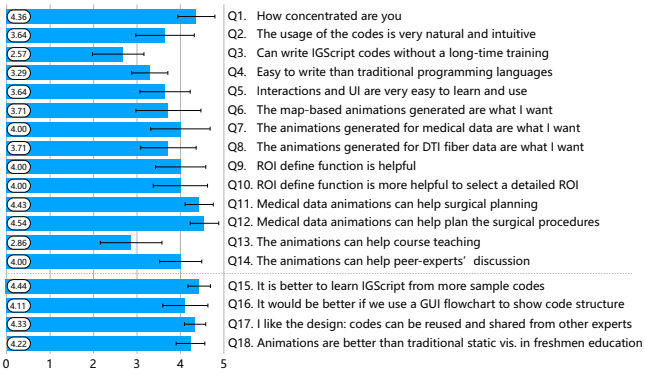


Figure 4: Post-study: questionnaire results. Most of participants react positively to IGScript.

Most of the IGScript's usage scenarios in the paper are medical data presentations, including the medical volume data, DTI fiber data, and code generator evaluation on DTI fiber data. Participants need to write their own script codes by IGScript during their explorations. All the participants have their identification numbers (P1-P14).

5.2 Procedures of the User Study

Participants fill out a consent form and personal information, accomplish five tasks with IGScript, including a free exploration by using IGScript, and then conclude with a post-study survey and an interview about the feedback and suggestions.

Demonstration and Training (20 mins). The investigators hand out manuals related to data and scripts and introduced scientific visualization to participants. The syntax and usage of IGScript and different data forms that IGScript used are also introduced. Participants need to explore the data we provided. They could request guidance and assistance when necessary. Participants are encouraged to explore freely until they feel confident about the usage and the user study.

Replications (25 mins). The investigators show tasks to be replicated by participants. In this part, participants need to write scripts to reproduce carbon emission data presentation animation, volume data presentation animation, diffusion tensor imaging data presentation animation and pathline data presentation animation. Most available features of IGScript are covered in the scripts that required replications.

Free Exploration (10 mins). Next, participants are free to explore the data used in replications to test the dual-space view to evaluate the ROI definition functions. The presentation process can be recorded by the code generator and the corresponding presentation scripts are generated automatically. If an area caught the attention of the participant, he could save the region as a custom ROI for a second loading.

Post-study Survey and Interview (15 mins). We use a five-point Likert scale (1: strongly disagree, 5:strongly agree) in this user study. The questions in the questionnaire relate to involvement (Q1), usability (Q2-Q5), correctness (Q6-Q8), practicability of ROI (Q9-Q10) and effect of IGScript (Q11-Q14). Besides, the investigators also collect some comments and suggestions from the participants, which are about the most impressive task and any suggestions of IGScript.

5.3 User Study Results

All participants complete the tasks in about 25 minutes. After that, the investigators have a conversation with participants to get feedback on scripts and tools. Figure 4 shows the questionnaire and the average score of each question. We will analyze questionnaire scores and qualitative evaluation from involvement, usability, correctness, practicability of ROI and effectiveness of IGScript.

Usability. Since the target audiences are domain experts and the public who have limited knowledge about programming, IGScript should be easy to learn and use. Based on the evaluation results of Q2 ($\mu = 3.64$, 95% $CI = [2.97, 4.32]$, the evaluation for G1), IGScript is recognized for its usability. In particular, regarding Q3 ($\mu = 2.57$, 95% $CI = [1.99, 3.16]$, G2 and G3), participants react with neutral responses in the user study evaluation. We find the domain experts tend to give lower scores on Q3. Thus we revisit some of participants again and they consider it better if we provide more sample codes.

For Q4 ($\mu = 3.29$, 95% $CI = [2.87, 3.70]$, G2 and G3) and Q5 ($\mu = 3.64$, 95% $CI = [3.06, 4.22]$, G1), users reacted positively to the comprehensible scripts. When the investigators show our method, "it seems to be much easier to get precise 3-D positions of an ROI by the dual-space linked views" (P6, P7, P8, P9, P10, P11 and P13). Most of the participants find IGScript codes are more intuitive (Q2) and easier to write than traditional languages (Q4 and Q5), because they are closer to natural languages supported by the standalone compiler.

Correctness. In the correctness questions, a lot of positive feedback has been collected. For example, Q6 ($\mu = 3.71$, 95% $CI = [2.96, 4.47]$, G2) is to evaluate the map-based data presentation, Q7 ($\mu = 4.00$, 95% $CI = [3.31, 4.69]$, G2) for the medical volume data presentation, and Q8 ($\mu = 3.71$, 95% $CI = [3.07, 4.35]$, G2) for the DTI data presentation. However, seven participants indicate that they have difficulty in entering perfectly correct scripts at the beginning. In this regard, we guide them to reference some sample codes for each data presentation.

Practicability of ROI definitions. Regarding ROI definitions, most of the participants describe the design is helpful in their explorations. We can see their positive comments of ROI definitions: "It makes a good contribution to better explore medical data by using ROI definitions" (P1). They are also interested in the function of automatically generating code to record interactive explorations. P7 declares that this feature is friendly to novice students. At first, some participants are concerned about the tasks because they have never explored in this way (P7). We note that participants know the purpose of the ROI definitions and the ease of using ROI as they develop a deeper understanding of IGScript. The questions Q9 ($\mu = 4.00$, 95% $CI = [3.42, 4.58]$) and Q10 ($\mu = 4.00$, 95% $CI = [3.37, 4.63]$) are also designed to evaluate G1.

