Multiverse data-flow control

Multiverse data-flow control

February 28, 2011

Benjamin Schindler, Jürgen Waser, Raphael Fuchs, Ronny Peikert

Abstract

In this paper we present a data-flow system which supports comparative analysis of time-dependent data and interactive simulation steering. In such a system data is created on-the-fly and the user can introduce branching events to investigate multiple scenarios. The novel data-flow model automatically generates the required data for user requests without further interaction. In particular, we generalize the notion of capabilities, jobs and settings of a node which steer the data-flow execution. Based on a notion of scope and validity it becomes straightforward for the user to specify settings which vary over time or scenarios. Intuitive navigation concepts let the user select what he wants to see, the decision which data is required throughout the data-flow is then handled automatically. The approach is based on a graph-based model of a topologically sorted data-flow. Based on that we show that the notion of a scope allows to describe the computation of jobs and validity using set intersections and set unions. We demonstrate the approach by discussing two small case studies.

Keywords: dynamic data management, data-flow, time-varying data, visual knowledge discovery, graph-based data handling, visualization system design

1 Introduction

In 1989 Upson et al. [UFK*89] published a seminal paper on the data-flow based visualization system AVS. In the following years this topic received a surge of attention and several large systems besides AVS, such as IRIS Explorer (SGI), Visualization Data Explorer (IBM) and VTK (Kitware) were developed. Researchers quickly pointed out several issues related to data-flow based systems [RBM*92], for example: intuitiveness of the user interface, creative exploration of data based on the data-flow, extension to non-visualization tasks, complex event handling so that information can be passed to other nodes conditionally, transfer of other information than raw data (such as user selections), ability to handle multiple data sets simultaneously and parallel node execution. It was noted that "the existing systems are certainly not the last word" [RBM*92]. Recent work has extended the meaning of *simulation steering* from interaction with a single simulation run while the computation is still active [Par99] to the possibility to perform a large number of related runs interactively [WFR*10, Wor]. In the

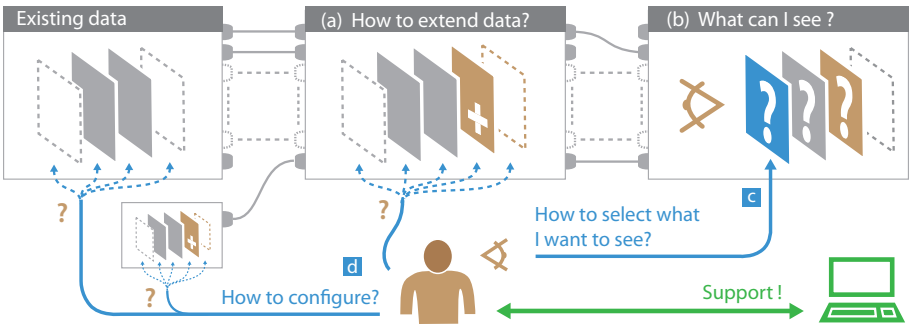

Figure 1: Problems when steering a complex data-flow system. (a) The user needs control mechanisms to increase the existing data base to support multiverse analysis. (b) The system lacks multiverse flow processing algorithms to report what the user can see in the available views. (c) Intuitive concepts are missing to let users select what they want to see. (d) Finally, the user is responsible for distributing the work load to each of the involved nodes. Currently, there is no machine support for these tasks.

Visdom [Vis] framework data is created on-the-fly in a GPU-based fluid simulation system. One important difference is the need to model data-creation which creates new timesteps of which we cannot predict their timevalue since the simulation does adaptive timesteps. In this setting, novel ways to coordinate node settings in the data flow are required and, indeed, the standard data-flow handling cannot be the last word. Internal handling and user interaction with multifield data and the selection of attributes to be passed to the nodes is non-trivial already. The problems become even more complicated when techniques such as finite time Lyapunov exponents, pathlines and other time-dependent visualization techniques have to be computed for simulated data which can be created on the fly and can contain branching events. The dataflow execution presented in this paper is inspired by the generic dataflow analysis concept often used in the optimization pass in compilers [Kil73]. We use it to compute the set of data the dataflow can produce and to calculate the minimal set of data required to fulfill a user request.

2 Overview

In this paper we discuss how to extend the capabilities of a data-flow based visualization framework to support multiverse data creation and analysis. This requires a novel algorithm to analyze the data-flow but also new interaction concepts for flexible view navigation for visual analysis. Based on the new interaction concept World Lines can be used to test and compare alternative decisions made during interactive visual analysis. Nodes which perform time-independent operations do not require additional logic. The contributions of this paper are as follows

- Data-flow analysis which allows node-execution for multiverse data and on-the-fly data creation.

- Intuitive user interaction by integration of data-flow settings management and the World Lines interaction metaphor.
- With the suggested data-flow processing algorithms, World Lines can be linked to views and nodes to steer any data-flow module, not only simulation modules.

3 Terms

We need to recapitulate some terms to simplify the discussion in the remaining text [WFR*10]. A *state-space* is the space spanned by all variables of a system including positions, time values or internal object states. A *frame* is a representative of one point in state-space, for example a simulation result at a given point in time. A *track* is a consecutive set of frames. For example, a single simulation run that comprises a specific set of simulation parameters is represented as such a track. An *event* is a modification of the system parameters (e.g. a user intervention or an external data update) that results in a change of the systems behavior and is recorded as a branch. A set of causally related tracks is called a *World Line* and presents a possible outcome. The parallel worlds at a given point in time are defined as the frames of all tracks that temporally overlap at this time value. The whole data set is a *multiverse*.

4 Related Work

The data flow diagram is an abstract representation of the flow of data during data processing [Gan79]. Based on this visual description of data processing Haeberli [Hae88] describes a visual programming language for interactive graphics which is based on a graph of connected nodes. Myers [Mye90] suggested a taxonomy for visual programming where he stresses that the high-level description of the data-flow helps in gaining insight into the computation process. LabView [Bis09] is a commercial platform for visual programming, released first in 1986. This approach is also applicable to creating visualization systems. Upson et al. [UFK*89] presented AVS, a data-flow based visualization system. Later, Schroeder et al. [SML96] published their concept of the visualization toolkit (VTK) C++ library, which describes a data-flow network which can be steered using a procedural language. Parker [Par99] discusses SCIRun, a problem solving environment which allows to steer simulations during their execution. Maubach and Telea [MT05] present an integrated system for the numerical solution of differential equations. Bruckner and Möller [BM10] present an approach to design computer graphics effects based on sampling the parameter space of a fluid simulation.

