Multiverse data-flow control

Benjamin Schindler, Jürgen Waser, Hrvoje Ribičić, Raphael Fuchs, Ronald Peikert, *Member, IEEE*

Abstract—In this paper we present a data-flow system which supports comparative analysis of time-dependent data and interactive simulation steering. The system creates data on-the-fly to allow for the exploration of different parameters and the investigation of multiple scenarios. Existing data-flow architectures provide no generic approach to handle modules that perform complex temporal processing such as particle tracing or statistical analysis over time. Moreover, there is no solution to create and manage module data which is associated with alternative scenarios. Our solution is based on generic data-flow algorithms to automate this process, enabling elaborate data-flow procedures, such as simulation, temporal integration or data aggregation over many time steps in many worlds. To hide the complexity from the user, we extend the World Lines interaction techniques to control the novel data-flow architecture. The concept of multiple, special-purpose cursors is introduced to let users intuitively navigate through time and alternative scenarios. Users specify only what they want to see, the decision which data is required is handled automatically. The concepts are explained by taking the example of the simulation and analysis of material transport in levee-breach scenarios. To strengthen the general applicability, we demonstrate the investigation of vortices in an offline-simulated dam-break data set.

Index Terms—time-varying data, visual knowledge discovery, visualization system design, temporal navigation, multiple simulation runs, data-flow, dynamic data management

1 Introduction

Decision support systems with integrated simulation and visualization capabilities are rapidly gaining importance in fields such as industrial engineering and response planning. However, the decisions the user can make vary greatly depending on the use case and on local conditions. An inflexible, hard-to-modify system requires much work to be done every time the decision space changes. Modern data-flow systems avoid this issue by offering a modular architecture that allows the user to add or change simulation and visualization components through an interactive flow diagram [1], [2]. In such a graphical interface, the modules (nodes) are represented as boxes that have input and output ports (Figure 1a). By changing the connections between ports and adding new nodes, the system can be adapted as the situation requires it.

Adding simulation capabilities to data-flow systems poses interesting new challenges. If a certain simulation state is requested, the system needs to make sure that the previous states are already computed. This additional dependency seems unnatural for data-flow systems because the data-flow models computation order through connections only. A similar challenge arises with time-dependent visualizations. Pathlines, which can be used to study material transport, require a whole time span from its inputs. Again, a simple data-flow connection insufficiently represents the additional temporal dependencies. The situation gets even more complicated if decision-support systems are used to analyze alternative scenarios. The results of a module can depend not only on multiple time values, but also multiple alternative parameter values.

- *B. Schindler, R. Fuchs and R. Peikert are with the Scientific Visualization Group, ETH Zürich*
- *J. Waser and H. Ribičić are with the VRVis Vienna*

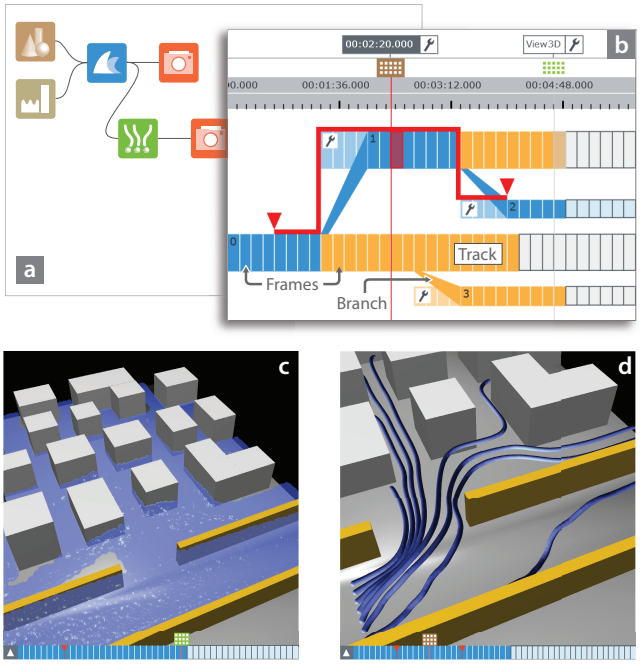


Fig. 1. Investigation of material transport through water in (c) a levee-breach scenario. (b) World Lines navigate the underlying (a) data-flow nodes across time and multiple simulation runs (visualized as tracks) to calculate (d) pathlines that describe the transport phenomena in alternative scenarios.

In this paper, we show how we solve all of the above problems using generic data-flow algorithms. The user can introduce various simulation, analysis and visualization functionalities as simple data-flow modules, with the arising dependencies transparently resolved behind the scenes. For example, if asked to produce pathlines, the system can automatically generate the data required by using an in-

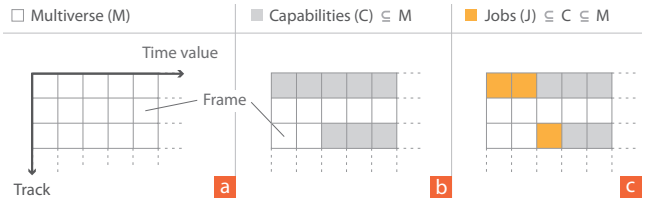


Fig. 2. Definitions. A frame is defined by a time value and a track. (a) The multiverse is the space of all frames. (b) The capabilities of a node is a subset of the multiverse and consists of all frames the node can produce data for. (c) The jobs of a node are a subset of the capabilities the node has to generate data for.

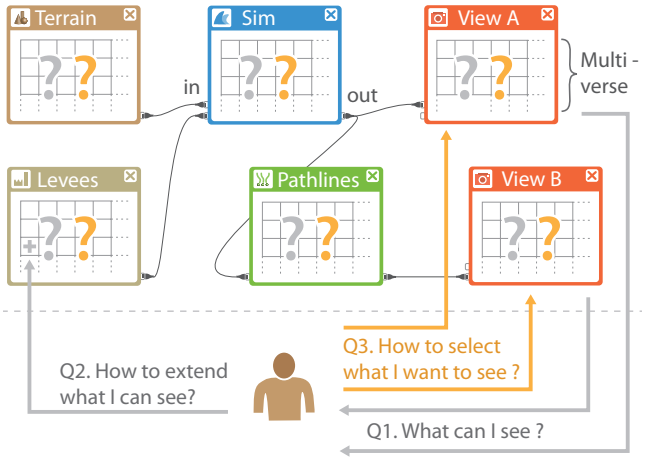


Fig. 3. Challenges when navigating a complex data-flow system. (Q1) The system lacks a visualization to report what the view nodes can generate. (Q2) The user needs control mechanisms to increase the number of data items a node can produce, e.g., to generate an additional data item in the levees node that models a levee-breach. (Q3) Intuitive navigation concepts are missing to let users select what they want to see. The data-flow network corresponds to the setup in Figure 1 which has the task to generate pathlines.

tegrated simulation on the fly. The same algorithms allow the World Lines [3]–[5] (Figure 1b) to be used in classical visualization scenarios by loading pre-simulated data.

We exploit the data-flow modularity to extend a levee-breach scenario (Figure 1c) with analysis and visualization nodes to investigate the material transport through the water using pathlines (Figure 1d). This data-flow setup is used as an example throughout this paper in order to explain the presented concepts.

1.1 Problem Description

During the execution of a data-flow system, each node calculates and outputs one or more data items. A data item can be requested for every *frame*, a coordinate consisting of a time value and a track which identifies a parallel world. The space of all frames is a *multiverse* and can be represented by a two-dimensional grid (Figure 2a), as we'll do in the figures of this paper.

Depending on its type, settings and position within the network, a node is capable of producing data for a particular subset of the multiverse. We refer to this subset as the *capabilities* of a node (Figure 2b). For example, the terrain node in Figure 3 generates only one data item, which represents the static, geometric boundary conditions of the simulation. The time-dependent simulation and pathlines nodes are capable of generating data for several frames. The set of frames these nodes can produce depends not only on the node itself, but also on the frames for which input data is present. Thus, we require a data-flow algorithm to compute what the nodes can produce, and a visualization to present these capabilities to the user (Q1).

The investigation of alternative scenarios requires the ability to manually extend the capabilities of a node (Figure 3, Q2). By default, the levees node in Figure 3 calculates one geometric data item that models the intact levees of the city. To manually enforce a levee breach, the node needs to be able to calculate an additional data item. To be able to explore alternatives as needed, a generic system for capabilities extension is required.

With the evaluation and visualization of capabilities at hand, we further require interactive navigation concepts that let users select what they want to see (Figure 3, Q3). For this purpose, a mechanism is needed that automatically assigns *jobs* to the nodes. A job is a subset of the capabilities which a node has to compute data for (Figure 2c). In existing systems, the user was responsible for assigning jobs to each of the involved nodes manually.

1.2 Proposed Solution

Our solution is based on two additional data-flow passes used to evaluate node capabilities and to enable job assignment. The user controls the system via World Lines, which operate on top of the data-flow. As illustrated in Figure 4, the multiverse can be mapped onto the visual entities representing frames that World Lines provide. To show the user what can be calculated and visualized, the framework first determines the capabilities of all the view nodes. In this computation, the data-flow is traversed in topological order from the source nodes to the view nodes. This downstreaming of capabilities is illustrated through grey arrows on the data-flow connections in Figure 4. After the capabilities of all view nodes in the network are calculated, their union is highlighted in World Lines.