Effectiveness of IGScript. Participants have different opinions for the effectiveness of IGScript. However, they still gave positive feedback on Q11 ($\mu = 4.43$, 95% $CI = [4.10, 4.75]$, G2) and Q12 ($\mu = 4.54$, 95% $CI = [4.21, 4.87]$, G2). "IGScript is very useful in popularization of science. It helps to understand the structure of the lesion while having a good sense of substitution. In medical diagnosis, it offers an approach to record the exploration route" (P8). Eight participants think that IGScript can help teaching to some extent (Q13 ($\mu = 2.86$, 95% $CI = [2.15, 3.57]$, G2)), and 11 participants think IGScript can help peer-experts' discussion (Q14 ($\mu = 4.00$, 95% $CI = [3.51, 4.49]$, G2 and G3)). Besides, "the generated animation of the hand data is impressive" (P6), and "the lung data animation clearly reflects the structure of the lesion, which is easy to understand" (P11).

Involvement. In response to Q1 ($\mu = 4.36$, 95% $CI = [3.93, 4.78]$), almost all participants think they feel quite concentrated on the study, and they all agree with the design philosophy of IGScript.

Suggestions and User Response for Improvements. In addition, some constructive suggestions are made by two participants, which are listed as follows:

P5: "I think IGScript is applicable to science popularization and teaching. It can go deep into surgical navigation, especially the definition of ROIs". P8: "The encoding can be organized in a fill-in-the-blank scheme. Some design about bug fixing could be introduced to further improve the usability when script codes are too long". Given the user feedbacks, we consider optimizing the code editor in the future to avoid having users memorizing the statement identifiers as much as possible.

In particular, we find the domain experts tend to give lower scores on Q3. Thus we revisit some of participants and interview more experts after the first user study. We find that they consider it would be better if we provide more sample codes (Q15, $\mu = 4.44$, 95% $CI = [4.18, 4.70]$). Besides, they think it would be better if the identifiers and parameters of codes could be redefined using their technical terms or closer to natural languages. Actually, the compiler of IGScript makes it flexible for us to redefine the identifiers and statement structures for different domains. We will discuss the future work about domain-specific grammars in Section 7.

We have additionally interviewed another 12 non-experts with different majors, who also have little knowledge about programming. Most of participants like the IGScript design because it enables them to customize the data presentation codes that can be reused and shared by other experts (Q17). They also think the presentation animations generated by

IGScript are much better than the traditional visualizations in freshmen education (Q18). Furthermore, most of them suggest us to design a GUI-based flow chart with multiple visual code blocks in the coding space (Q16). Users can edit the code blocks by clicking them and they just need to edit the editable parameter values in the coding space. They think it would be much better to conceal some currently unchanged codes. All potential extensions of IGScript will be discussed in the future work part in Section 7.

6 DEMONSTRATIONS AND RESULTS

We demonstrate the usability, customizability, and flexibility of IGScript by several presentation examples about the traditional scientific data such as 2-D scalar field data, 3-D scalar field data, vector field data, and diffusion tensor imaging fiber data.

Furthermore, all the application-specific cases are designed according to the domain experts' requirements. We find that visual traversals across ROIs can help peer-experts discussion (Q14) and surgical planning for doctors for the three medical data (Q11 and Q12). In addition, OD flow query [29, 31, 54, 55] and 3D box queries [1, 14, 33, 52] for DTI data are also requirements of domain experts.

All the animation demonstrations could be found in the **supplementary video** of the submission.

6.1 2-D Scalar Field Data Exploration

The presentation example on 2-D scalar field data in this paper is conducted on a map-based visualization. The dataset used in this case includes carbon emission data, which are observation data collected by satellites. We conduct two experiments for the 2-D scalar field data presentations.

In the first case, a script-driven illustrative animation video is generated by a simple and legible statement ***rotate***. Figure 5 (a) shows only two snapshots of the animation. The Earth is rotated along the *axis Y* (i.e., the Earth's axis). The script is flexible enough to change the parameter values into a general form by using an overloading function of ***rotate*** or even a collection of basic interaction by the code block ***parallel***.

In the second case, an illustrative animation that has several transitional scenes switching across multiple ROIs is generated. The five ROIs used in this case are defined by the statement ***defineROI*** (G1). Then, each ROI can be located one by one by using ***locate***. All the ROIs need to be defined only once through the code generator (Section 3.2), and then all defined ROIs for each dataset are saved to a local file, as shown in Figure 5 (b) (G1).

A challenging issue of the traditional demo animation production is that achieving precise control to generate gentle and elegant transitions is difficult. For example, rotating the Earth by 10 longitudes per second evenly around the Earth's axis is highly impossible for people, because it is hard for people to achieve a precise rotation angle by mouse within a given time. Besides, making a slight transition in shot changes between ROIs, such as the style in film photography in the second case, is also difficult.

Furthermore, the data presentation steps of the traditional demo animation production could not be changed once it is rendered to a video file. However, IGScript makes the data presentation steps editable (cut, copy, paste, delete, append, stitching, etc.) (G3). In the second case, for example, if a user such as a climatologist is going to give a popularization of science talk (or peer-expert's discussion) on climate change, and we note that some of the audiences may be interested in the carbon emission in the North Pole and the South Pole, editing the script codes to insert two seamless short animation clips into the existing animation. It allows to output a new demo video easily and efficiently by IGScript without changing the scientific visualization codes. The two corresponding ROIs ("NorthPole" and "SouthPole") could be flexibly defined by the code generator of IGScript (G3). The demonstration of this process could be found in the supplementary video of the paper. Nevertheless, it requires to re-record the whole demo video by the traditional demo animation production to satisfy the above-mentioned requirement.

6.2 3-D Scalar Field Data Exploration

A script-driven illustrative animation for volume (3-D scalar field) data presentation can also be generated by IGScript. In this case, we use two volume datasets (human lung and hand) to demonstrate its usability and customizability.