Abram and Treinish [AT95] discuss how data-flow execution per time-step can be achieved and suggest a roots-to-leaves pass to determine which nodes need execution. Telea [Tel00] discusses the relationship between architectural models for building visualization systems and discriminates between application libraries, turn-key systems and data-flow based application builders. He concludes that the disadvantage of the traditional data-flow model is, that it is “hard to use an existing visualization framework to perform simulations“. The extended data-flow concept presented in this paper alleviates this apparent conflict.

Even though the data-flow concept is not new, recently, there is renewed interest in extending and simplifying the use of data-flow based systems. Bavoil et al. [BCC*05] present a meta-data-flow implementation which can be used to represent data-flows from multiple visualization systems (e.g. VTK) as XML files and also provide a unified user interface. This allows them to track the evolution of data-flows during interaction [CFS*06]. VisMashup [SLA*09] allows to create custom applications based on a more general visualization system such as VTK by creating a small application based on a fixed data-flow. Still, the data-flow execution model matches the one of the underlying system. Recommender systems to provide guidance for the user which nodes are applicable were presented by Fujishiro et al. [FTIN97] and Telea et al. [TvW00]. Recently, Koop et al. [KSC*08] discuss a data-driven recommender system for VTK which is based on a database of examples. Botha and Post [BP08] distinguish event- and demand-driven scheduling and present an interesting hybrid scheduling approach for data-flow execution with streaming, which includes analysis of the dataflow for time-independent data to find streamable tasks. Childs et al. [CBB*05] describe a contract based system which allows nodes to communicate additional information when processing data and domain decomposition is applied. Wolter et. al [WAH*09] present a time model for the representation of time suited to the visualization and animation of time-dependent data. VTK uses a demand-driven execution model and even though there has been much work on extending this data flow model, Vo et al. [VOS*10] stress that it is still challenging to extend this model; in this work Vo et al. extend the data-flow system of VTK for parallel execution on multi-core systems.

Biddiscombe et al. [BGMT07] extend the VTK data-flow to support nodes which require multiple timesteps. Their approach does not perform caching when processing multiple timesteps, which greatly simplifies the required logic. Furthermore, they do not formalize how to decide which nodes actually have to create which information. Since our work is related to this extension of VTK we want to list several other differences explicitly: our model is suited to a multiverse scenario which can contain branches, we have to consider the fact that new data can be created by the user at any time, we present an extensive graphical user interface to simplify interaction with world- and time-dependent data, algorithms do not have to steer themselves by passing specific request to the data-flow management system (compare Section 5.1 of [BGMT07]), the analysis allows partial data flow executions and we perform data caching.

5 Interactive data-flow navigation

The presented data-flow algorithms open up a variety of steering and analysis possibilities in a modular environment. With the new processing, nodes are capable to report their capabilities to the user. From the user perspective, the capabilities of a node are a set of frames which can be generated by the node. Since the display as tracks and frames is a natural way to visualize the computable frames, World lines are suitable for this task.

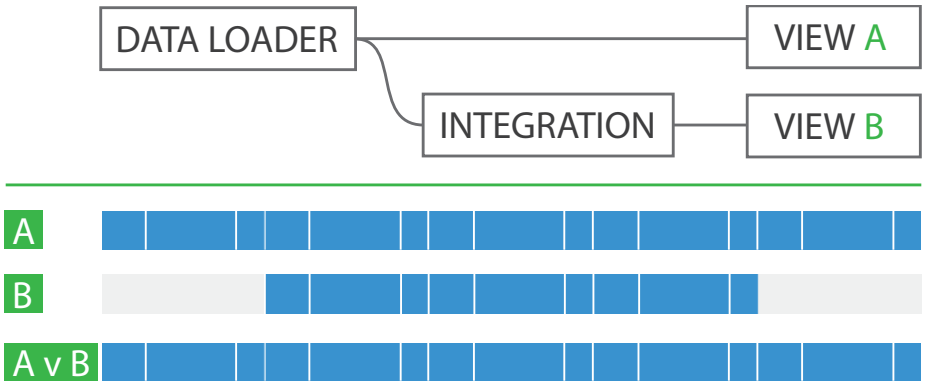

Figure 2: Frames of different size and duration to visualize node capabilities in the World Lines view. (A) Capabilities of node A. (B) Capabilities of node B (The input time-integration node 'cuts off' frames from the data loader). (A∨B) Unified capabilities.

5.1 Visualization of Capabilities

Data-flow capabilities can be directly translated into the frames of World Lines. Each view node controlled by World Lines supplies his capabilities which are visualized as tracks of frames (see Figure 2). The width of a frame is determined by the step width of the simulation. For data items without duration, a default value is assigned. Per default, World Lines visualize the unified capabilities of all views. Optionally, the user can visualize individual capabilities of specific nodes (see Figure 2).

5.2 Navigation of Capabilities

The user can navigate the World Lines from frame to frame using cursors (Figure 3-a). The cursors can be used to select the time value of the active frame in the active World Line. We provide a World Lines cursor for every linked view. Each of these cursors has the following properties:

Time value Controlled by the position of the cursor.

World Line A specific path through the tree of tracks.

Valid nodes The list of nodes the cursor is able to navigate.

Name and color Used to color the cursor handle for identification in the linked view node(s).

A cursor can only select frames which are part of the capabilities of all linked nodes. In other words, a cursor navigates the intersection of the capabilities of its controlled nodes. The navigation techniques need to account for this fact. For example, the jump action to a particular time value snaps to the closest frame that is valid. All cursor properties can be edited inline. We follow the principle of keeping property editors close to the visual entity. Therefore, an edit-button next to the cursor label (see Figure 3-a) toggles the visibility of the cursor configuration panel.