To allow the user to study alternative scenarios, the node capabilities can be extended by providing new settings. The new settings can be introduced at the level of an entire track or individual frames. For example, in Figure 4 both the levees and the terrain are defined by a single settings object valid across the whole multiverse. Should the user wish to experiment by breaching a levee, this can be done by branching a new track off the parent track (Figure 5, S2). The user specifies a new set of settings describing the breach, valid across the newly created track. The levee node can then produce two distinct data items, one where the levees are intact, and another which models a breach.

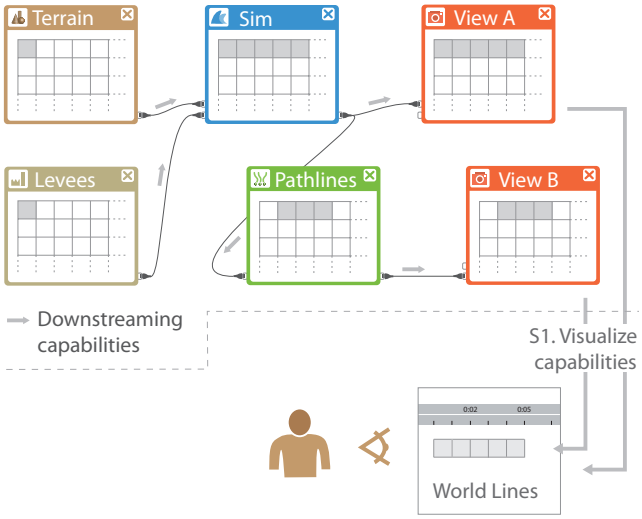


Fig. 4. Proposed solution, part one. (S1) A data-flow downstream process (grey arrows on connections) is in charge of computing node capabilities. World Lines visualize the unified capabilities of the view nodes, i.e., all frames the view nodes can show.

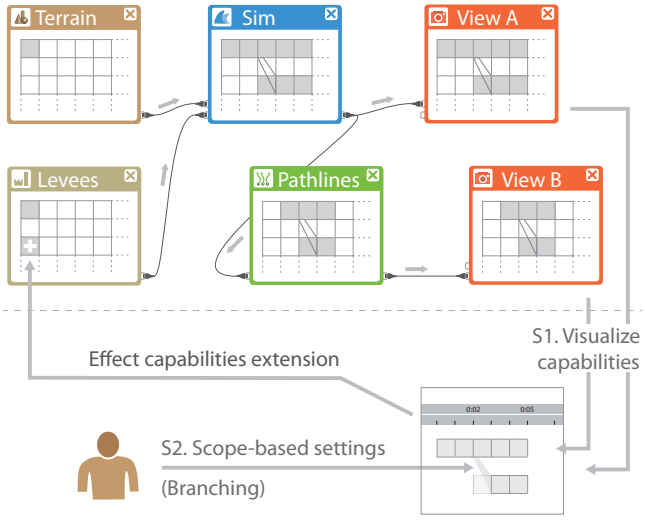


Fig. 5. Proposed solution, part two. (S2) The capabilities of a node can be extended through the specification of track settings. Interactively, this is accomplished via branching in World Lines. Here, the user branches with a parameter of the levees node to model a breach. The extended capabilities are downstreamed the data-flow to update the World Lines view with the newly created track.

As can be seen in Figure 5, the extension of the capabilities triggers another downstreaming pass, causing the World Lines to update and show the newly available capabilities.

Once the capabilities are shown, the user can perform interactive job assignment via multiple cursors (Figure 6, S3). For each view node, the World Lines view provides an individual cursor. The position of the cursor determines which frame or frames a view node is to visualize. These frames are the view nodes' jobs. The jobs of all other

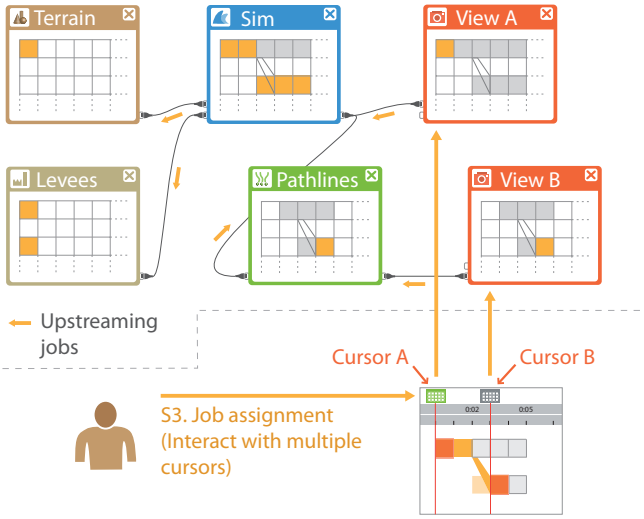


Fig. 6. Proposed solution, part three. (S3) The user tells the view nodes what to calculate through interaction with multiple cursors. We utilize a data-flow upstream process (yellow arrows on connections) which assigns jobs to other nodes automatically.

nodes are determined by a data-flow upstream process which guarantees that each node in the network generates the required information. In this process, the data-flow is traversed in topological order from the view nodes to the source nodes. This upstreaming of jobs is illustrated through yellow arrows on the data-flow connections in Figure 6. Once the upstreaming is complete, a standard data-flow pass can be done to calculate all the necessary data items, and display the results of the calculation.

To demonstrate our solution, we consistently use the levee-breach scenario throughout the paper. This example is later joined by two more case studies - a more complex real-world simulation scenario and a study of vortices in pre-simulated dam-break data loaded from files.

The contributions of this paper can be summarized as follows:

- A generic data-flow system which combines modules that perform complex, temporal processing such as simulation, integration and aggregation across time and alternative scenarios.
- Interactive data-flow navigation through World Lines using multiple special-purpose cursors.
- Data-flow algorithms for automatic node configuration according to user requests.
- A multiverse hierarchy of settings and computed data to enable flexible experimentation with all parameters in a data-flow setup.
- Providing a unified approach to analyzing both pre-computed data and simulated data produced on-the-fly, using World Lines as the interface.

2 Related Work

In 1989, Upson et al. [2] published a seminal paper on the data-flow based visualization system AVS [6]. In the

following years, this topic received a surge of attention and several large systems besides AVS, such as IRIS Explorer [7], the Visualization Data Explorer [8] or VTK [9] have been developed. The data-flow model is based on a topologically sorted graph. To generate visualizations, the nodes are *executed* in a downstream process, where data flows from the source nodes to the sink nodes. Each visited node is in charge of an independent sub-process [10]. From its input ports, a node fetches all data that is required to compute the results of the sub-process. The resulting data items are passed on to connected nodes via output ports.

Temporal processing in data-flows In its original design [2], the data-flow model cannot encompass complex algorithms that do any temporal processing such as particle tracing or statistical analysis over time. However, the management of time-dependent data in data-flows has seen little attention. AVS [6] and IRIS Explorer [7], for example, have no explicit model for handling time. Instead, nodes can have a time-step setting which calls for a specific implementation to process different time values, e.g., to load data from files according to a user-defined time step. In this data-driven approach, only one time step can be processed at a time and has to be forwarded by one or more source nodes. Amira [11] and Avizo [12] support temporal processing through a global time-value setting which is automatically assigned to all nodes. In case of intricate data-flows, where nodes require access to multiple time steps at once, this system quickly reaches its limits.

VTK [9], including VisIt [13] and Paraview [14], are based on a demand-driven pipeline as opposed to the data-driven model where the data only flows downstream. In the demand-driven approach, users can request data from the sink nodes. This information is then communicated to nodes that are further upstream through an additional data-flow process. Childs et al. [15] exploit this technique to optimize large data visualization, loading only those parts of the data which are required to visualize the requested features. Biddiscombe et al. [16] extend the VTK data-flow with a demand-driven approach to support nodes which perform complex temporal processing. In this system, nodes can request – and consequently access – multiple data items of different time steps from nodes that are further upstream. To realize this, the authors suggest a combination of downstream and upstream processing. We improve this work by adding formalisms to the data-flow process, resulting in a general model which requires very little developer intervention. This leads to numerous improvements, including the ability to operate in a multiverse setting where data from alternative scenarios is involved.

Interactive navigation in time The time slider [17] is the predominant interaction element for time navigation in media players and audio or video editing tools. This component has been largely adopted for time navigation in scientific visualization systems. These systems often provide multiple coordinated views of the same data set to let users productively combine the information gathered from different views [18]. There are different approaches to accomplish a coordinated time navigation among the linked views, if the underlying logic is a data-flow model. To achieve time synchronization, the time parameter can be global or can be converted to a node connector and streamed down the nodes [6]. Voreen [19] relies on a property-linking mechanism to achieve time synchronization between views. In general, there is no visual interface to achieve a more sophisticated, synchronized navigation behavior between different views, e.g., to enforce a temporal offset between two views to study the evolution of a tsunami wave. The comparative analysis of multiple simulation runs could benefit from a time-value synchronization, if each view is associated with one particular simulation run, enabling a direct comparison of alternative outcomes through navigation in time.

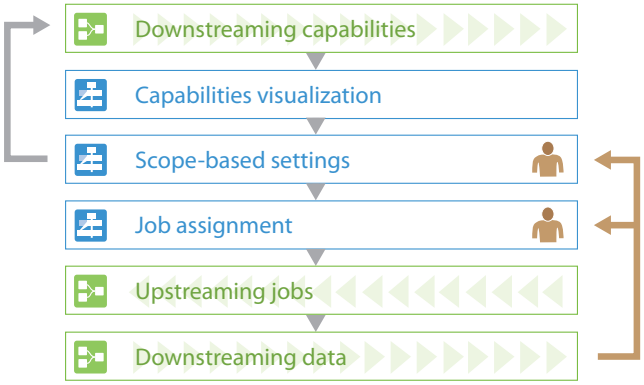


Fig. 7. Flow chart of the basic steps in the navigation cycle. Image production involves three data-flow traversals (green) and three steps in the World Lines view (blue). The only tasks the user performs are cursor movement (job assignment) and modification of settings through World Lines.