Two animations for the two datasets are generated by IGScript, some snapshots of which are shown in Figure 6. For the human lung, the lesions around lungs are illustrated in an overview-to-details way. For the lesions on the left lung, for example, the animations are generated from the overview to the details (the left front view and right front view), as shown in Figure 6 (a) (G2). A potential application of this case can be a talk that spreads knowledge about how coronavirus disease 2019 (COVID-19) infects a lung. It can be found that it is easy for users to customize the presentation steps by IGScript in this scenario, as shown in the supplementary video of the paper.

For the second dataset (Figure 6 (b)), the overview-to-details animation can also be generated by similar script codes. The transitional scene switch follows the shot change style in film photography, and the stay time and switch time can be changed to make the transitional scene switching between ROIs as smooth as possible (G2). Similarly, the script codes are easy to edit, e.g., to cut, copy, paste, delete, add new ROIs, or remove some defined ROIs for different application scenarios (G3).

6.3 Vector Field Data Exploration

Clustering analysis and OD query are two important techniques used in line-based visualization for vector field data. These approaches can be employed to reduce visual clutter and support some further analysis such as OD analysis. The clustering-driven camera splitting and tracking can be implemented by the statement ***trace*** by specifying the ***mode*** to be ***realtime*** in the camera tracking step (G2). The value

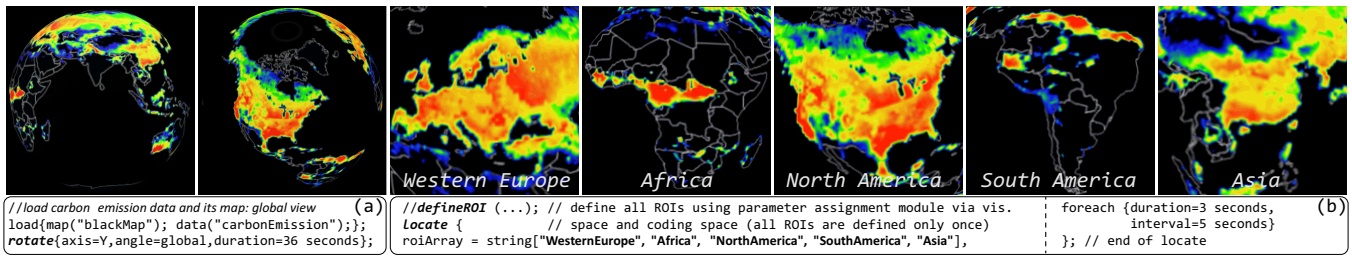


Figure 5: Animation snapshots of overview-to-details presentation for 2-D scalar field data: (a) (overview) a 36-sec presentation animation will be generated by the statement *rotate*. (b) (details) a presentation animation with transitional scene switching across multiple ROIs is generated by the statements *defineROI* and *locate*.

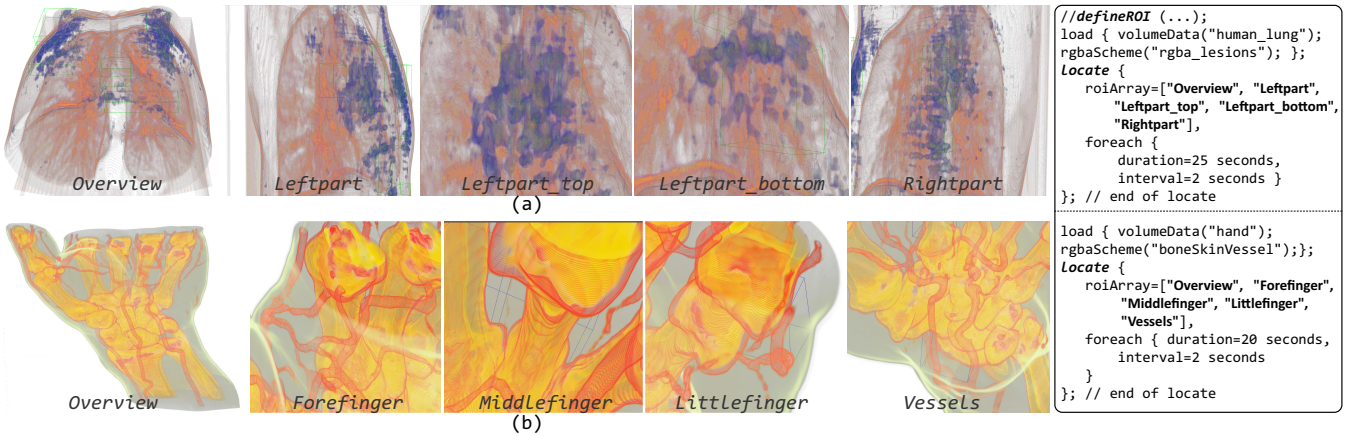


Figure 6: Snapshots of overview-to-details animations for the dataset lung (a) and the dataset hand (b). The transitional scene switching follows the shot change style in film photography.

of N could be either determined by the number of clusters C, e.g., $N = 0.8 \times C$, or assigned in the codes by ***colorOfCenterPathline***.

Alternatively, users can specify the camera splitting/merging scheme manually by ***halfMergeSplit*** after getting the representative pathlines and OD query clusters. Figure 7 (a) and Figure 7 (b) are the four snapshots of the two tracking animations for origin points clustering and destination points clustering, respectively. The camera spliting/merging strategy is customized by ***halfMergeSplit*** (G2). The parameter list "[overview3days,5,9,0,3]" means to generate a presentation animation starting from an overview of all the pathlines for their first 3 days' pathline tracing. In the following steps, one camera is split into two for the next 5 days' tracing, two is split into four for the next 9 days' tracing, four is merged into two for the next 0 day's tracing due to the maximum camera number is restricted by $N = 4$ (skip this one), and finally merged into one (overview) for the final 3 months' tracing.