5.3 Job Assignment

By navigating the cursors, users can specify what they want to see. The frame selected by a cursor determines a request that is sent to the underlying data-flow network, assigning jobs to the controlled view nodes respectively. The users need not worry about the internals, a correct flow execution is guaranteed. For example, the user specifies multiple branching World Lines which are not yet simulated. It is then possible to select any frame in any branch and the system guarantees that all required frames are simulated in the correct order with the appropriate settings. The related data-flow analysis is discussed in Section 6. If the computations are time-consuming, the user can schedule them using the cursors.

5.4 Cursor Activation

Only one cursor can be active at a time. A cursor is activated by clicking onto its draggable component. The World Lines view immediately highlights the associated World Line by coloring. Active and non-active cursors differ with respect to appearance as shown in Figure 3-c.

Even though individual views can be navigated with their own cursors, the notion of an active cursor is still important. This is due to the optional presence of steering monitors that have been introduced with the World Lines system. These monitors are linked to the World Lines view to enable modification of track settings in a semantic way. The state of these monitors is synchronized with the track containing the active frame.

5.5 Special Purpose Cursors

The presented World Lines cursor is used to specify a single frame. In the following we refer to this type of cursor as a standard cursor. To manipulate node parameters conveniently we introduce special purpose cursors. We suggest two cursor types to manipulate time- and world-related node properties. Figure 3 shows a list of the available cursors along with example visualizations.

Integration Cursor This type of cursor can be used to visualize and control the integration range of an integration node. The cursor operates in two modi. Section 6.5.1 details the internal logic of these modi. In the integration mode (Figure 3-b), we attach to the cursor a colored ribbon tangent to the active World Line. The ribbon forms a stair case pattern if spanning multiple tracks of the World Line. The component is interactive in a way similar to a dual-slider widget. Users can drag the end knobs of the component in order to manipulate the range of the integration. In integration mode, the ribbon describes the integration range in forward- and backward-time. The integration starts at the active frame and spans the selected integration range. Figure 3-b illustrates this mode. Note, that the free surface remains the same since the cursor position is unchanged. In the simulation mode the integration starts at a fixed frame indicated by the start of the ribbon and extends to the cursor's frame. Figure 3-c). illustrates the simulation mode. The cursor has been moved, resulting in both a change of the integration range and the free surface.

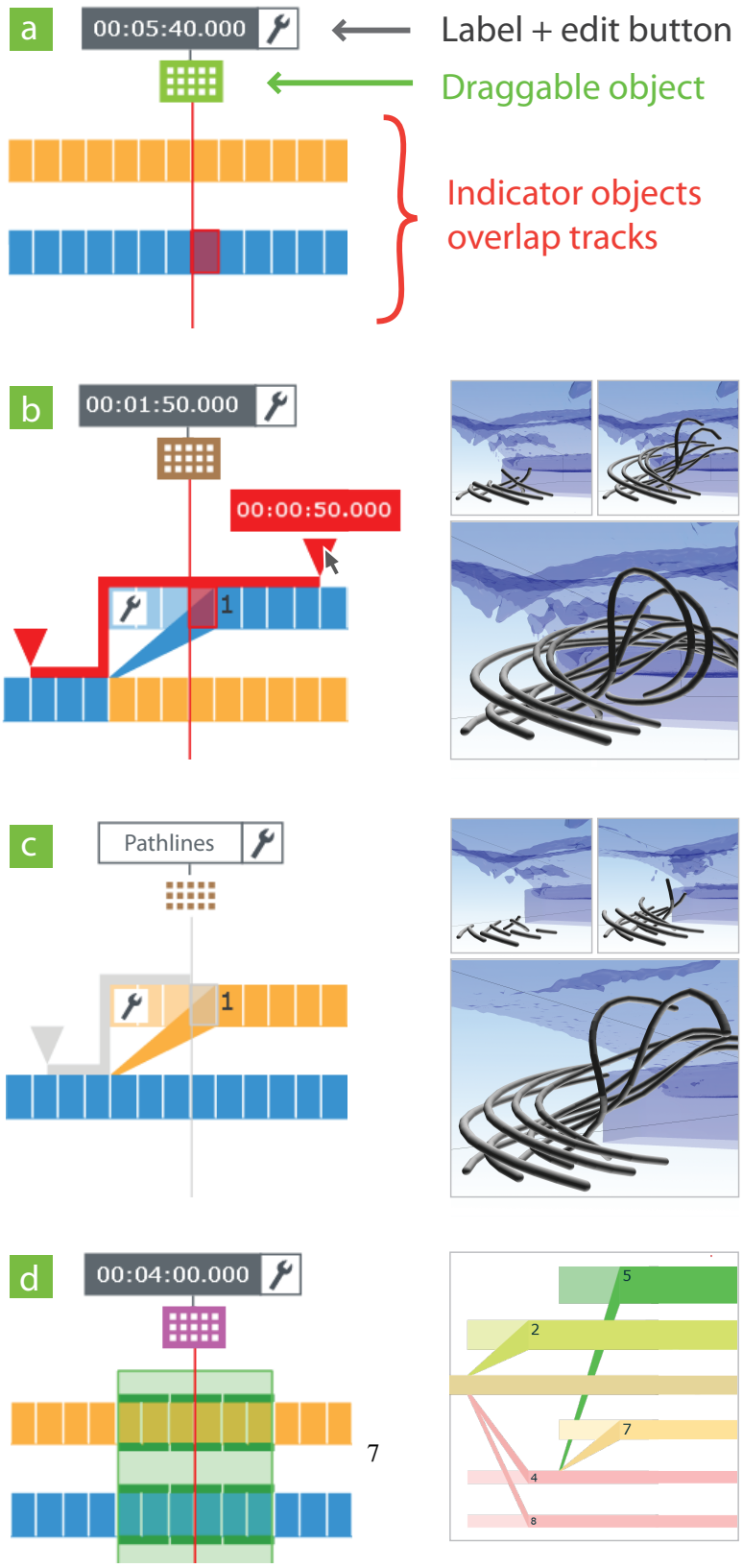

Figure 3: World Lines cursors. (a) Standard cursor. (b) Integration cursor in integration mode. The integration range is changed, producing a series of pathlines while the free surface remains unchanged. (c) Integration cursor (inactive) in the simulation mode.