Previous work with World Lines involved the extension of the classic data-flow paradigm with a special meta-flow used to communicate settings between nodes [20]. This mechanism and the algorithms of this paper complement each other, but have orthogonal purposes. The meta-flow emulates only the most basic features of the presented algorithms by sharing time values between nodes.

3 Overview of the Navigation Cycle

In this section, we provide an overview of the most important components in the multiverse navigation cycle. The chart in Figure 7 depicts an operational step-by-step work-flow of the data-flow tasks (green boxes) and World Lines tasks (blue boxes). The terms used in the paper are summarized in Table 1.

Downstreaming capabilities The multiverse navigation always begins with this data-flow process which evaluates the computable frames of all view nodes. The capabilities of a node are calculated as an intersection of the capabilities of its input nodes. This intersection is further modified by nodes which perform temporal processing. For example, a simulation node appends frames to show how far the simulation can proceed.

TABLE 1
Terms used in the paper.

node	data-flow module with input and output ports
input nodes	all nodes directly connected to the input ports
output nodes	all nodes directly connected to the output ports
settings object	collection of node parameters
track	user-interface element repr. an alternative scenario
frame	coordinate consisting of a time-step and a track, also the user-interface representation
scope	data structure repr. either a frame (per-frame scope) or a track (per-track scope)
scope set	collection of scopes
capabilities	scope set repr. data a node is able to produce
jobs	scope set repr. data a node is required to produce
cache	scope set repr. data a node has produced
settings set	scope set repr. all settings objects of a node

Capabilities visualization The capabilities of all view nodes are transmitted to the World Lines view. World Lines employ a frame-wise representation to visualize the received capabilities.

Scope-based settings In this optional step, users configure the settings of nodes. If the user creates a new branch or a setting influences a node's capabilities, the downstreaming of capabilities is triggered again.

Job assignment Each view node is associated with a cursor in the World Lines view. If a cursor moves, one or more frames from the related view node have to be computed.

Upstreaming jobs The user request is communicated from the view node upstream to all connected nodes in the data-flow.

Downstreaming data The final step concerns the standard data-flow execution where each node performs its allocated jobs to compute all requested data. The navigation cycle starts anew if the user assigns additional jobs using cursors or modifies node settings.

4 Scopes

Before explaining the data-flow algorithms in more detail, it is necessary to introduce the concepts of *scopes* and *scope sets*. As mentioned before, data-flow connections lack any sort of temporal or parameter-related information. To remedy this, almost all the objects in our system have additional information attached to them in the form of scope-based structures. Since the design of these structures has a great effect on the performance and abilities of our algorithms, they will be explained here in detail.

4.1 Scope and Scope Set

Almost all the objects in our system can be said to occupy some subset of the multiverse or originate in it. Were it not for the fact that some changes can be applied to entire tracks only, we could use frames as the building blocks of our system. Instead, we define a scope as the basic unit in the system, and we define it dually: A scope can be either per-frame or per-track. Per-frame scopes are defined by a time value and a track and functionally identical to a frame. On the other hand, a per-track scope encompasses all the frames belonging to the same track. This definition

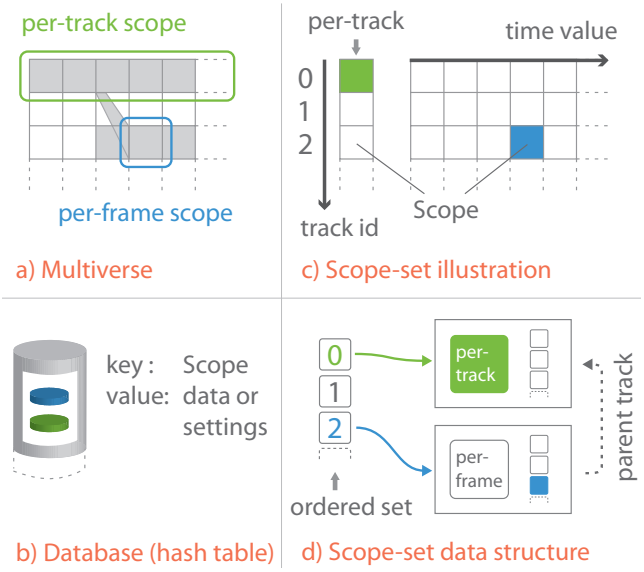


Fig. 8. Relations between the multiverse representation (a) and internal data structures (b,c,d). (b) Data is stored at different scopes. (a) A per-frame scope (blue) is defined by a particular time value and track, a per-track scope (green) covers all time values of a track. (c) A scope set is a collection of scopes, (d) internally represented by an ordered set of track structures.

allows us to treat all sorts of changes the user can introduce to the system the same way in the algorithms that follow. Examples of scopes are shown in Figure 8a.

Scope sets are defined as a collection of unique scopes. They can contain both per-track and per-frame scopes, as can be seen in Figure 8c which illustrates how we represent scope sets for the explanation of our algorithms. Scope sets are used to represent how object values are distributed across the multiverse. The values are not stored in scope sets - they only serve as a repository of scopes which can be used as a key to retrieve the actual object values from hash tables (Figure 8b). For example, the scopes can describe capabilities, but also data items which a node has computed.

The implementation of scope sets is important to the performance of our algorithms, making it necessary to stop considering a scope set a black box. To show the example of the internal organization of a set, we will use the example scope from Figure 8c. The insides of this particular set is shown in Figure 8d. The time-values of all the per-frame scopes belonging to the same track are stored in a single ordered set. This set, along with a boolean for the per-track scope, is stored in a map. The map associates track identifiers with the track-specific data. To see why sets are organized as they are, it is time to define the basic operation that can be performed on scope sets.

4.2 Hierarchical Lookup

The reason why it is possible to have multiple data or setting values associated with a single node stems from the hierarchical nature of World Lines. To explore different

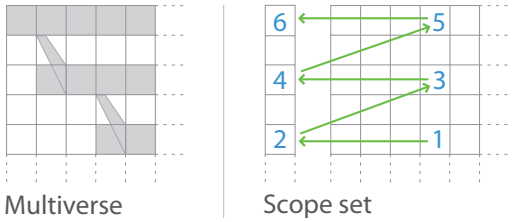


Fig. 9. An example for a hierarchical lookup chain in a scope set. The input scope is assumed to be at position 1. The scopes are inspected in the given order

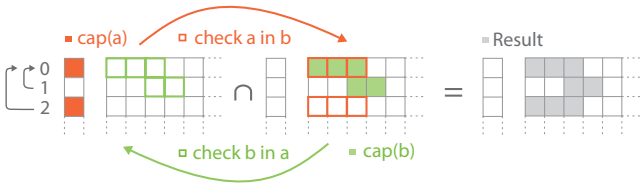


Fig. 10. Intersection of two scope sets, cap(a) and cap(b), implemented as a mutual lookup. The arrows on the left denote the track hierarchy.

simulation or visualization parameters, the user can create new tracks by branching off existing ones. However, it would be impractical if unaffected nodes would perform unnecessary calculations recreating existing data. In the flooding scenario, for example, it is not necessary to recalculate the terrain for every parameter modification. In this case, the terrain should be fetched from the scope of the root track, no matter what scope is currently being handled by the data-flow. This is why every node has a hierarchy of different values it can produce represented by a scope set.

Algorithm 1 ScopeSet::hierarchicalLookup
input: Scope scope
output: Scope resultScope

```
1:  track ← scopeSet.getTrack(scope.trackId)
2:  while scopeSet.exists(track) do
3:      if scope.type==per-frame and
        track.hasTimeValue(scope.timeValue) then
4:          resultScope.trackId ← scope.trackId
5:          resultScope.type ← per-frame
6:          resultScope.timeValue ← scope.timeValue
7:          return resultScope
8:      else if track.containsPerTrackScope then
9:          resultScope.trackId ← scope.trackId
10:         resultScope.type ← per-track
11:         return resultScope
12:     end if
13:     track ← track.parent
14: end while
15: throw("Scope not found")
```

When producing new data items from input nodes whose scopes are not identical to the processing node, a *hierarchical lookup* must be performed to retrieve a *matching scope*. Figure 9 provides an example for such a lookup chain. At the beginning, the lookup algorithm searches for the given scope directly (1). If nothing is found, the lookup continues in the scope of the track associated with the given scope (2). From there, the algorithm searches in the parent track, starting at the scope which has the same time value as the given one (3). If this scope is not present either, the scope of the parent track is inspected (4). This recursive search goes up the track hierarchy until a matching scope is found (5,6). The implementation of the lookup procedure is presented in Algorithm 1. The runtime of the lookup is of order $O(n \cdot \log m)$ where n is the number of tracks and m the number of time values associated with a track. It should be noted that this is a worst case analysis. In common scenarios, the factor n will behave more like $\log n$ because the World Lines tree usually has logarithmic depth.

Our choice for the lookup chain is motivated by the need for flexible parameter experimentation. In Section 7 we will demonstrate how to exploit this feature for the animation of parameters which are valid across many tracks. In the following sections, we explain how to use scope sets to evaluate capabilities, to assign jobs and to cache computed data.