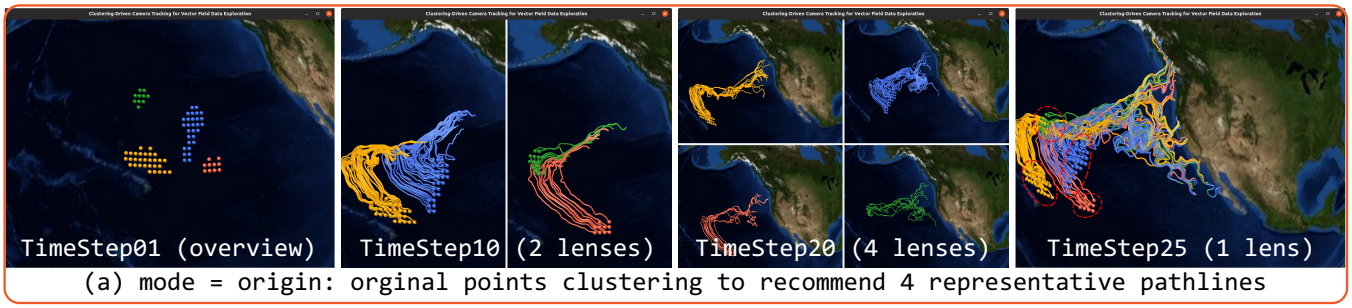
The clustering and principal direction extraction algorithms applied in the camera splitting step and the camera tracking step can be customized in the script codes by the identifier ***clusteringAlg***. The optional algorithms could be PCA, DBSCAN, or K-means.

6.4 Diffusion Tensor Imaging Data Exploration

DTI is a technique that measures the direction of water diffusion in biological tissues. The characteristics of water diffusion in biological structures (e.g., brain, heart, etc.) can be mathematically summarized by a diffusion tensor field [14]. A DTI dataset can be represented with a set of fiber tracts or 3-D pathways. In this case, we use three DTI fiber datasets (from open data [13]), i.e., two human brain datasets, and one pig heart dataset.

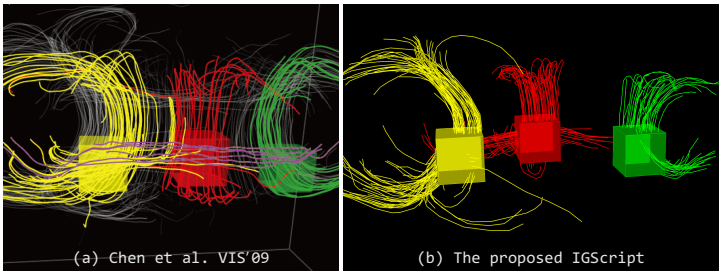
Chen et al. developed a novel interface [13, 14] and a series of novel techniques [15, 16] to visualize DTI fiber data. IGScript can be easily used to reproduce one of the results generated by the work [13, 14] (Figure 8 (a)). It just needs to write three simple IGScript codes by using ***placeANewBox*** to show the individual query results in different colors, as shown in Figure 8 (b). It should be noted that IGScript not only allows to reproduce the static image result (Figure 8 (a)), but also enables to make the static image **animated** without changing the visualization codes by adding a statement ***animate*** (G2). The demonstration could be found in the supplementary video.

More importantly, arbitrary logic combinations of multiple query boxes can be easily defined by the query expression in ***show (queryExpr)*** due to the flexible, legible, and unambiguous script grammar of IGScript. The query is difficult to




```
load {vectorFieldData("ocean.dat")};
animate {speed=moderate};
trace { mode=origin, //or destination
    clusteringAlg=[pca, dbscan],
    lifeTime = 20 days,
    colorsOfCenterPathline
      = [yellow, blue, red, green]
}; //4 colors means 4 lenses (max)
halfMergeSplit {
  timeSlots
      =[overview3days,5,9,0,3] };
//manually cyclic camera splitting
```

Figure 7: Animation snapshots with four time-steps (#01, #10, #20, #25) for the ocean data presentation. The camera splitting/merging strategy is customized by *halfMergeSplit* in the two animations. We select the top N ($N = 4$) representative pathlines in this case study for both the origin clustering (a) and the destination clustering (b) for better comparison.




```
// define all ROIs in parameter assignment module via visualization
// space and coding space, which will be invoked only once for each ROI
// defineROI (...); // defined ROIs are global variables

load {dtiData("human_brain01")};

lineStyle {color=green, width=1.0};

placeANewBox {boxID("A"), at("ROI#HB11"), color=green, alpha=0.5};
placeANewBox {boxID("B"), at("ROI#HB12"), color=red, alpha=0.6};
placeANewBox {boxID("C"), at("ROI#HB13"), color=yellow, alpha=0.6};
```

