VisComplete: Automating Suggestions for Visualization Pipelines

David Koop, Carlos E. Scheidegger, Steven P. Callahan, Juliana Freire, and Cláudio T. Silva

Abstract—Building visualization and analysis pipelines is a large hurdle in the adoption of visualization and workflow systems by domain scientists. In this paper, we propose techniques to help users construct pipelines by consensus—automatically suggesting completions based on a database of previously created pipelines. In particular, we compute correspondences between existing pipeline subgraphs from the database, and use these to predict sets of likely pipeline additions to a given partial pipeline. By presenting these predictions in a carefully designed interface, users can create visualizations and other data products more efficiently because they can augment their normal work patterns with the suggested completions. We present an implementation of our technique in a publicly-available, open-source scientific workflow system and demonstrate efficiency gains in real-world situations.

Index Terms—Scientific Workflows, Scientific Visualization, Auto Completion

1 Introduction

Data exploration through visualization is an effective means to understand and obtain insights from large collections of data. Not surprisingly, visualization has grown into a mature area with an established research agenda [27], and a number of software systems have been developed that support the creation of complex visualizations [18, 6, 24, 36, 19, 30, 13, 37]. However, a wider adoption of visualization systems has been greatly hampered due to the fact that these systems are notoriously hard to use, in particular, for users who are not visualization experts.

Even for systems that have sophisticated visual programming interfaces, such as DX, AVS and SCIRun, the path from the raw data to insightful visualizations is laborious and error-prone. Visual programming interfaces expose computational components as *modules* and allow the creation of complex visualization pipelines which combine these modules in a dataflow, where *connections* between modules express the flow of data through the pipeline. They have been shown to be useful for comparative visualization and efficient exploration of parameter spaces [2]. Through the use of a simple programming model (i.e., dataflows) and by providing built-in constraint checking mechanisms (e.g., that disallow a connection between incompatible module ports), they ease the creation of pipelines. Notwithstanding, without detailed knowledge of the underlying computational components, it is difficult to understand what series of modules and connections ought to be added to obtain a desired result. In essence, there is no "roadmap"; systems provide very little feedback to help the user figure out which modules can or should be added to the pipeline. A novice user (i.e., an experienced programmer that is unfamiliar with the modules and the dataflow of the system), or even an advanced user performing a new task, often resorts to manually searching for existing pipelines to use as examples. These examples are then adapted and iteratively refined until a solution is found. Unfortunately, this manual, time-consuming process is the current standard for creating visualizations rather than the exception.

Recent work has shown that provenance information (the metadata required for reproducibility) can be used to simplify the process of pipeline creation by allowing pipelines to be refined and queried by example [31]. For example, a pipeline refinement can act as an analogy template for creating new visualizations. This is a powerful tool and can be helpful in situations when the user knows in advance what they want the end result to be. However, during pipeline creation, it is not always the case that the user has an analogy template readily available for the visualization that is desired. In these cases, the user is relegated to manually searching for examples.

In this paper, we present VisComplete, a system that aids users in the process of creating visualizations by using a database of previously created visualization pipelines. The system learns common paths used in existing pipelines and predicts a set of likely module sequences that can be presented to the user as suggestions during the design process. The quality and nature of the suggestions depend on the data they are derived from. Whereas in a single-user environment, suggestions are derived based on pipelines created by a specific user, in a multi-user environment, the "wisdom of the crowds" can be leveraged to derive a richer set of suggestions that includes examples the user is not familiar with. User collaboration and social data reuse has proven to be a powerful mechanism in various domains, such as recommendation systems in commercial settings (e.g., Amazon, e-Bay, Netflix), knowledge sharing on open Web sites (e.g., Wikipedia), image labeling for computer vision (e.g., ESPGame) and visualization creation (e.g., ManyEyes). The underlying theme shared by these systems is that they use information provided by many users to solve problems that would be difficult otherwise. We apply a similar concept to pipeline creation: pipelines created by many users enable the creation of visualizations *by consensus*. For the user, VisComplete acts as an auto-complete mechanism for pipelines, suggesting modules and connections in a manner similar to a Web browser suggesting URLs. The completions are presented graphically in a way that allows the user to easily explore and accept suggestions or disregard them and continue working as they were. Figure 1 shows an example of VisComplete incorporated into a visual programming interface and Figure 2 shows some example completions for a single module.