Brush Cursor This type of cursor can be used select multiple frame relative to the cursor. This supports comparative visual analysis of multiple frames (Figure 3-d). In the previous World Lines version, a current timestep visualization mode for World Lines was presented. In this mode, all frames that are parallel at the current time are automatically brushed in order to compare them according to specific analysis criteria. The brush cursor now enables a generic extension to this mode, providing the means to move a custom selection tool across time and allowing the inclusion of additional frames. This is useful to analyze statistical quantities such as average values across several frames instead of inspecting the frames at a single time step alone. Per default, a cursor for each view node is generated. In addition, any node in the flow network can request the creation of associated special purpose cursors. Users can also create additional cursors and assign them multiple nodes. This can be useful for bookmarking or to create cursors for synchronized navigation of several selected views while keeping individual navigation cursors at the same time.

5.6 Cursor Grouping

A cursor can be associated with a list of views in order to navigate their nodes synchronously. There are cases where the analyst is interested in inspecting the temporal evolution of nodes that are offset to each other with respect to time or parallel worlds. This is useful for example to monitor the evolution of two simulation scenarios in two views side by side. For this purpose, the concept of nested cursor grouping has been developed. Interaction with one cursor modifies all grouped cursors. In a context menu, the user can specify whether the contained cursors should share a common World Line or a common time value or both. Cursors groups can also be further nested into other groups in order to define complex view-linkage behavior. It is possible to analyze the data-flow network to group cursors automatically. An example is the combination of an integration cursor and a view cursor if the underlying nodes are connected. The ability to request several multiverse frames at once from a single node is only possible with the presented data-flow processing.

5.7 View Navigators

In many visualization systems, monitors are equipped with a slider-based component directly below views to enable navigation in time. This is an intuitive and fast way to change the time value of a view without interacting with other components. Based on this approach, we propose the direct navigation of a World Line in view navigators (Figure 4). A view navigator consists of three main components:

Cursor selector A list of World Line cursors capable of navigating the view.

Surrogate cursor A miniature version of the linked World Lines cursor. When navigating the surrogate, the original reflects the changes and vice versa.

Flattened World Line A horizontal navigation bar representing the World Line associated with the linked cursor.

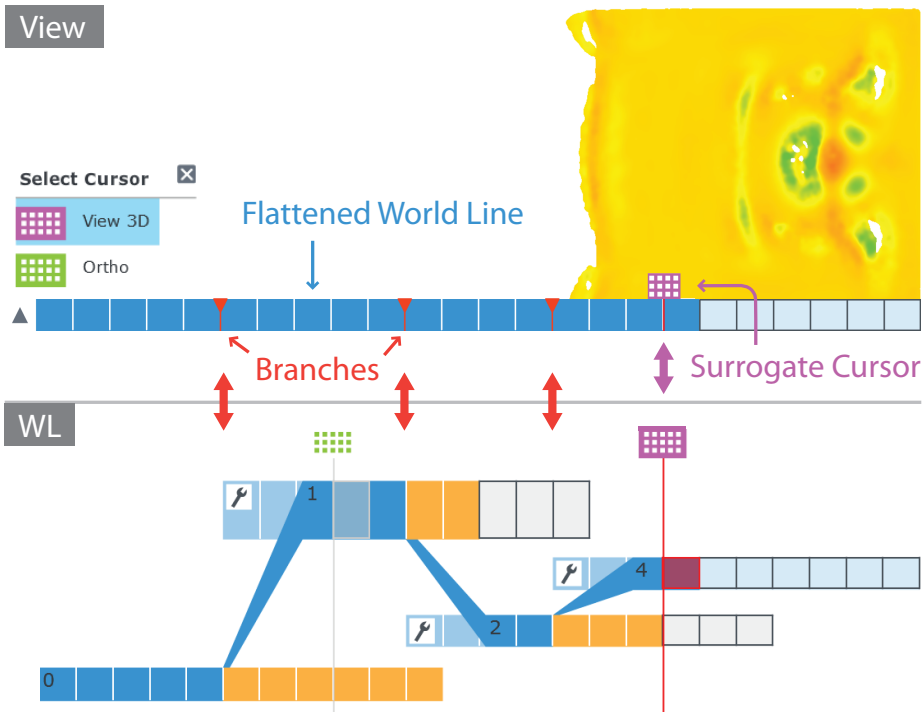

Figure 4: View navigator below a view. The view shows a flattened representation of the active World Line.

The flattened World Line is able to transport basic information obtained from its counterpart in the World Lines view. This includes the visualization of progress and the indication of branch locations. We can now use the aforementioned cursor properties and grouping methods to link the multiverse navigation between several view navigators in a highly customizable way.

5.8 Branching in the analysis stage

Based on the proposed data-flow analysis, World Lines can now be used to study node parameter changes without limitations with respect to the data-flow setup. An example is the integration of pathlines starting at a common frame but extending to different branches of the multiverse.

There are differences to be aware of when managing a data-flow without a simulation component as opposed to data-flows that involve simulations. First, there is no strict temporal causality for modules that load time steps or synthetically produce time steps. Therefore, as opposed to a simulation system, a frame can be generated without having to compute all frames before. Second, non-simulation modules can be computed in either temporal direction. Therefore the specification of a time value other then the start time of a parent track seems pointless for a branch. There are three reasons why we still provide the option to branch at any point in time: First, reduction of visual clutter. The user might not be interested in modification of certain frames and wants to discard them in the branched track. In this case, the internal data-flow cache can be optimized as well. Second, flexible usage of cursors that are defined along World Lines. For example, the integration cursor can be used to flexibly inspect integration ranges across tracks. Third, to visualize the analysis path of the user by a set of branches.

6 Dataflow Processing

In the following subsections we introduce the data-flow processing for multiverse data. The processing is based on the scope-set data structure; this data structure is used to describe the capabilities and jobs performed by a node in the data-flow. Based on this we explain the data-flow passes required to compute capabilities and jobs for each node. We also describe how settings which are not constant during execution can be included into this processing. Finally we explain notion of a non-standard node which makes our dataflow generally applicable.