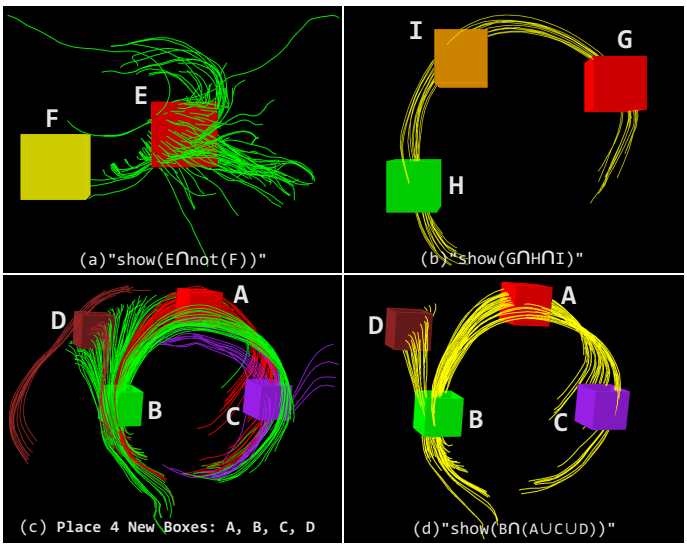
Figure 8: The case of DTI fiber data and its IGScript codes. It is easy to use IGScript to reproduce a similar result to the work [14] without changing the visualization and rendering codes. (a) The ROI query result generated by the work (Chen et al. [14]). (b) Three query boxes are placed at the positions of the defined ROIs to query DTI fibers individually.

achieve using the traditional designs [13, 14] or the traditional GUI interfaces, as mentioned in Section 3.4. For example, a query expression ***show***(E∩not(F)) highlights the molecule pathways passing through the box E but not F, as shown in Figure 9 (a), and the expression ***show***(G∩H∩I)) highlights the pathways passing through G, H, and I successively, as shown in Figure 9 (b).

A more complicated case such as ***show***(B∩(A∪C∪D)), which presents the molecule pathways that consist of three parts: (1) ***show***(B∩A) passing through B and A successively, (2) ***show***(B∩C) passing through B and C successively, and (3) ***show***(B∩D) passing through B and D successively, as shown in Figure 9 (c-d) (G3). The code generator can be used to define a box named A at the inner capsule, B at the corpus callosum, C at the brain stem, and D at outer capsule. Thus, the pathways passing through the corpus callosum and the other three tissues can be highlighted in the color specified by ***with***.

6.5 Performance and Complexity Analysis

We have also conducted performance evaluation and complexity analysis for the designed compiler. The compiling time for all script codes used in the experiments is shorter than 1.253 milliseconds (Appendix B) because it is only an interpreting compiler without a relocator and linker. The




```
// defineROI (...); // defined ROIs are global variables
load {dtiData("human_brain02")};
placeANewBox {boxID("E"), at("ROI#HB21"), color=red, alpha=0.8};
placeANewBox {boxID("F"), at("ROI#HB22"), color=yellow, alpha=0.8}; (a)
with (color=green,alpha=1.0) {
   show(E⋂not(F));          }; // end of with
```

```
load {dtiData("pig_heart")};
placeANewBox {boxID("G"),at("ROI#PH11"), color=red,alpha=0.8};
placeANewBox {boxID("H"),at("ROI#PH12"), color=green,alpha=0.8};
placeANewBox {boxID("I"),at("ROI#PH13"), color=orange,alpha=0.8}; (b)
with(color=yellow,alpha=1.0) {
   show(G⋂H⋂I); }; // end of with
```

```
load {dtiData("pig_heart")};
lineStyle {color=red, width=1.0};
placeANewBox {boxID("A"), at("ROI#PH14"), color=red,alpha=0.8}; (c)
placeANewBox {boxID("B"), at("ROI#PH15"), color=green,alpha=0.8};
placeANewBox {boxID("C"), at("ROI#PH16"), color=purple,alpha=0.8};
placeANewBox {boxID("D"), at("ROI#PH17"), color=brown,alpha=0.6};
```

```
load {dtiData("pig_heart")};
with (color=yellow,alpha=1.0) {
   show(B⋂(A∪C∪D));                                    (d)
   pause(5 seconds); // after 5-sec pause,
   // show(...);    // then show other queries
}; // end of with
```

Figure 9: The expression-based queries for DTI fiber data. (a) The queried fiber tracts of "*show*(E∩not(F))" (can be also written as *show*(E-F)). (b) The queried fiber tracts of "*show*(G∩H∩I))". (c) Place four boxes to highlight the DTI fiber tracts which pass through each box. (d) The queried fiber tracts of "*show*(B∩(A∪C∪D))" (can be also written as (B∩A) ∪ (B∩C) ∪ (B∩D)).

compiling time of the codes for pathline presentation is slightly longer than that of the others because it needs to do color bindings for all clusters in the statement ***trace***. For more information about the performance evaluation of the compiler, please refer to **Appendix B**.

7 DISCUSSION AND FUTURE WORK

There are some limitations of IGScript and we plan to address them in future work:

The utility of IGScript would be enhanced by further working on its automatic ROI detection because the ROIs are vital to obtaining a deep insight into scientific data presentations and analysis. The ROIs in the current version are interactively defined by ***defineROI*** in the dual-space tool of the code generator. We plan to use machine learning approaches to recommend ROIs, where the regions are the locations with salient features. The feature would enable the animation of ROI traversals to be generated fully automatically. Alternatively, the log data of presentation animation can be fully utilized to recommend ROIs. The interactive exploration processes can be recorded as script codes by IGScript, and the target regions can be recorded into log data. Besides, an eye-tracking device would be a good option for collecting data for ROI recommendations.

IGScript is suitable for post-analysis for volume data presentation because the rendering results are dependent on the transfer function design. IGScript uses the statement ***rgbaScheme*** to select and load the existing transfer functions. Thus, the transfer functions should be designed and ready before the volume data presentation if we are desired to get an effective case study (or storytelling).

The current version of IGScript covers data story cases on the four frequently-used general data types in scientific visualization. It can be easily adapted to other use cases for these types of data. As for a new data type, APIs on data loading and application-specific presentations (if any) should be added to the IGScripts library, while the general-purpose presentations are directly supported. The coverage of the general-purpose presentations of IGScript includes ROI definition (G1) and viewpoint changing (G2). Besides, IGScript is not limited by data size. It can process larger data if the hardware can handle it. We do not evaluate the scalability on data size because it is not the current contribution of this work.