5 Capabilities

The capabilities of a node consists of all scopes for which the node is able to produce data. While the capabilities of all nodes have to be calculated, the user is interested in the capabilities of the view nodes only - the end results that can be produced.

5.1 Downstreaming Capabilities

At the beginning of the multiverse-navigation cycle, nothing is known about the results the nodes may produce. Source nodes can determine their capabilities based on the input data or settings used, but for all other nodes a downstream process must be performed. To calculate a node's capabilities, the capabilities of all of the input nodes need to be already calculated. Once prepared, they are combined with the following condition: Any scope that cannot be matched within the other input nodes' capabilities has to be discarded. An unmatched scope signifies that not all input nodes can provide data necessary to calculate the output at this scope and thus cannot be part of the node capability.

In mathematical terms, the operation can be expressed as the scope-set intersection

$$cap(node) = \bigcap_{n \in inputs(node)} cap(n). \qquad (1)$$

The intersection operation is performed pairwise, being both commutative and associative. Figure 10 illustrates an example to explain the intersection of two scope sets denoted as $cap(a)$ and $cap(b)$. The related track hierarchy comprises three tracks, both track 1 and track 2 are a direct branch from track 0. As shown in Figure 10, it is implemented as a two-way lookup. We attempt to match

Algorithm 2 ScopeSet Intersection ∩
input: ScopeSet set[0], ScopeSet set[1]
output: ScopeSet result

```
1:  result ← emptyScopeSet
2:  for i = 0 to 1 do
3:      other ← (i+1) mod 2
4:      for each scope ∈ set[i] do
5:          if set[other].hierarchicalLookup(scope) then
6:              result.insert(scope)
7:          end if
8:          if scope.type == per-track then
9:              track ← set[i].getTrack(scope.trackId)
10:             while set[other].exists(track.id) do
11:                 otherTrack ← set[other].getTrack(track.id)
12:                 for each otherScope ∈ otherTrack do
13:                     if otherScope.type == per-frame then
14:                         scopeToInsert ← otherScope
15:                         scopeToInsert.trackId ← scope.trackId
16:                         result.insertScope(scopeToInsert)
17:                     end if
18:                 end for
19:                 track ← track.parent
20:             end while
21:         end if
22:     end for
23: end for
24: return result
```

scopes of $cap(a)$ in $cap(b)$, and then scopes of $cap(b)$ in $cap(a)$. Every scope matched by a hierarchical lookup (not the found matching scope) is added to the resulting scope set (see also Listing 2, lines 4-6). The only exception to this rule are the per-track scopes. Should the lookup of a per-track scope fail, the scope is decomposed into per-frame scopes which are matched individually. Any of the matched per-frame scopes are added to the result as well. An example of this can be seen in Figure 10, where the lookups of the two track-global scopes in cap(a) fail in cap(b). However, the decomposition of the per-track scopes results in three successful lookups each, which are added to the result of the intersection. Due to the unlimited number of possible per-frame scopes, this view of the decomposition cannot be effectively implemented in practice. Our optimization is shown in Listing 2, line 8-21.

The runtime of the complete intersection is of order $O(2 \cdot n^2 \cdot m \cdot \log m + 2 \cdot n^2 \cdot m) = O(n^2 \cdot m \cdot \log m)$ where n is the number of tracks and m the number of time values in the larger of the two input scope sets. The first term accounts for the hierarchical lookups and the second term covers the special case.

5.2 Visualization of Capabilities

Upon completion of the capabilities downstreaming process, we need to report the results to the user. World Lines are ideally suited to visualize the per-frame scopes present inside the results. Figures 11a and 11b show screenshots

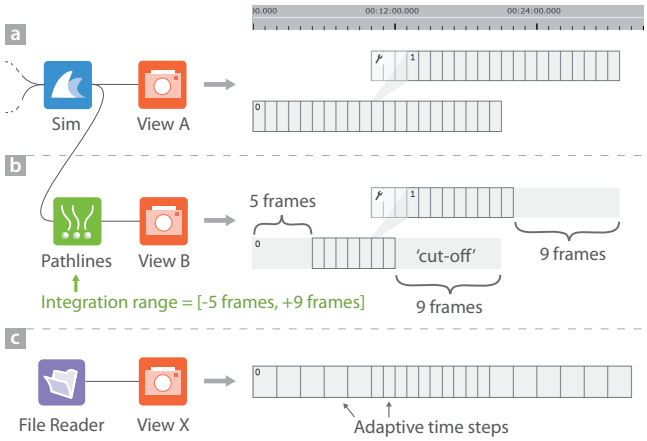


Fig. 11. Frame-wise representation of tracks to visualize capabilities. (b) Capabilities of node 'View B', which is connected to the pathlines node, contain less frames than the capabilities of (a) 'View A'. (c) Visualization of adaptive time steps loaded from external files.

of World Lines displaying the capabilities of view nodes. Tracks are always drawn according to when they start and end, but dark gray borders beginning at frames' time values highlight the subset of frames that can actually be calculated. Adaptive time steps, vital to the performance of some simulations, are easy to display in this way, as shown in Figure 11c.

There are cases when some frames cannot be calculated, such as the one involving the pathlines node in Figure 11b. This integration node requires access to a certain number of frames before and after the desired frame. Some frames at the beginnings and ends of tracks do not satisfy this condition, which shows as a lack of borders on the track. If view nodes from both Figures 11a and 11b are present, World Lines will highlight frames present in at least one capabilities, making the result look like Figure 11a.

6 Jobs

The jobs of a node are defined as all the scopes for which the node has to compute data. The user determines these for the view nodes using cursors. The necessary jobs of all other nodes are automatically computed in the correct order.

6.1 Navigation through Multiple Cursors

The notion of a cursor used to select which frame is visualized was introduced by the original World Lines [4]. The cursor determines the active frame, which is the desired job of the entire system of linked views. While this concept has served us well, it has its disadvantages, the primary being that it is not possible to navigate multiple views for comparison, each showing different frames. To overcome this limitation, we provide a cursor for every linked view, as can be seen in Figure 12.

As with the original cursor, the basic properties of the cursor include the *time value* and the *World Line*. The

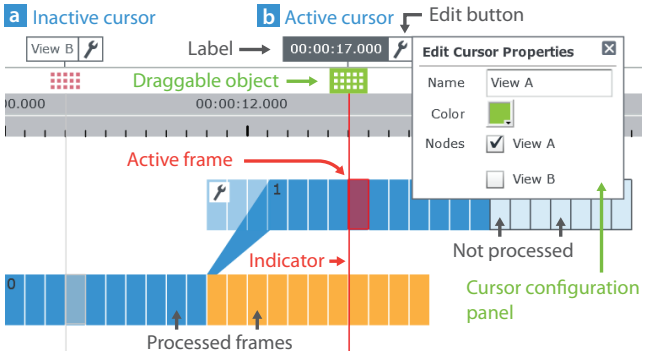


Fig. 12. Navigation using two cursors (a)(b). The World Line which belongs to the active cursor (b) is colored in blue. The cursors can be configured and customized.

World Line (blue) is a user-defined path through the tree of tracks. The cursor's horizontal position determines the time value, which defines the frame the cursor is on based on the information from the World Line. By marking the cursor as active and moving it, a new job based on the cursor's position is created and executed.

A property that was not present in the original cursor is the list of associated nodes. A single cursor can be configured to control multiple view nodes at once, issuing multiple jobs with each movement.

6.2 Upstreaming Jobs

After the jobs have been assigned to the view nodes, the jobs of all other nodes are automatically evaluated via an upstream process (Figure 6). For each of these nodes, the result depends on the jobs of all output nodes. In principle, a node has to compute data for any scope that is requested by any of its output nodes. In mathematical terms, this operation can be expressed as the scope-set union

$$jobs(node) = \bigcup_{n \in outputs(node)} jobs(n). \qquad (2)$$

As its name implies, the scope-set union simply merges all the scopes of its scope-set operands together, avodiding duplicate entries (see also Listing 3). The union produces a set of jobs we call output jobs, but which are not necessarily equivalent to the set of jobs the node must perform. Nodes with global scopes such as the terrain node only need to produce one data item regardless of which scopes this data item is used to calculate. The real scope set of jobs the node must perform is calculated by performing a hierarchical lookup of every output job in the node's capabilities, and inserting the matching capabilities scope into this scope set. In this way, the system is guaranteed to produce the minimum of calculations necessary, leaving no room for redundancy. The runtime of the scope-set union amounts to $O(n \cdot m)$, where n denotes the number of tracks and m the number of time values in the largest scope set. The ordered internal nature of the scope sets allows this operation to be performed quickly.

Algorithm 3 ScopeSet Union $\cup$
input: ScopeSet set[0], ScopeSet set[1]
output: ScopeSet result

```
1: result ← emptyScopeSet
2: for each scope ∈ [set[0], set[1]] do
3:     result.insert(scope)
4: end for
5: return result
```

6.3 Downstreaming Data

As soon as we know the jobs of all nodes, the requested data can finally be produced. In this step, the data-flow is traversed in a downstream process to execute all nodes. During execution, a node loops over its assigned jobs to produce data for each scope inside. If the node has to fetch data from an input node, another hierarchical lookup is performed to automatically retrieve the correct data items. This way, data at per-track scope can be fetched from input nodes even if the node is processing a per-frame scope. This is a common use case in the flooding scenario, where the simulation node has to fetch the static terrain and the track-specific levees no matter what simulation step is currently being handled (see also Figure 5).