6.1 Scope-Set data structure

A *scope* describes for which frames data items are valid. Data items can be valid for a whole track or for a single frame. If a data item is valid for a whole track it has *global* scope. If it is valid for a single frame it has *per-frame* scope. Every scope belongs to a track in the World Lines. A scope-set is a hierarchical collection of scopes which we will use in the following sections. It is used to model information related to the hierarchical structure of the World Lines. Figure 6 illustrates this concept.

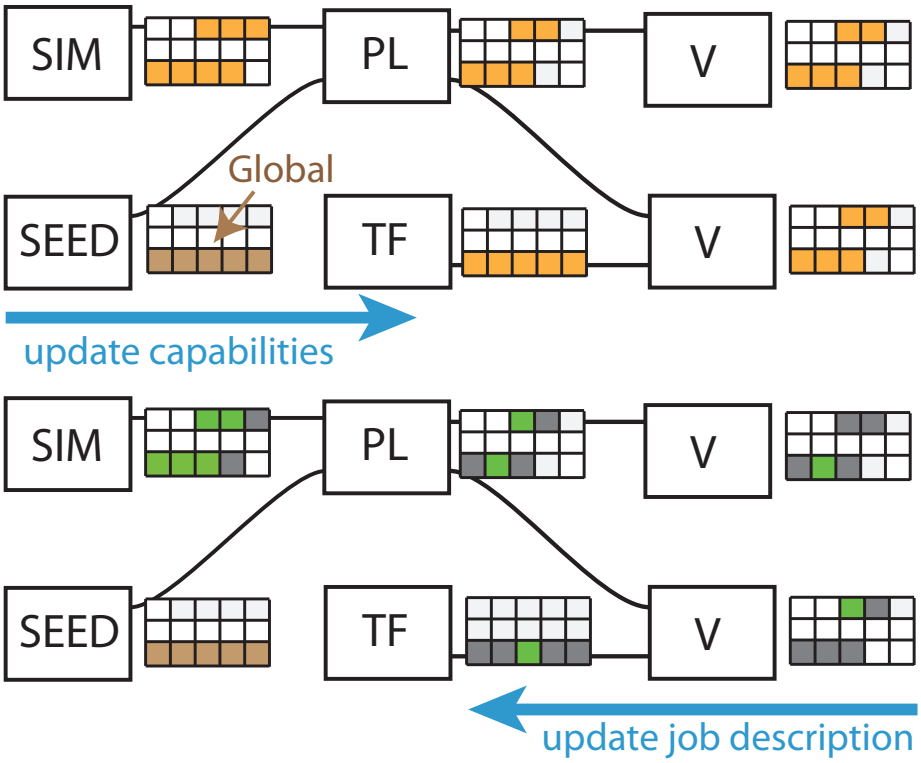

Figure 5: The dataflow passes illustrated with a simple example. A simulation node (SIM) creates data on the fly. The seeding node (SEED) creates input positions for the pathline node (PL). The result is visualized using two views (V), one using a transfer function (TF) input. To the right of every node is the scope set produced in respective pass; the capabilities in the upper figure, the job description in the lower figure.

A scope-set allows the two types of queries. The first query searches for a scope and returns true if it exists. We call it `find-scope`. In Figure 6 `find-scope` for a *global* scope at B1 would return false, and `find-scope` for a *per-frame* scope at timevalue 0 in R would return true. This is for example useful to check whether a data element has been produced by an earlier run.

The second query performs a hierarchical existence check `find-scope-hierarchical`: it returns true if a scope in the scope-set matches the requested scope. Scope *x* matches scope *y* if the range of scope *y* is completely overlapped by *x*'s range. For example a *global* scope in B2 from Figure 6 matches any *per-frame* scope in B2. The range of a scope extends to all the children tracks of its track. The hierarchical existence check therefore may have to traverse the World Lines hierarchy up to the root until it finds a matching scope. This hierarchical lookup can be compared to the way Cascading Style Sheets [CSS] work.

Every data object produced by a node has a scope. This way global data elements, i.e. data which does not change over time or track, are not produced for every single

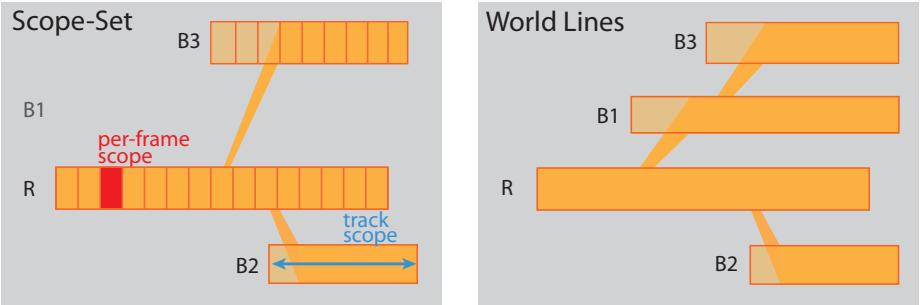

Figure 6: Illustration of the scope-set data structure. The scope-set is a subset of the World Lines. The World Lines have a root track R and three branches B1-B3. In the scope-set, R and B3 have *global* scope, B2 has multiple *per-frame* scopes. B1 inherits the scopes from its parent track R.

frame. This saves a lot of memory.

6.2 Node Capabilities

In this section we describe what node capabilities are and how they can be computed. The capability of a node is a scope-set describing the data it can produce. We compute the capability by traversing the dataflow from source to output. See Figure 6 for an illustration. Root nodes which produce data without input (i.e. file readers) have to fill the scope-set by themselves. But how is the capability of other nodes defined? It can be computed by the set intersection of the capabilities of the inputs:

$$capability(node) = \bigcap_{n \in inputs(node)} capability(n). \quad (1)$$

In other words, a node can produce a given scope only if all inputs can provide the requested scope. The crucial part is the definition of the intersection operator $\cap$ for scope-sets X and Y. Roughly speaking, the intersection of two scope-sets the intersection of two sets of sets. For every scope x in X we use `find-scope-hierarchical` to search for a matching scope in Y. If one is found x is added to the resulting scope-set. Then, for every scope y in Y we use `find-scope-hierarchical` to search for a matching scope in X. If one is found y is added to the resulting scope-set.