Regarding the general-purpose grammars, it is easy to connect them with the existing open-source systems like Paraview and VTK. IGScript uses quaternions, which can be converted to homogeneous transformation matrices. The quaternions and matrices can be directly loaded by the trackball component of the open-source systems. Regarding the application-specific grammars, they should be added to the libraries of the open-source systems. For example, the libraries of pathline tracking in vector field data presentations and set operators on DTI fiber queries should be added into the open-source systems, if the application-specific grammars are also expected to be supported.

Some of user study participants consider it would be better if we provide more sample codes, or if the identifiers and parameters can be redefined using their technical terms or closer to natural languages, as described in Section 5.3. In the future work, we plan to design domain-specific grammars for different domains by using the corresponding technical terms or making the grammar closer to natural languages. Furthermore, some of participants suggest us to visualize the flow chart with visual code blocks. Users just need to edit each code blocks and the editable parameter values in

the coding space. They think it would be much better to provide the flow charts for each presentation animation and conceal some currently unchanged codes. In the future, we plan to visualize the flow chart with code blocks for each script-driven presentation animation generation.

Additionally, we plan to apply IGScript to storytelling talks or storytelling discussions among peer experts in the future. Potential examples include the popularization of science talks on Coronavirus Disease 2019 (COVID-19), or the generation of storytelling animations similar to Hans Rosling's TED Talks. Besides, we plan to extend the work to support more interactive illustrations for scientific data. Another extension of the work is to apply IGScript to remote immersive exploration in cooperative visualization. The overloaded data exchange between VR/AR devices is one of obstacles to the widespread use of immersive devices, especially remote cooperative data explorations. In this scenario, IGScript would translate the interactive data story generation steps into script codes, which would replace the visualized result data (e.g., image data or sub-volume data) in data transmission. This technique would alleviate the burden on real-time data exchange among VR devices to a large extent. Finally, we aim to support more general types of scientific data.

8 CONCLUSIONS

Most existing interpretive grammar-based scientific visualization approaches enhance the customizability in either the computation stage, or the rendering stage, or both. However, few approaches focus on the data presentation stage. Furthermore, almost all of these approaches leverage the existing components from the GPLs instead of developing a standalone compiler. The style of the grammars and programming of them are highly dependent on the host GPLs, which poses a great challenge about the steep learning curves for the domain experts who have limited knowledge about programming. Thus in this paper, we design an interaction grammar-based tool named IGScript for interactive scientific data presentations. IGScript exerts the advantages of standalone compiler and script-like grammar named interaction grammar. The script codes are independent on the host language and look simple and can be easy to follow the style of natural language-like grammars, which is of significance to the domain experts who have limited knowledge about programming. A special-purpose compiler is designed to convert natural language-like grammar scripts into data presentation animations, and a code generator (or a decompiler) is developed to translate the presentation steps into script codes. Linked dual-space views are designed to provide visual steering and visual feedback. It makes the animation clips of the data presentation easy to be cut, copied, pasted, or deleted. The demonstration results show that the flexible, legible, and unambiguous interaction grammars of IGScript support general-purpose and several application-specific data presentations. Evaluations including a user study, four case studies and performance analysis demonstrate the usability and customizability of IGScript.

ACKNOWLEDGMENTS

We thank the reviewers for their valuable comments and appreciate all the domain experts and user study participants. This work was supported by National Nature Science Foundation of China (61702271), and Postgraduate Research & Practice Innovation Program of Jiangsu Province (SJCX20_0445).

REFERENCES

[1] David Akers, Anthony Sherbondy, Rachel Mackenzie, Robert Dougherty, and Brian Wandell. 2004. Exploration of the brain's white matter pathways with dynamic queries. In *IEEE Visualization*. IEEE, Austin, Texas, USA, 377–384.

[2] Mohamed Almorsy, John Grundy, Richard Sadus, Willem van Straten, David G. Barnes, and Owen Kaluza. 2013. A Suite of Domain-Specific Visual Languages For Scientific Software Application Modelling. In *IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, San Jose, USA, 91–94.

[3] Luke Anderson, Tzu-Mao Li, Jaakko Lehtinen, and Frédo Durand. 2017. Aether: an embedded domain specific sampling language for Monte Carlo rendering. *ACM Transactions Graphics* 36, 4 (2017), 99:1–99:16.

[4] Francisco Pérez Andrés, Juan De Lara, and Esther Guerra. 2007. Domain specific languages with graphical and textual views. In *International Symposium on Applications of Graph Transformations with Industrial Relevance*. Springer, Springer, Kassel, Germany, 82–97.

[5] Dimitar Asenov and Peter Muller. 2013. Customizing the visualization and interaction for embedded domain-specific languages in a structured editor. In *IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, San Jose, USA, 127–130.

[6] Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. 2004. Rule-Based Runtime Verification. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, Springer, New Orleans, USA, 44–57.

[7] Howard Barringer, Alex Groce, Klaus Havelund, and Margaret Smith. 2010. Formal analysis of Log files. *Journal of aerospace computing, information, and communication* 7, 11 (2010), 365–390.

[8] Howard Barringer and Klaus Havelund. 2011. Internal versus external DSLs for trace analysis. In *International Conference on Runtime Verification*. Springer, Springer, Los Angeles, USA, 1–3.

[9] Howard Barringer, David Rydeheard, and Klaus Havelund. 2008. Rule systems for run-time monitoring: from Eagle to RuleR. *Journal of Logic and Computation* 20, 3 (2008), 675–706.