We employ another scope set, the *cache* (Table 1), to remember all scopes for which the node has generated data. The actual computed data is cached inside a hashtable. This approach allows for arbitrary caching strategies which are useful for the task at hand. In our current implementation, all data is kept due to frequent navigation in time and across tracks. In order to prevent excessive memory use, the system is able to swap out computed data to the hard disk. When a node fetches data from its inputs, the data is pinned. Pinned data is loaded from the disk if necessary, and cannot be swapped out as long as it remains pinned.

7 Scope-Based Settings

To study alternative scenarios with World Lines, we require the ability to manually extend the capabilities of a node. In the use case of the levee scenario, we want to model different types of levee breaches. Each breach is modeled by branching off a new track. Consequently, additional settings objects are constructed and assigned to the related levees node. The settings objects are internally treated much as another input to the node. Like the data items produced by nodes, each of these receives a scope, and is stored in a hash table with the scope as a key. When a data value is calculated, the appropriate settings object is also retrieved via a hierarchical lookup.

The present settings affect the capabilities calculation. The scope set of the settings, the *settings set*, can cause additional capabilities scopes to be generated. Including the settings set into the capabilities calculation is simple:

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

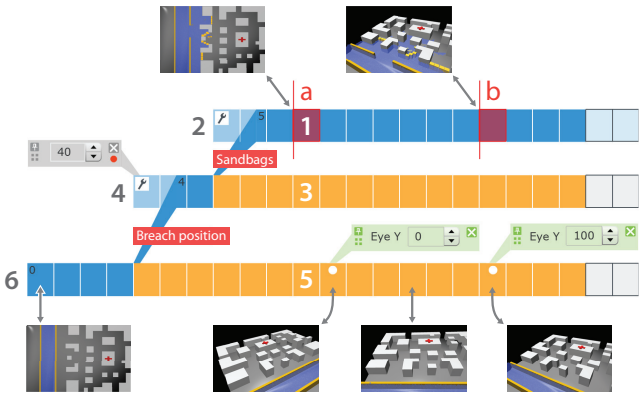


Fig. 13. Scope-based settings. The root track has key-frames (white dots) that comprise individual settings for the camera position. The numbers depict the search path for settings when calculating the frame indicated by cursor a. The camera setting of the root track is found. At cursor position b, the system fetches the camera location defined at the parallel frame in the root track.

When a new track is created using the World Lines, the settings set is updated, causing the desired capabilities extension.

Scope-based settings can be exploited to achieve interesting effects. With the proposed solution, settings can not only be assigned to whole tracks but to individual frames too. For example, in Figure 13, we experiment with time-varying camera positions. Two frames of the root track define explicit camera locations along the river. As a consequence of the lookup mechanism, these locations transfer into child tracks unless explicitly overridden. When navigating to the frame at the position of cursor b, the scene is rendered with the same perspective as defined for the parallel frame in the root track.

To achieve smooth transitions of settings between several frames, we adapt an interpolation technique based on key-frames [13]. The frames that explicitly define settings become key-frames, and the parameter values of frames inbetween are interpolated. This way, the user can animate parameters over time. In Figure 13, we apply this functionality to simulate a camera moving along the river. This animation is present in all the child tracks as well.

8 Scope-Set Modifiers

The nodes we have used to demonstrate the algorithms so far are simple nodes (Figure 14a). These nodes process values from only one scope at a time, and both the nodes and the node creators are oblivious to the actual scopes involved. However, with the presented framework, any kind of time-varying or track-dependent node can be implemented. As an example, we consider a time-interpolation node which is capable of producing new data for additional frames through temporal interpolation of its input data. The evaluation of capabilities according to Equation 1 is no longer sufficient.

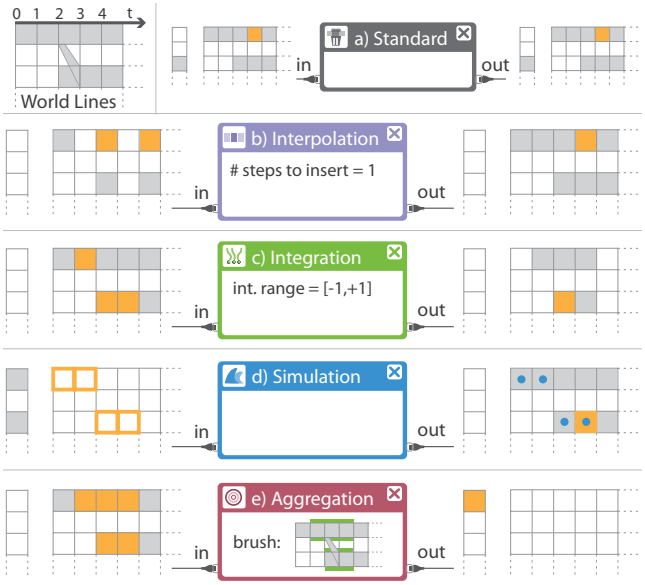


Fig. 14. Scope set modifiers are nodes which manipulate their computed capabilities (grey) and assigned jobs (yellow). A distinction between input (left) and output (right) capabilities and jobs has to be made. The top, left image depicts the World Lines structure involved. (a) In the standard case, input and output capabilities and jobs are equal. (b-e) Scope-set modifiers insert or remove scopes to have access to all required data.

The left scope set of Figure 14b illustrates the capabilities intersection the algorithms described so far would produce. The time-interpolation node is capable of producing more data. The node notes this by extending the capabilities set to the one shown on the right-hand side of the node. When one of the extended capabilities is selected as an output job, the node has to determine the input jobs necessary for interpolation manually rather than rely on lookups. The time-interpolation node is a typical example of a *scope-set modifier* which alters its capabilities and jobs.

Scope-set modifiers get a chance to alter their output capabilities and input jobs after they have been calculated using the normal algorithms. Although any type of scope-set modifier can be implemented in our framework, we have identified certain categories of these that we will describe in more detail. All nodes in a category share a certain type of behavior. This allows us to provide a base implementation which, much like a simple node, reduces the quantity of scope manipulation that the user must perform to a minimum. An explanation of how various categories work can be seen in Figure 14. Some of these also rely on special types of cursors to ease the manipulation of navigation-related node parameters. Figure 15 lists the available cursors together with example visualizations.

8.1 Integration

Integration nodes require a range of input data to generate their output. The aforementioned pathlines module is a typical example for this node category. In Figure 14c, the

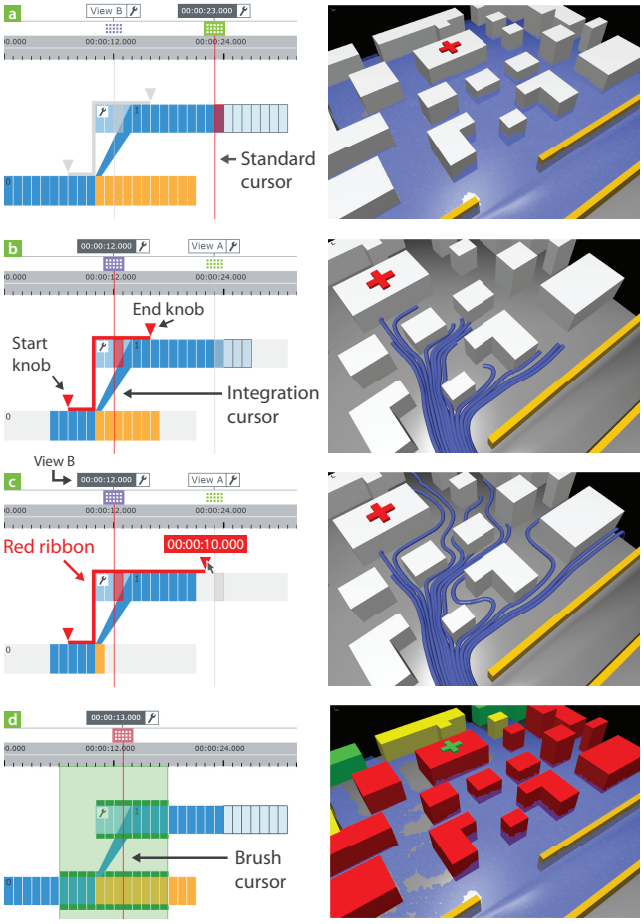


Fig. 15. Special purpose cursors. The right column displays renderings corresponding to the active cursors of the left column. (a) Standard cursor for navigation to a single frame. (b) Integration cursor to manipulate the integration range of the pathlines node. World Lines show the capabilities of 'View B' only. (c) The user increases the forward integration time, consequently less frames can be produced for 'View B'. The rendered lines become longer. (d) Brush cursor to select multiple frames for comparative analysis.

integration range is set to one time step in both forward and backward direction. Thus, the first scope of the root track and the last scopes of the two tracks are removed from the input capabilities to form the actual, modified capabilities. If jobs are assigned, the node inserts further scopes to request input scopes that cover the complete integration range.

The *integration cursor* can be used to visualize and control the integration range of an integration node. To achieve this, the cursor is attached a red ribbon that follows the World Line (Figure 15b) associated with the cursor. The ribbon forms a stair-case pattern if spanning multiple tracks of the World Line. The users can drag the knobs of the component to extend or shorten the integration range. In Figure 15c, the user increases the forward time. The image to the right shows the effects on the pathline rendering.