For example, in Figure 6 a simulation node (SIM) produces two tracks. A seeding node (SEED) produces global output in the root track. The intersection between these two scope sets is performed by first matching all scopes in SIM with those in SEED. Since the SEED root is global all 7 scopes in SIM are added to the result. In the other direction, no scope which overlaps the global root is found, therefore it is not added to the result of the intersection. The resulting intersection is therefore exactly the output of SIM.

6.3 Job Description

In this section we describe what job descriptions are and how they can be computed. A job description is a scope-set which represents the data the node is required to produce. This scope-set is calculated by traversing the dataflow from leaf to root. See Figure 6 for an illustration. Having scopes in the job description which are not in the capability makes no sense. Therefore, the job description is a subset of the capability of a node; it is a subset in the sense that every scope in the job description must be matched by a scope in the capability.

As discussed in Section 5.2, the user specifies the requested frames using the World Lines. The World Lines are therefore the link between the capabilities and the job description. The frames which the user has selected are converted to a scope-set which are then set to the respective leaf nodes. For example the user selects several timesteps for rendering an animation. The frames which the user has selected are then passed to the view nodes to specify which timesteps they have to render.

A node must produce all scopes requested by its output nodes. Otherwise, the output nodes are not capable of producing the required outputs. If a node has several output connectors it accumulates the set of scopes from its outputs. Therefore, the job description of a node can be computed by the set union of the job descriptions of the outputs:

$$jobdesc(node) = \bigcup_{n \in outputs(node)} jobdesc(n). \tag{2}$$

The union of scope-sets is defined similarly to the intersection: Every scope x in X is inserted into the result. Then, every scope y in Y is inserted into the result.

For every element in the job description, the node will find the matching scope in the capability and produce the data for this scope. This way data objects with a scope other than *per-frame* scopes can be produced saving needless computations. In the dataflow example (Figure 6), we have two view nodes with different job descriptions. Therefore, the pathline node whose output is consumed by both view nodes has to produce both scopes. This is the result of the union operation of both job descriptions. The transfer function node gets a job request for a *per-frame* scope in the branched track. It searches hierarchically for a matching scope and finds the *per-frame* scope in the root track. Then it calculates the matching scope in the root track overlapping the requested frame. The seed node produces the data associated with its only scope, the global scope.

6.4 Settings Validity

The World Lines can be used to specify different settings across different tracks. For example we can change the transfer function for each track. Therefore it is not possible to specify settings directly at a node. We extend the notion of node settings by assuming that every settings object is assigned a scope, much like any data object. This way it is possible to assign different settings for different scopes in the capability. For example with this formalism we can set a different transfer function for each timestep for a single node. We call the scope-set of all setting objects the settings capability. This is called settings capability since a node cannot produce outputs for a scope for which no

settings are specified. In other words, the node capability is influenced by the settings capability. For example, a node produces only a single data element associated with the *global* scope of the root track, but it has settings capability of several *global* scopes, each for a different track. In this case the node will produce different data objects for all these *global* scopes which by needs to be reflected in the node capability. The calculation of the capability of a node is therefore modified:

$$cap.(node) = \left(\bigcap_{n \in inputs(node)} cap.(n) \right) \cap settingsCap. \tag{3}$$

We can understand this formula by interpreting the node settings as a an input of the node. In the dataflow example (Figure 6), we have a transfer function producing *per-frame* objects. Per default, a transfer function produces just one *global* output. But if there are multiple *per-frame* setting objects, the capability becomes a set of multiple *per-frame* scopes. This could be used to animate a transfer function for example.

6.5 Standard and Non-standard Nodes

Standard nodes output the same scope as they received as input. For standard nodes each scope of the job description can be handled by processing the input data of this scope. Examples of such nodes are time-independent visualization techniques such as contouring, volume rendering, etc. When a node modifies the capability, we speak of a non-standard node. For example, a time interpolation node creates new data and therefore its capability is larger than the input capability.

A non-standard node differentiates between input- and output-capability and between input- and output-job-description. Input-capability is calculated by Equation 3 for non-standard nodes, too. But, the node is allowed to modify the input-capability and to create the new output-capability, e.g. by cutting away frames in case of a pathline node. The list of scope sets in a non-standard node can therefore be summarized as:

Input Capability: the set of scopes all input nodes are able to produce.

Output Capability: the set of scopes the node is able to produce. Calculated based on the input capability.

Output Job Description: the set of scopes the node is requested to produce from all its outputs. The node is allowed to add scopes to this set.

Input Job Description: The set of scopes a node requires from its inputs.

6.5.1 Simulation versus Integration

Nodes which perform simulation are an example of non-standard nodes. They can produce new timesteps based on the boundary conditions, by stepping forward in time.

Integration nodes are another important class of non-standard nodes. Pathlines, streaklines, particle integration and FTLE are just a few examples. There are two ways

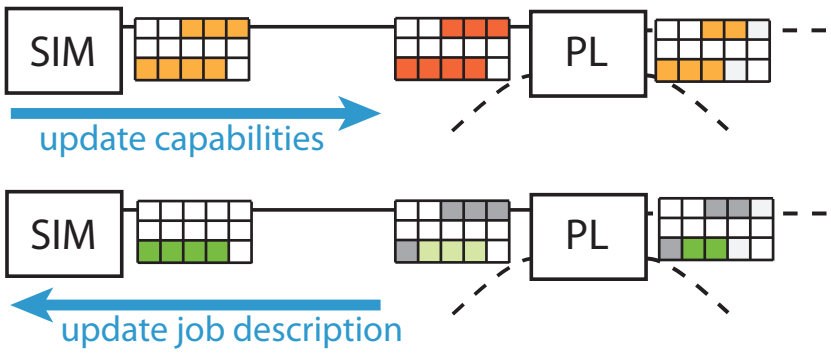

Figure 7: The simulation and the pathline node from figure 6 shown with input capability and input job description.

such a node could operate: for example by integrating lines starting from each frame. Or, integrating lines starting at a fixed position for the duration of the requested frame. In the first case an animation would show new lines for each frame. In the second case we would see one set of growing lines. In this case an integration node behaves similar to a common simulation node. Both modes have their uses. Within the presented general framework, it is possible to support both modes with a single node and one single code base.