[10] Michael Bostock and Jeffrey Heer. 2009. Protovis: A Graphical Toolkit for Visualization. *IEEE Transactions on Visualization and Computer Graphics* 15, 6 (2009), 1121–1128.

[11] Kevin J Brown, Arvind K Sujeeth, Hyouk Joong Lee, Tiark Rompf, Hassan Chafi, Martin Odersky, and Kunle Olukotun. 2011. A heterogeneous parallel framework for domain-specific languages. In *2011 International Conference on Parallel Architectures and Compilation Techniques*. IEEE, IEEE, Galveston, TX, USA, 89–100.

[12] Stefan Bruckner and M. Eduard Grller. 2005. VolumeShop: An Interactive System for Direct Volume Illustration. In *IEEE Visualization*. IEEE, Minnesota, USA, 671–678.

[13] Wei Chen, Zi'ang Ding, Song Zhang, Anna MacKay-Brandt, Stephen Correia, Huamin Qu, John Allen Crow, David F. Tate, Zhicheng Yan, and Qunsheng Peng. 2009. Datasets of DTI Fiber Explorer. http://sourceforge.net/projects/dtiexplorer/. http://www.cad.zju.edu.cn/chenwei/interface/index.html A Novel Interface for Interactive Exploration of DTI Fibers.

[14] Wei Chen, Zi'ang Ding, Song Zhang, Anna MacKay-Brandt, Stephen Correia, Huamin Qu, John Allen Crow, David F. Tate, Zhicheng Yan, and Qunsheng Peng. 2009. A Novel Interface for Interactive Exploration of DTI Fibers. *IEEE Transactions on Visualization and Computer Graphics* 15, 6 (2009), 1433–1440.

[15] Wei Chen, Zhicheng Yan, Song Zhang, John Crow, David Ebert, Ron Mclaughlin, Katie Mullins, Robert Cooper, Zi'ang Ding, and Jun Liao. 2009. Volume Illustration of Muscle from Diffusion Tensor Images. *IEEE Transactions on Visualization and Computer Graphics* 15 (2009), 1425–1432.

[16] Wei Chen, Song Zhang, Stephfan Correia, and David Ebert. 2008. Abstractive Representation and Exploration of Hierarchically Clustered Diffusion Tensor Fiber Tracts. *Computer Graphics Forum* 27, 3 (2008), 1071–1078.

[17] Hyungsuk Choi, Woohyuk Choi, Tran Minh Quan, David Hildebrand, Hanspeter Pfister, and Won-Ki Jeong. 2014. Vivaldi: A Domain-Specific Language for Volume Processing and Visualization on Distributed Heterogeneous Systems. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 2407–2416.

[18] Valerio Cosentino, Massimo Tisi, and Javier Luis Cánovas Izquierdo. 2015. A model-driven approach to generate external DSLs from object-oriented APIs. In *International Conference on Current Trends in Theory and Practice of Informatics*. Springer, Springer, Czech Republic, 423–435.

[19] Marcelo d'Amorim and Klaus Havelund. 2005. Event-based runtime verification of Java programs. *ACM SIGSOFT software engineering notes* 30, 4 (2005), 1–7.

[20] Romuald Deshayes. 2013. A Domain-Specific Modeling Approach for Gestural Interaction. In *IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, San Jose, USA, 181–182.

[21] Zachary DeVito, Michael Mara, Michael Zollhöfer, Gilbert Bernstein, Jonathan Ragan-Kelley, Christian Theobalt, Pat Hanrahan, Matthew Fisher, and Matthias Nießner. 2017. Opt: A Domain Specific Language for Non-Linear Least Squares Optimization in Graphics and Imaging. *ACM Transactions Graphics* 36, 5 (2017), 171:1–171:27.

[22] David J. Duke, Rita Borgo, Malcolm Wallace, and Colin Runciman. 2009. Huge Data But Small Programs: Visualization Design via Multiple Embedded DSLs. In *International Symposium on Practical Aspects of Declarative Languages*. Springer, Savannah, GA, USA, 31–45.

[23] Martin Ester, Hans-Peter Kriegel, Jrg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 226–231.

[24] Maurice René Fréchet. 1906. *On some points of the functional calculation*. Springer, Chapter Rendiconti del Circolo Matematico di Palermo, 1–72.

[25] Lars George, Arif Wider, and Markus Scheidgen. 2012. Type-safe model transformation languages as internal DSLs in Scala. In *International Conference on Theory and Practice of Model Transformations*. Springer, Springer, Prague, Czech Republic, 160–175.

[26] John C. Grundy, John Hosking, Karen Na Li, Norhayati Mohd Ali, Jun Huh, and Richard Lei Li. 2013. Generating Domain-Specific Visual Language Tools from Abstract Visual Specifications. *IEEE Transactions Software Engineering* 39, 4 (2013), 487–515.

[27] Sebastian Günther. 2009. *Agile DSL-Engineering with Patterns in Ruby*. Technical Report. Very Large Business Applications Lab.

[28] Sebastian Günther and Sagar Sunkle. 2012. rbFeatures: Feature-oriented programming with Ruby. *Science of Computer Programming* 77, 3 (2012), 11–18.

[29] Diansheng Guo and Xi Zhu. 2014. Origin-Destination Flow Data Smoothing and Mapping. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 2043–2052.

[30] Charles Hansen and Chris R. Johnson. 2005. *The Visualization Handbook*. Elsevier, Academic Press.

[31] Bernhard Jenny, Daniel M. Stephen, Ian Muehlenhaus, Brooke E. Marston, Ritesh Sharma, Eugene Zhang, and Helen Jenny. 2017. Force-directed layout of origin-destination flow maps. *International Journal of Geographical Information Science* 31, 8 (2017), 1521–1540.