Contributions and Outline. We propose a recommendation system that leverages information in a collection of pipelines to provide advice to users of visualization systems and aid them in the construction of pipelines. By modeling pipelines as graphs, we develop an algorithm for predicting likely completions that searches for common subgraphs in the collection. We also present an interface that displays the recommended completions in an intuitive way. Our preliminary experiments show that VisComplete has the potential to reduce the effort and time required to construct visualizations. We found that the suggestions derived by VisComplete could have reduced the number of operations performed by users to construct pipelines by an average of over 50%. Note that although in this paper we focus on the use of VisComplete for visualization pipelines, the techniques we present can be applied to general workflows.

The rest of this paper is organized as follows. In Section 2 we discuss related work. In Section 3 we present the underlying formalism for generating pipeline suggestions, and in Section 4 we describe a

- *David Koop and Juliana Freire are with the School of Computing at the University of Utah. email: {dakoop, juliana}@cs.utah.edu.*
- *Carlos E. Scheidegger, Steven P. Callahan, and Cláudio T. Silva are with the Scientific Computing and Imaging (SCI) Institute at the University of Utah. email: {cscheid, stevec, csilva}@sci.utah.edu.*

Manuscript received 31 March 2008; accepted 1 August 2008; posted online 19 October 2008; mailed on 13 October 2008.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

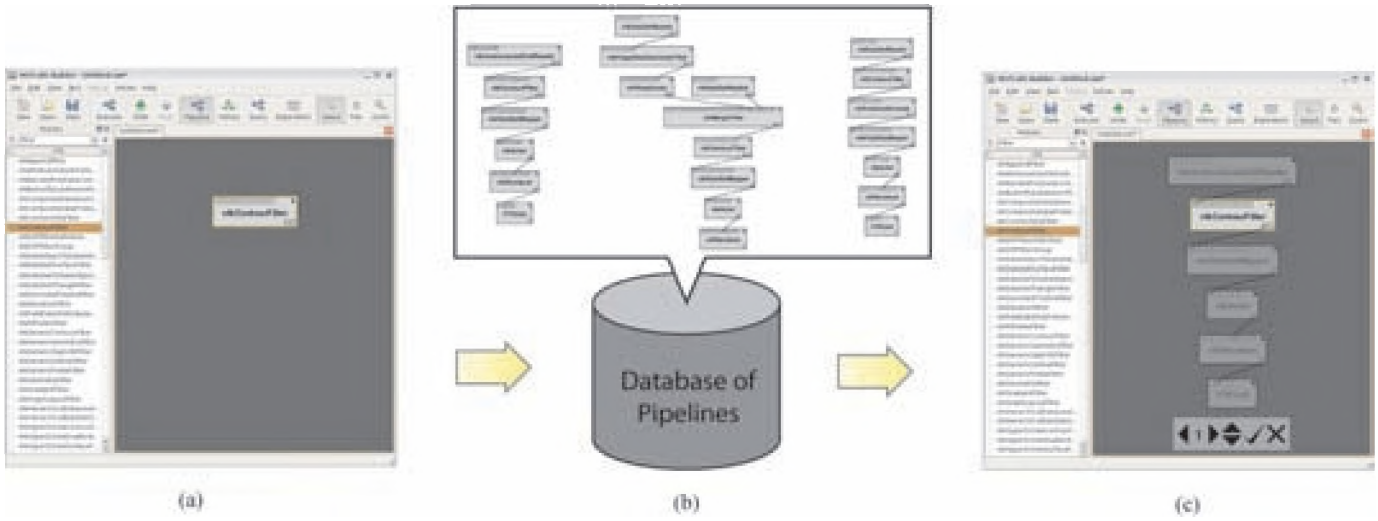


Fig. 1: The VisComplete suggestion system and interface. (a) A user starts by adding a module to the pipeline. (b) The most likely completions are generated using indexed paths computed from a database of pipelines. (c) A suggested completion is presented to the user as semi-transparent modules and connections. The user can browse through suggestions using the interface and choose to accept or reject the completion.

practical implementation that has been integrated into the VisTrails system [36]. We then detail the use cases we envision in Section 5, report our experiments and results in Section 6, and provide a discussion of our algorithm in Section 7. We conclude in Section 8, where we outline directions for future work.