In the data-flow example above (see Figure 7) we have the two non-standard nodes from Figure 6, a simulation node and a pathline node operating in integration mode. In the update capability pass, we see that the pathline node cuts away two frames from the input capability because they would require input which the simulation node is not able to produce. In the update job description pass the pathline node inserted a frame into the input job description. This frame is required because of the pathline extending in time. The simulation node inserted another two frames into its output job description. This happened because the simulation node incrementally produces data based on the previous step. Assuming no data has been generated yet, all data from the initial timestep needs to be generated. After the update job description pass, the data-flow is now ready to execute. The node logic needs to loop over all scopes in the output job description of each node to produce the required data.

6.5.2 Simulation nodes and branching

A simulation node incrementally produces data, i.e. it requires a current state to produce the next state. Without branches the selection of missing frames based on the job description is straightforward. This is done for example in Figure 7, where the simulation node added the first frame to its job description (light green). With branches, the situation becomes more complicated. When a branch is created, the user selects a setting of a node to change. As a result, the settings capability of that node, call it node A, will have the newly created track in its capability. There are now two different scenarios to consider. First, if the node A is unrelated to the simulation node, the sim-

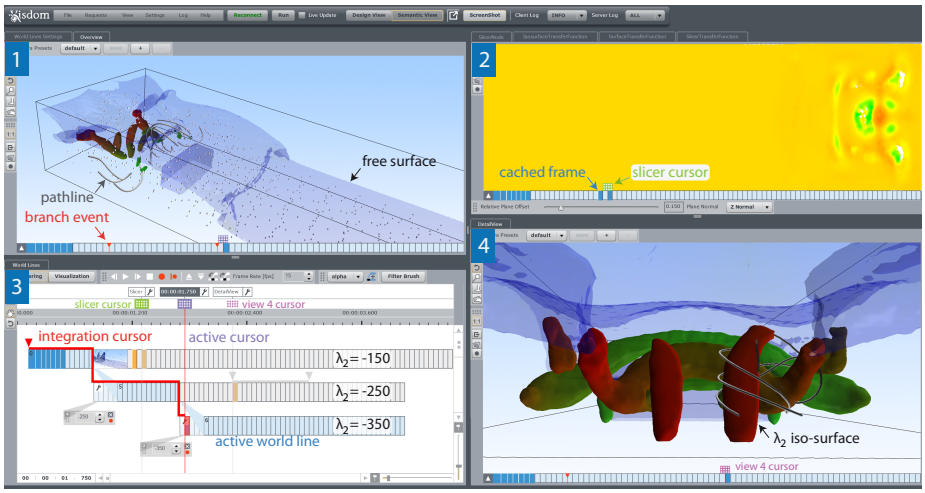

Figure 8: World lines used for the visualization of static data with branching to change the isovalue for the λ_2-isosurfaces. The active cursor has an integration cursor in simulation mode. The integration cursor is linked to the particles seen in the upper left view, which were seeded at the initial timestep. The rightmost (and inactive) cursor is linked to the lower right view. The attached integration cursor is in integration mode for the pathlines seen swirling around the λ_2-isosurface. The upper right figure shows the λ_2 volume sampled on a plain. Its associated cursor is the leftmost which is grouped with the middle cursor to link them with respect to navigation.

ulation will not see the newly created track. This means that requests for data objects on the new track will be served using the data from the old track, avoiding needless computation. In case the output of node A is used by the simulation node, the input capability of the simulation node will contain the new track. Therefore, a request for a data object from the new track will then be properly served by simulating using the requested simulation parameters.

7 Breaking Dam

To highlight the difference to branching when steering a simulation we use an SPH data set of a breaking dam [KFV*05]. It has 87 timesteps with approximately 650k particles in each frame. In Figure 8 we show a screen-shot of the analysis setup. The main point is that the whole data-flow can be steered only using the graphical user interface extensions of the World Lines presented in this paper. (1) shows a three dimensional overview rendering including an λ_2 iso-surface, pathlines and the free surface. (2) Shows a slice through the λ_2 volume. (3) Shows the World Lines setup. (4) The close-up view shows the vortices in detail. The World Lines show the frames available in the data. Frames for which the iso-surface has not yet been computed are colored light blue for the active World Line or grey everywhere else. Frames with cached values are shown in dark blue or yellow, respectively. The branching occurs in

relation to parameter changes, the tracks represent the iso-values -150, -250 and -350. The cursors allow to control the linked nodes. The integration cursor controls a time-dependent node, in this example the integration time of the particles. The integration cursor which steers the integration time of the pathlines is inactive and greyed out. The two active cursors (slicer cursor and the overview cursor) are grouped with an offset of 10 timesteps. Moving one also moves the other. Below each view special view-navigators are shown which are simplified World Lines where branching events are shown by small red glyphs.

8 Conclusions

In 1992 it has been said regarding data-flow based systems that "there is no more effective solution for the visualization industry than these systems" [RBM*92]. In this paper we have demonstrated that this statement is still valid today, even for complex, interactive, exploratory problem solving environments where data can be created on-the-fly.

We do not handle data decomposition and streaming. This is basically an orthogonal problem for which data-flows have already been shown to be capable of [CBB*05, BP08, VOS*10]. We are currently working on implementing parallelization, data storage management (disk, CPU, GPU) and data domain decomposition automatically. In general more research to optimize scheduling of CPU+GPU resources will be necessary if we want to make the processing power of the GPU available to a wide range of node developers. For practical use it might be necessary to allow database repository type of data storage for storing results long term. Currently it is not possible to create new nodes and links automatically via interaction with the World Lines. For example when cloning a cursor, a monitor is automatically added. In future work we would like to explore a machine assisted analysis process which integrates machine learning [FWG09], parameter space sampling [BM10] and interactive visual exploration for multiverse analysis.