8.2 Simulation

Nodes of the simulation category can produce an unlimited amount of per-frame data. The data is produced from initial and boundary conditions supplied through input nodes. In Figure 6, the simulation node receives the boundary geometries from the terrain and the levees nodes. These dependencies often remain static within a track, i.e., a specific simulation run. The simulation node modifies the capabilities, outputting only per-frame scopes based on track start times and track durations. If jobs are assigned to the node, causality needs to be accounted for. All the frames causally preceding the desired frame have to be simulated before it, and are thus inserted into both the input and the output jobs. The input jobs in Figure 14d are represented by yellow borders since they are not a subset of the input capabilities, and the new output jobs are represented by blue dots.

8.3 Aggregation

The frame-wise representation of node capabilities in World Lines supports the interactive specification of a brush, i.e. a selection of frames, through a rubberband tool. Brushed frames receive a thick, green border on their top and bottom sides (see node inlay of Figure 14e). The brushing mechanism supports a comparative visual analysis of multiple frames. Aggregation nodes can receive a brush in order to compute an analytical or visual representative of all data items marked by the brush. The buildings in Figure 15d, for example, are colored according to the dangers posed to them in the selected frames.

Internally, the brush is converted into a scope set which replaces the original input jobs (see left of Figure 14e). This way, the node has access to all input-data items required to compute the aggregation result. The *brush cursor* can be used to move the brush along with the cursor (Figure 15d). This is useful if we want to monitor the temporal evolution of many tracks to, e.g., inspect the robustness of an ensemble simulation.

9 Advanced Navigation Tools

The data-flow algorithms we've described so far allow us to transparently and efficiently perform any calculation the data-flow allows us. We can use this property to design complex interactions without having to worry about what they will involve. As mentioned before, a cursor can be associated with a list of views in order to navigate their nodes synchronously. In this case, the affected nodes receive jobs for the computation of exactly the same frames. 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 several simulation scenarios in several views side by side (Figure 16).

For this purpose, the concept of nested cursor grouping has been developed. A cursor group is formed through selection with a rubberband tool. Every action that is applied

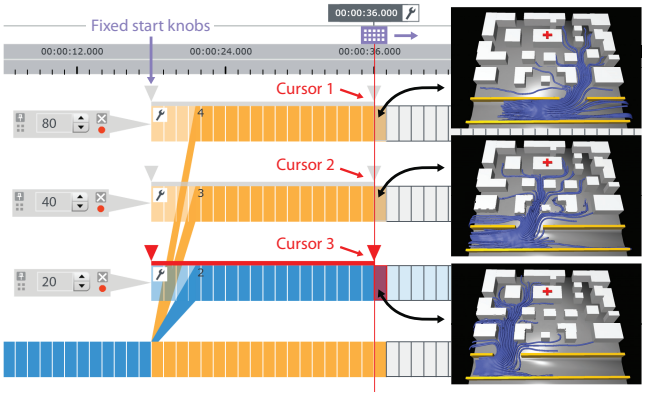


Fig. 16. Cursor grouping supports the synchronous comparison of material transport in three different levee-breach scenarios. The cursor group consists of three integration cursors, each of which has its own World Line. When moving the cursor group, the linked views display the temporal evolution of the pathlines synchronously.

to one cursor of the group modifies all other members too. 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. Figure 16 shows a group of three integration cursors to generate pathlines. All cursors share a common time value, but each of them has its own World Line to enable animated, comparative inspection of parallel worlds in three views. By moving the cursor group forward in time, we can compare the evolution of pathlines for three different levee-breach locations. Cursor groups can also be further nested, i.e., included into other groups in order to define complex view-linkage behavior. The important thing to note is that behavior such as this, where several frames are requested from a single node at once, would be incredibly difficult to perform with a normal data-flow system. By improving the underlying system, we have thus succeeded in improving the visualization capabilities of the system as well.

10 Case Study

To highlight the fact that our system is modular and useful for various problem types, we prepared two quite different case studies. Both of these use the data-flow algorithms to produce results that would have been significantly harder to obtain without the support our system provides.

10.1 Vortex Analysis of Pre-simulated Data

In this case study, we are given the data produced by a fluid simulation of a breaking dam consisting of 88 adaptive timesteps. The goal of the study is to analyze the vortices present in the fluid. Several visualization tools are present at our disposal: slices, pathlines and isosurfaces of λ_2 (a measure for vortex strength [21]). While we can combine these to create single images, with World Lines we can use the branching functionality to record the user work-flow during visual analysis. In this respect, the contribution is similar to the VisTrails approach [22] which utilizes a graph-based representation of the analysis steps. The basic purpose of the VisTrails history-tree is to return the system state to previous states which can be regarded as a sophisticated version of undo/redo functionality. There are no flexible, time-dependent navigation tools, nor is it possible to synchronously compare the results obtained at different steps of the recorded work-flow. These limitations are not given if using World Lines for the analysis of time-dependent data.

To begin with, we create two additional tracks in the World Lines view that comprise different isovalue settings. The two isovalues are chosen so that they highlight areas of different vortex strength. Figure 17c and Figure 17d visualize the isosurfaces (red) related to the two λ_2 values, giving a good idea of the vortex structures contained. In addition to the isosurfaces, we add the pathlines to help understand the flow surrounding the vortices better. The time span they involve is controlled and shown by the integration cursor. To provide a visual context, we render the free surface of the simulation data in a transparent blue color. For an overview on the area, we show a slice image of λ_2 in Figure 17b. This slice is also embedded in Figure 17a.

When navigating in time, we find that the vortex structures move significantly in vertical direction. To catch the features within the slice image, the vertical offset of the slicing plane needs to be adjusted. To remember the optimal positioning for each time value, we assign scope-based settings to the related frames of the root track (see key-frames in World Lines of Figure 17). Hereby, the slicing plane automatically assumes the optimal offset when navigating in time and across tracks.

The ability to record the exploration path using scope-based settings provides the basis for a thorough, comparative analysis of different parameters. In combination with the navigation through multiple cursors, we were able to quickly identify relevant vortices in the data. Moreover, results were easily reported to other users by navigating the recorded work-flow. By examining the work-flow, the user can see where and why the decision to use a different visualization parameter was made.

10.2 Movement of Swept-Away Vehicles

In the second case study, we decided to examine the problem of vehicle-caused danger during a flood. Our partners have provided us with the data necessary to create a scenario based on a real town's flood protection system, seen in Figure 18. A river prone to flooding flows through the town, and mobile walls are used to stop the water from spilling into the town.

However, these mobile walls are not perfect defenses, and should they fail or break, a torrent of water may spill into the town. The danger becomes much greater if heavy mobile objects such as cars are not removed, as the water may carry them at high speeds, causing much damage. The goal of the study is to try and determine which cars would be the most dangerous if not removed.

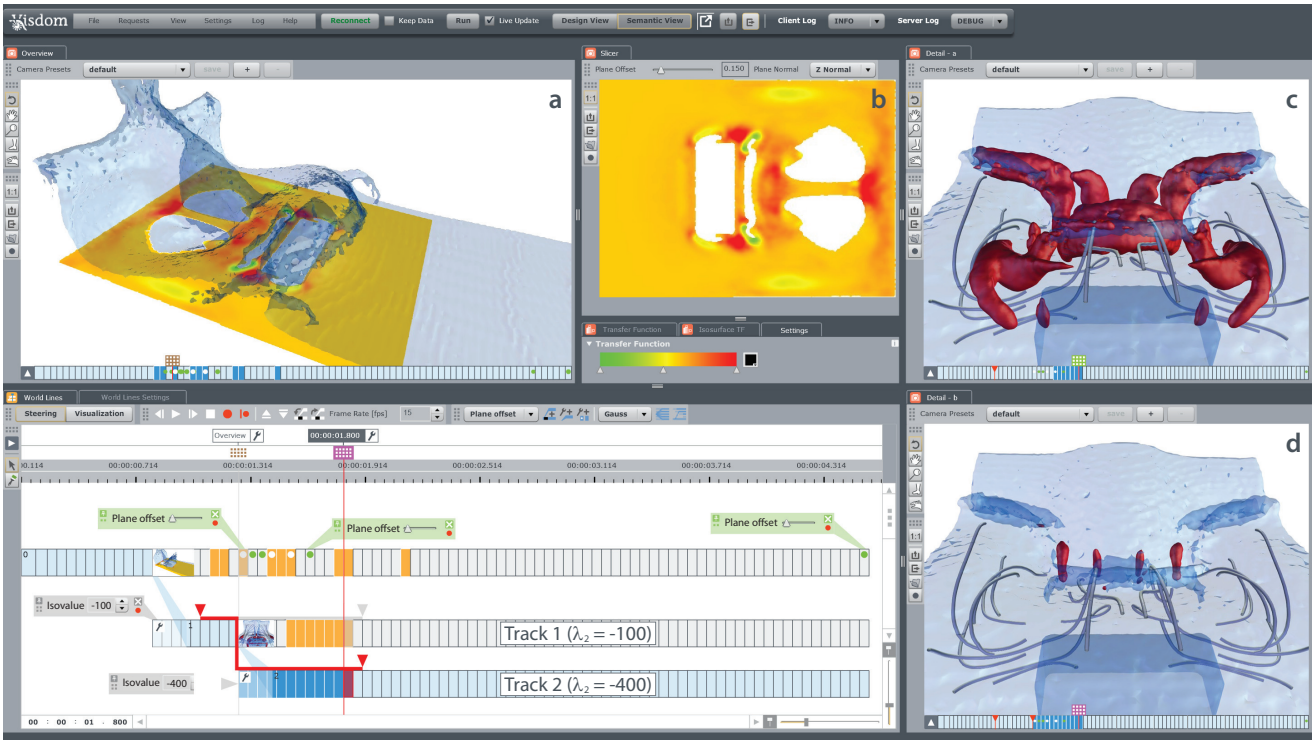


Fig. 17. Vortex analysis in a pre-simulated scenario of a breaking dam (screenshot). (a) The overview renders the free surface of the water while embedding (b) a slice of λ_2. Green colors highlight areas of high vortex strength, red colors depict low values. (c,d) Two views compare isosurfaces for different λ_2 values which are calculated for track 1 and track 2, respectively. Since vortices move vertically over time, we adjust the slice position through key-frames.