[32] Pushpak Karnick, Stefan Jeschke, David Cline, Anshuman Razdan, E. Wentz, and Peter Wonka. 2009. A Shape Grammar for Developing Glyph-based Visualizations. *Computer Graphics Forum* 28, 8 (2009), 2176–2188.

[33] Gordon Kindlmann, David Weinstein, and David Hart. 2000. Strategies for direct volume rendering of diffusion tensor fields. *IEEE Transactions on Visualization and Computer Graphics* 6, 2 (2000), 124–138.

[34] Gordon L. Kindlmann, Charisee Chiw, Nicholas Seltzer, Lamont Samuels, and John H. Reppy. 2016. Diderot: a Domain-Specific Language for Portable Parallel Scientific Visualization and Image Analysis. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (2016), 867–876.

[35] Guozheng Li, Min Tian, Qinmei Xu, Michael J. McGuffin, and Xiaoru Yuan. 2020. GoTree: A Grammar of Tree Visualizations. In *ACM CHI Conference on Human Factors in Computing Systems*. ACM, Honolulu, Hawaii, US, 1–13.

[36] Eduardo Marques, Valter Balegas, Bruno F Barroca, Ankica Barisic, and Vasco Amaral. 2012. The RPG DSL: a case study of language engineering using MDD for generating RPG games for mobile phones. In *The 12th workshop on Domain-specific modeling*. ACM, Arizona, US, 13–18.

[37] Jean-Eudes Marvie, Cyprien Buron, Pascal Gautron, Patrice Hirtzlin, and Gaël Sourimant. 2012. GPU Shape Grammars. *Computer Graphics Forum* 31, 7-1 (2012), 2087–2095.

[38] Tim McGraw. 2017. *Encyclopedia of Computer Graphics and Games*. Springer, Berlin, Germany, Chapter Tensor Field Visualization, 11–12.

[39] David Méndez-Acuña, José A Galindo, Thomas Degueule, Benoît Combemale, and Benoit Baudry. 2016. Leveraging software product lines engineering in the development of external DSLs: A systematic literature review. *Computer Languages, Systems & Structures* 46 (2016), 206–235.

[40] Marjan Mernik, Jan Heering, and Anthony M Sloane. 2005. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)* 37, 4 (2005), 316–344.

[41] Jennis Meyer-Spradow, Timo Ropinski, Jrg Mensmann, and Klaus Hinrichs. 2009. Voreen: A Rapid-Prototyping Environment for Ray-Casting-Based Volume Visualizations. *IEEE Computer Graphics and Applications* 29, 6 (2009), 6–13.

[42] Rebecca Morgan, Georg Grossmann, Michael Schrefl, Markus Stumptner, and Timothy Payne. 2018. VizDSL: a visual DSL for interactive information visualization. In *International Conference on Advanced Information Systems Engineering*. Springer, Springer, Tallinn, Estonia, 440–455.

[43] Karl Pearson. 1901. On Lines and Planes of Closest Fit to Points in Space. *Philos. Mag.* 2, 11 (1901), 559–572.

[44] John Plate, Thorsten Holtkaemper, and Bernd Froehlich. 2007. A Flexible Multi-Volume Shader Framework for Arbitrarily Intersecting Multi-Resolution Datasets. *IEEE Transactions on Visualization and Computer Graphics* 13, 6 (2007), 1584–1591.

[45] Peter Rautek, Stefan Bruckner, Eduard Grller, and Markus Hadwiger. 2014. ViSlang: A System for Interpreted Domain-Specific Languages for Scientific Visualization. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 2388–2396.

[46] Lukas Renggli and Tudor Gîrba. 2009. Why Smalltalk wins the host languages shootout. In *Proceedings of the International Workshop on Smalltalk Technologies*. ACM, Brest, France, 107–113.

[47] Christian Rieder, Stephan Palmer, Florian Link, and Horst K. Hahn. 2011. A Shader Framework for Rapid Prototyping of GPU-Based Volume Rendering. *Computer Graphics Forum (EuroVis'11)* 30, 3 (2011), 1031–1040.

[48] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2017. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2017), 341–350.

[49] Arvind Satyanarayan, Ryan Russell, Jane Hoffswell, and Jeffrey Heer. 2016. Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (2016), 659–668.

[50] Hans-Jörg Schulz, Thomas Nocke, Magnus Heitzler, and Heidrun Schumann. 2013. A Design Space of Visualization Tasks. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2366–2375.

[51] Liming Shen, Xueyi Chen, Richen Liu, Hailong Wang, and Genlin Ji. 2020. Domain-Specific Language Techniques for Visual Computing: A Comprehensive Study. *Archives of Computational Methods in Engineering* 28, Accepted (2020), 1–22.

[52] Anthony Sherbondy, David Akers, Rachel Mackenzie, Robert Dougherty, and Brian Wandell. 2005. Exploring connectivity of the brain's white matter with dynamic queries. *IEEE Transactions on Visualization and Computer Graphics* 11, 4 (2005), 419–430.

[53] Thomas Stahl, Markus Voelter, and Krzysztof Czarnecki. 2006. *Model-driven software development: technology, engineering, management*. Wiley, New Jersey, USA.

[54] Jo Wood, Jason Dykes, and Aidan Slingsby. 2010. Visualisation of Origins, Destinations and Flows with OD Maps. *The Cartographic Journal* 47, 2 (2010), 117–129.

[55] Yalong Yang, Tim Dwyer, Bernhard Jenny, Kim Marriott, Maxime Cordeil, and Haohui Chen. 2019. Origin-Destination Flow Maps in Immersive Environments. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2019), 693–703.