References

[AT95] ABRAM G., TREINISH L.: An extended data-flow architecture for data analysis and visualization. In *Proceedings of the 6th conference on Visualization* (1995), pp. 263–371.

[BCC*05] BAVOIL L., CALLAHAN S., CROSSNO P., FREIRE J., SCHEIDEGGER C., SILVA C., VO H.: Vistrails: enabling interactive multiple-view visualizations. In *Visualization, 2005. VIS 05. IEEE* (2005), pp. 135 – 142.

[BGMT07] BIDDISCOMBE J., GEVECI B., MARTIN K., THOMPSON D.: Time dependent processing in a parallel pipeline architecture. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1376–1383.

[Bis09] BISHOP R. H.: *LabVIEW 2009 Student Edition*. Prentice Hall, 2009.

[BM10] BRUCKNER S., MÖLLER T.: Result-driven exploration of simulation parameter spaces for visual effects design. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1467–1475.

[BP08] BOTHA C. P., POST F. H.: Hybrid scheduling in the devide dataflow visualisation environment. In *Proceedings of Simulation and Visualization: SimVis 2008* (2008).

[CBB*05] CHILDS H., BRUGGER E., BONNELL K., MEREDITH J., MILLER M., WHITLOCK B., MAX N.: A contract based system for large data visualization. *Visualization Conference, IEEE 0* (2005), 25.

[CFS*06] CALLAHAN S. P., FREIRE J., SANTOS E., SCHEIDEGGER C. E., SILVA C. T., VO H. T.: Managing the evolution of dataflows with vistrails. In *Proceedings of the 22nd International Conference on Data Engineering Workshops* (2006), ICDEW '06, IEEE Computer Society, pp. 71–79.

[CSS] Cascading style sheets `http://www.w3.org/Style/CSS/`.

[FTIN97] FUJISHIRO I., TAKESHIMA Y., ICHIKAWA Y., NAKAMURA K.: Gadget: goal-oriented application design guidance for modular visualization environments. In *Visualization '97., Proceedings* (1997), pp. 245 –252.

[FWG09] FUCHS R., WASER J., GRÖLLER M.: Visual human+machine learning. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1327–1334.

[Gan79] GANE C. P.: *Structured Systems Analysis: Tools and Techniques*. Prentice-Hall Software Series, 1979.

[Hae88] HAEBERLI P. E.: Conman: A visual programming language for interactive graphics. In *Computer Graphics* (1988), pp. 103–111.

[KFV*05] KLEEFSMAN K. M. T., FEKKEN G., VELDMAN A. E. P., IWANOWSKI B., BUCHNER B.: A volume-of-fluid based simulation method for wave impact problems. *Journal of Computational Physics 206*, 1 (2005), 363–393.

[Kil73] KILDALL G. A.: A unified approach to global program optimization. In *Proceedings of the 1st annual symposium on Principles of programming languages* (1973), POPL '73, ACM, pp. 194–206.

[KSC*08] KOOP D., SCHEIDEGGER C., CALLAHAN S., FREIRE J., SILVA C.: Viscomplete: Automating suggestions for visualization pipelines. *Visualization and Computer Graphics, IEEE Transactions on 14*, 6 (2008), 1691 –1698.

[MT05] MAUBACH J., TELEA A.: The NUMLAB numerical laboratory for computation and visualisation. *Computing and Visualization in Science 8* (2005), 1–17.

[Mye90] MYERS B. A.: Taxonomies of visual programming and program visualization. *Journal of Visual Languages and Computing 1* (1990), 97–123.

[Par99] PARKER S. G.: *The SCIRun Problem Solving Environment and Computational Steering Software System*. PhD thesis, University of Utah, School of computing, 1999.

[RBM*92] RIBARSKY W., BROWN B., MYERSON T., FELDMANN R., SMITH S., TREINISH L.: Object-oriented, dataflow visualization system-a paradigm shift? In *Proceedings IEEE Visualization* (1992), pp. 384 –388.

[SLA*09] SANTOS E., LINS L., AHRENS J., FREIRE J., SILVA C.: Vismashup: Streamlining the creation of custom visualization applications. *Visualization and Computer Graphics, IEEE Transactions on 15*, 6 (2009), 1539 –1546.

[SML96] SCHROEDER W. J., MARTIN K. M., LORENSEN W. E.: The design and implementation of an object-oriented toolkit for 3d graphics and visualization. In *Proceedings of the 7th conference on Visualization* (1996), pp. 93–81.

[Tel00] TELEA A. C.: *Visualisation and Simulation with Object-Oriented Networks*. PhD thesis, Eindhoven University of Technology, 2000.

[TvW00] TELEA A. C., VAN WIJK J.: Smartlink: An agent for supporting dataflow application construction. In *Proc. IEEE VisSym 2000 Symposium* (2000).

[UFK*89] UPSON C., FAULHABER JR. T., KAMINS D., LAIDLAW D. H., SCHLEGEL D., VROOM J., GURWITZ R., VAN DAM A.: The application visualization system: A computational environment for scientific visualization. *IEEE Comput. Graph. Appl. 9* (July 1989), 30–42.

[Vis] Visdom integrated visualization framework `http://www.visdom.at`.

[VOS*10] VO H. T., OSMARI D. K., SUMMA B., COMBA J. L. D., PASCUCCI V., SILVA C. T.: Streaming-enabled parallel dataflow architecture for multicore systems. *Computer Graphics Forum 29(3)* (2010), 1073–1082.

[WAH*09] WOLTER M., ASSENMACHER I., HENTSCHEL B., SCHIRSKI M., KUHLEN T.: A time model for time-varying visualization. *Computer Graphics Forum 28(6)* (2009), 1561–1571.

[WFR*10] WASER J., FUCHS R., RIBIČIĆ H., SCHINDLER B., BLÖSCHL G., GRÖLLER E.: World lines. In *IEEE Transactions on Visualization and Computer Graphics 15(6), to appear* (2010).

[Wor] Video world lines `http://visdom.at/media/slides/world_lines_final.mp4`.