Given a fixed distribution of cars, we experiment with an unknown breach location to see the dangers posed by car movement. From the data-flow point of view, the data is provided via a simulation node which produces data on demand. To visualize how the cars move, we employ a node which displays the movement paths of the cars as a line, colored according to distance moved. This node is an integration node, requiring the simulation node to produce both future and past frames of the visualized frame. It then connects the individual car positions computed by the simulation using a B-spline. In that sense, the node does not perform any numeric integration, but from a data-flow perspective, it acts like an integration node. We employ three views in parallel, showing three breach locations, which can be seen in Figure 19.

As the user navigates across different time values, the same view produces all three images, causing three integrations to happen. In turn, these cause the simulation node to produce any data that is missing. All of these actions are invisible to the user, who moves a grouped cursor and sees the images update.

11 Evaluation

The evaluation of the algorithms described in this paper proved to be troublesome because of their nature – if they work as they are intended to, the user is not aware of their presence. The only manifestation of their use is the ease with which additional functionality is integrated. Therefore, we evaluate whether the various features our algorithms enable would be useful to domain experts, whether they would consider using them, and whether the more advanced interface elements are intuitive.

Our first interviewed expert is a renowned hydrologist performing research related to flood simulation. The discussion revealed that to him, modularity can be seen as the most important feature. Many institutes studying flood simulations have their own simulation engines, and would prefer to use a system capable of integrating them easily. Our base simulation implementation allows this to be done with a minimum amount of effort. Another feature that caught his eye are the pathlines, which he would like to use to visualize and analyze transport of material such as driftwood or leaking oil.

We also engaged a second expert, another hydrologist, specializing in flood modelling. The second study's multiple and integration cursors struck him as useful for the purpose of examining and comparing the data. The presentation aspect of the system shown in the first study also proved interesting, as a way of animating and showing the evolution of multiple attributes of interest, such as soil moisture. Both the pathlines and the car lines bear resemblance to his current field of study: the transportation of sediments along riverbeds. An echo of the first expert's comments could be heard when asked if he would consider using the system: yes, but on the condition that a specific simulation engine could be used.

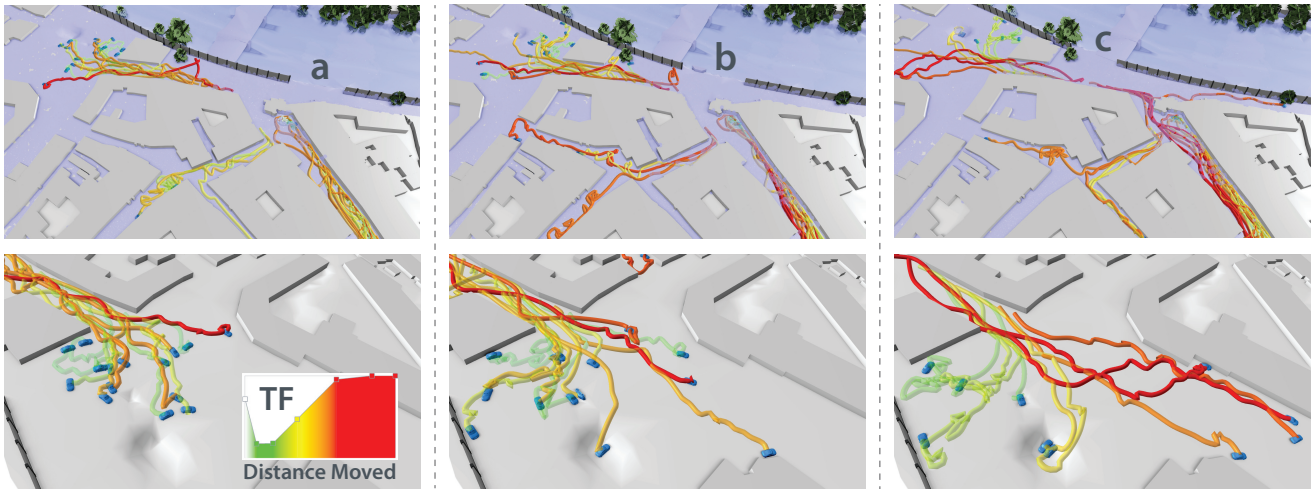


Fig. 19. The visualization of the car movement. The columns contain various breach positions (a,b,c), shown in the first row. The second row shows the effects of the movement after some time had passed.

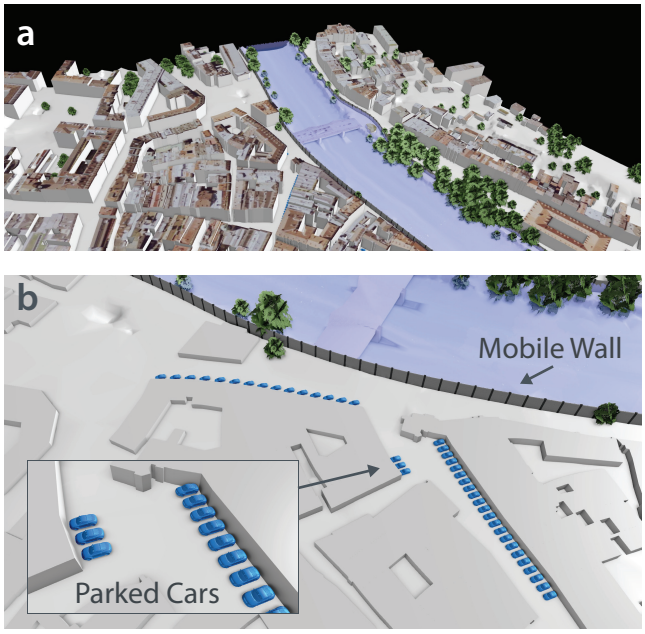


Fig. 18. The simulated scenario used in the second case study. The river that flows through the town (a) is held at bay by the mobile protection walls (b). The parked cars pose a great danger if still present when a flood happens.

Aside from the two in-depth conversations summarized here, the system and coincidentally the algorithms have been presented to a number of experts and potential users. They all liked both the navigation enabled by World Lines and the advanced functionality brought by the scope-set modifier nodes, but also wanted to be sure that their simulations and use cases could be adapted to be used alongside these. Given that it is the algorithms of this paper that make this possible by allowing the time- and multiverse-dependent nodes to be interchangeable and modular, we consider it a proof for the relevance of our system.

12 Performance

In the following subsections, we try to quantify the overhead of our approach as well as the memory it saves.

12.1 System Overhead

There are two sources of overhead within the presented system - the capabilities and the jobs calculation. As Table 2 shows, the capabilities update is the more expensive of the two. However, capabilities change rarely and an interactive response is not necessary for a streamlined user experience. Given that the data-flow of the swept-away vehicles simulation is very complex (55 nodes, 10 tracks, 600 timesteps), the measured time needed is a very positive result.

The jobs calculation is performed with almost every action the user makes. It is thus crucial that its cost is insignificant compared to the time necessary to execute a job. The timings in Table 2 confirm that this is the case. The measurements shown there are performed when the user moves the cursor ahead one frame, with the previous frame already computed. After a cold start of the system, the jobs calculation can take significantly longer (up to 3ms in case of the Moving Vehicles case). This is because every node needs jobs to be assigned to it, and in case of a simulation node, the number of jobs can be quite large. However, even in this case, the jobs calculation is a minimal overhead, comparing favorably to the execution time. With more precise simulations using a better resolution and more time, the system overhead becomes even less significant.

12.2 Memory Savings

One of the biggest advantages of the system is that additional computations can be avoided thanks to our hierarchical approach. As we are not aware of another system with similar capabilities, we compare it to a naive system computing all the data for each requested frame.

TABLE 2
Performance timings for the two case studies

Case Study	Update Capabilities	Update Jobs	Execute Simulation	Execute Remainder
Vortex Analysis	8.5 ms	0.13 ms	N/A	3.97 s
Moving Vehicles	84 ms	0.3 ms	10 s	2.7 ms

TABLE 3
Memory usage for the vortex analysis for a single frame

Data Type	Input Data	λ_2	Free Surface	Path Lines	Isosurface	Total
Size (MB)	76	2.5	2.8	0.02	1.3	82.62

Table 3 lists the memory size of the individual data objects for a single frame. Assuming the three tracks from the vortex analysis case, the naive system would require three times the total amount, or $247.86MB$. Thanks to the hierarchy, only the isosurface has to be recomputed for the different tracks. Therefore, with the presented approach, only $85.22MB$ are used, which is a reduction to one third of the memory used by the naive system.

In case of the Movement of Swept-Away Vehicles case study, the memory savings are marginal as the memory consumption is dominated by the simulation data. This data differs from track to track, and only small amounts of memory can be saved using our approach. This example highlights that both system and memory overhead are mainly dependent on the use case and the user interaction. Our system is designed to improve performance wherever it is possible.

13 Conclusion

The data-flow pipeline is known as the most generic architecture for modular visualization environments. Indeed, many modern systems are built upon the data-flow approach [9], [12], [13]. Even though all of these systems have some capabilities of working with time-dependent data, none provide a generic formalism for the automatic management of time-varying data. A strategy to handle alternative scenarios as induced by multiple simulation runs is also largely missing. As a result, either programmers or users have to make sure that nodes can access data from all time steps required to compute the requested results. In this paper, we have shown how to extend the data-flow model with algorithms which enable elaborate data processing across many time steps and alternative outcomes. By combining the new data-flow model with the interactive exploration power of World Lines, the complexity of temporal and multiversal processing is hidden from the user. Intuitive navigation concepts let users easily choose what they want to see, and internally, the work load is distributed to the nodes automatically.

All of the discussed features have been implemented in Visdom [23]. The framework supports simulation nodes, different kinds of temporal integration nodes such as pathlines or streak surfaces and many forms of data aggregation. Extending Visdom with new time-dependent nodes requires little effort from the programmer, because the management code resides in the core of the system. With the presented concepts, World Lines can now be used to explore parameter spaces of any node, not only those of simulation nodes. The ability to cache and visualize computed results for different parameters in form of interactive tracks and frames enables a flexible comparative analysis of parameter spaces.

Even though the current implementation is single-threaded, the presented concepts do not prohibit parallelization. To the contrary, the explicit computation of jobs and their dependencies allows for parallelization based on jobs and not just on nodes. This can have a huge impact on computation time as nodes can already start computing results when their input nodes are still running. In a decision-support system, this gives the user the possibility to analyze simulation data while the simulation itself is still running and producing results.

Another limitation of our current implementation is a lack of a caching strategy. Even though we are able to control the memory usage by swapping out data to the hard-drive, we do not swap data based on frequency of use, nor delete data that could be easily recomputed. Introducing a cache manager would be simple, as the current design supports data eviction, and regenerates data automatically when needed. As future work, we therefore plan on studying various caching and eviction strategies.

The data-flow is a splendid example of modular architecture. By simply adding and connecting nodes, the user can integrate diverse functionalities into the desired processing scheme. In our opinion, the greatest contribution of our work is that it allows complex components such as simulations, integrations or interpolations to be introduced in the same way as the simplest filter node. It was not impossible to add these before, but the amount of work necessary to do so has been lowered significantly. Easier experimentation, in conjunction with the presentation abilities, can contribute to the interesting work currently happening in the area of ensemble simulations and uncertainty visualization. There exist many different application areas where these techniques could be used - traffic simulations, computer fluid dynamics, and climate research being just some examples. We hope that our approach will help the spread and reuse of advanced visualization techniques.

Acknowledgments

The project SemSeg acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET-Open grant number 226042. This work was supported by Swiss National Science Foundation (SNF) under grant 200021_127022 (SPHVis++) and in part by grants from the Austrian Science Fund (FWF):P 22542-N23 (Semantic Steering) and the Vienna Science and Technology Fund (WWTF):ICT12-009 (Scenario Pool). The authors thank M. Eduard Gröller, Günter Blöschl, Jürgen Komma, GeoConsult, and the

Hochwasserschutzzentrale der Stadtentwässerungsbetriebe Köln, AöR for their input.

REFERENCES

[1] Scientific Computing and Imaging Institute (SCI), "SCIRun: A scientific computing problem solving environment," http://www.scirun.org (last visited on March, 11th 2011).

[2] C. Upson, T. Faulhaber, Jr., D. Kamins, D. H. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam, "The application visualization system: A computational environment for scientific visualization," *IEEE Computer Graphics and Applications*, vol. 9, pp. 30–42, July 1989.

[3] "Video world lines," http://visdom.at/media/slides/world_lines_final.mp4 (last visited on September, 15th 2011).

[4] J. Waser, R. Fuchs, H. Ribičić, B. Schindler, G. Blöschl, and M. E. Gröller, "World Lines," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1458–1467, 2010.

[5] J. Waser, H. Ribičić, R. Fuchs, C. Hirsch, B. Schindler, G. Blöschl, and M. E. Gröller, "Nodes on Ropes: A Comprehensive Data and Control Flow for Steering Ensemble Simulations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, pp. 1872–1881, 2011.

[6] Advanced Visual Systems Inc., "AVS - Advanced Visual System," http://www.avs.com/ (last visited on March, 21st 2011).

[7] J. Walton, "NAG's IRIS Explorer," http://www.nag.co.uk/Welcome_IEC.asp (last visited on March, 21st 2011), NAG - Numerical Algorithms Group, Tech. Rep. IETR/12 (NP3654), 2004. [Online]. Available: http://www.nag.co.uk/Welcome_IEC.asp

[8] IBM, "Open visualization data explorer," http://www.research.ibm.com/dx/ (last visited on May, 3rd 2011).

[9] VTK, "Kitware: The visualization toolkit and paraview," http://www.kitware.com.

[10] R. B. Haber and D. A. McNabb, "Visualization idioms: A conceptual model for scientific visualization systems," *IEEE Visualization in Scientific Computing*, pp. 74–93, 1990.

[11] Visage Imaging, Inc., "Amira - a platform for 3d visualization," http://www.amira.com/ (last visited on September, 9th 2011).

[12] Visualization Sciences Group (VSG), "Avizo - 3d analysis software for scientific and industrial data," http://www.vsg3d.com/avizo/overview (last visited on September, 9th 2011).

[13] Department of Energy (DOE) Advanced Simulation and Computing Initiative (ASCI), "VisIt - An interactive parallel visualization tool for viewing and animating visualizations from scientifc data," https://wci.llnl.gov/codes/visit/home.html (last visited on February, 9th 2011).

[14] Kitware, "Paraview - an open source, multi-platform data analysis and visualization application." http://www.paraview.org/ (last visited on March, 17th 2011).

[15] H. Childs, E. Brugger, K. Bonnell, J. Meredith, M. Miller, B. Whitlock, and N. Max, "A contract based system for large data visualization," *Visualization Conference, IEEE*, p. 25, 2005.

[16] J. Biddiscombe, B. Geveci, K. Martin, K. Moreland, and D. Thompson, "Corrections to 'time dependent processing in a parallel pipeline architecture'," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 1, p. 241, 2008.

[17] Y. Koike, A. Sugiura, and Y. Koseki, "Timeslider: an interface to specify time point," in *Proceedings of the 10th annual ACM symposium on user interface software and technology (UIST)*, 1997, pp. 43–44.

[18] SimVis Gmbh, "SimVis - A novel visualization solution for simulation results," http://www.simvis.at (last visited on May, 18th 2011).

[19] J. Meyer-Spradow, T. Ropinski, J. Mensmann, and K. H. Hinrichs, "Voreen: A rapid-prototyping environment for ray-casting-based volume visualizations," *IEEE Computer Graphics and Applications*, vol. 29, no. 6, pp. 6–13, 2009.

[20] J. Waser, R. Fuchs, H. Ribičić, and G. Blöschl, "Visuelle Aktionsplanung im Hochwassermanagement mit Visdom," in *Forum für Hydrologie und Wasserbewirtschaftung/FgHW*, vol. 30, no. 11, 2011, pp. 280–286.

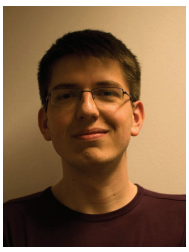
[21] J. Jeong and F. Hussain, "On the identification of a vortex," *Journal of Fluid Mechanics*, vol. 285, pp. 69–84, 1995.

[22] C. T. Silva, J. Freire, and S. P. Callahan, "Provenance for visualizations: Reproducibility and beyond," *Computing in Science and Engineering*, vol. 9, no. 5, pp. 82–89, 2007.

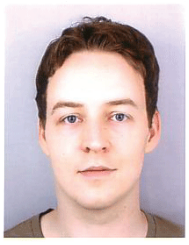
[23] "Visdom integrated visualization framework," http://www.visdom.at (last visited on September, 9th 2011).

Benjamin Schindler is a graduate student at the ETH Zürich, Switzerland. He received his masters in Computer Science at the ETH in 2009. His research focuses on unsteady flow topology and visualization of SPH data.

Jürgen Waser graduated in 2008 from the General Physics Institute at Vienna University of Technology, Austria. In 2011, he received his Ph.D. degree in computer science from the Vienna University of Technology. He is core developer and co-founder of the Visdom framework. His research fields of interest include simulation-steering and web-based visualization systems.



Hrvoje Ribičić is a Ph.D student at the VRVis research center for virtual reality and visualization. He has graduated from the Faculty of Electrical Engineering and Computing at the University of Zagreb in 2010. His current research topics involve the visualization and analysis of interactively steerable simulation scenarios.



Raphael Fuchs graduated in computer science (2006) at Philipps-University Marburg (Germany) and received his Ph.D. degree in computer science (2008) from the Vienna University of Technology (Austria). He is now with the Chair of Computational Science (CSElab) at the ETH Zurich (Switzerland). His fields of interest include scientific visualization and applications of machine learning to visualization.

Ronald Peikert received his diploma in mathematics in 1979 and his Ph.D. in mathematics in 1985, both from ETH Zurich. He is currently a titular professor in the computer science department of ETH Zurich. His research interests include flow visualization, feature extraction techniques, and industrial applications of visualization. He is a member of the IEEE Computer Society.