2 Related Work

Visualization systems have been successfully used to bring powerful visualization techniques to a wide audience. Seminal workflow-based visualization systems such as AVS Explorer [37], Iris Explorer [28], and Visualization Data Explorer [13] have paved the way for more recent systems designed using an object-oriented approach such as SciRun [30] for computational steering and the Visualization Toolkit (VTK) [19] for visualization. Systems that incorporate standard point-and-click interfaces and operate on data at a larger scale, such as VisIt [6] and ParaView [18], still use workflows as their underlying execution engine. Development in workflow systems for visualization is ongoing, as seen in projects such as MeVisLab [25] for medical visualization and VisTrails [36] for incorporating existing visualization libraries with other tools in a provenance capturing framework. Our completion strategy can be combined with and enhance workflow and workflow-based visualization systems.

Recommendation systems have been used in different settings. Like VisComplete, these are based on methods that predict users' actions based solely on the history of their previous interactions [12]. Examples include Unix command-line prediction [20], prediction of Web requests [8, 29], and autocompletion systems such as IntelliSense [26]. Senay and Ignatius have proposed incorporating expert knowledge into a set of rules that allow automated suggestions for visualization construction [32], while Gilson et al. incorporate RDF-based ontologies into an information visualization tool [9]. However, these approaches necessarily require an expert that can encode the necessary knowledge into a rule set or an ontology.

Fu et al. [8] applied association rule mining [1] to analyze Web navigation logs and discover pages that co-occur with high frequency in navigation paths followed by different users. This information is then used to suggest potentially interesting pages to users. VisComplete also derives predictions based on user-derived data and does so in an automated fashion, without the need for explicit user feedback. However, the data it considers is fundamentally different from Web logs: VisComplete bases its predictions on a collection of graphs and it leverages the graph structure to make these predictions. Because association rule mining computes rules over *sets* of elements, it does not capture relationships (other than co-occurrence) amongst these elements.

In graphics and visualization, recommendation systems have been proposed to simplify the creation of images and visualizations. Design Galleries [23] were introduced to allow users to explore the space of rendering parameters by suggesting a set of automatically generated thumbnails. Igarashi and Hughes [14] proposed a system for creating 3D line drawings that uses rules to suggest possible completions of 3D objects. Suggestions have also been used for view point selection in volume rendering. Bordoloi and Shen [39] and Takahashi et al. [34] present methods that analyze the volume from various view points to suggest the view that best shows the features within the volume. Like these systems, we provide the user with prioritized suggestions that the user may choose to utilize. However, our suggestions are data-driven and based on examples of previous interactions.

An emerging trend in image processing is to enhance images based on a database of existing images. Hays and Efros [10] recently presented a system for filling in missing regions of an image by searching a database for similar images. Along similar lines Lalonde et al. [21] recently introduced Photo Clip Art, a method for intelligently inserting clip art objects from a database to an existing image. Properties of the objects are learned from the database so that they may be sized and oriented automatically, depending on where they are inserted into the image. The use of databases for completion has also been used for 3D modeling. Tsang et al. [35] proposed a modeling technique that utilizes previously created geometry stored in a database of shapes to suggest completions of objects. Like these methods, our completions are computed by learning from a database to find similarities. But instead of images, our technique relies on workflow specifications to derive predictions.

Another important trend is that of social visualization. Web-based systems such as VisPortal [3, 15] provide the means for collaborative visualization from disjoint locations. Web sites such as Sens.us [11], Swivel [33], and ManyEyes [38] allow many users to create, share, and discuss visualizations. One key feature of these systems is that they leverage the knowledge of a large group of people to effectively understand disparate data. Similarly, VisComplete uses a collection of pipelines possibly created by many users to derive suggestions.

3 Generating Data-driven Suggestions

VisComplete suggests partial completions (i.e., a set of structural changes) for pipelines as they are being created by a user. These suggestions are derived using structural information obtained from a collection $\mathscr{G}$ of already-completed pipelines.

Pipelines are specified as graphs, where nodes represent modules (or processes) and edges determine how data flows through the modules. More formally, a *pipeline specification* is a directed acyclic graph

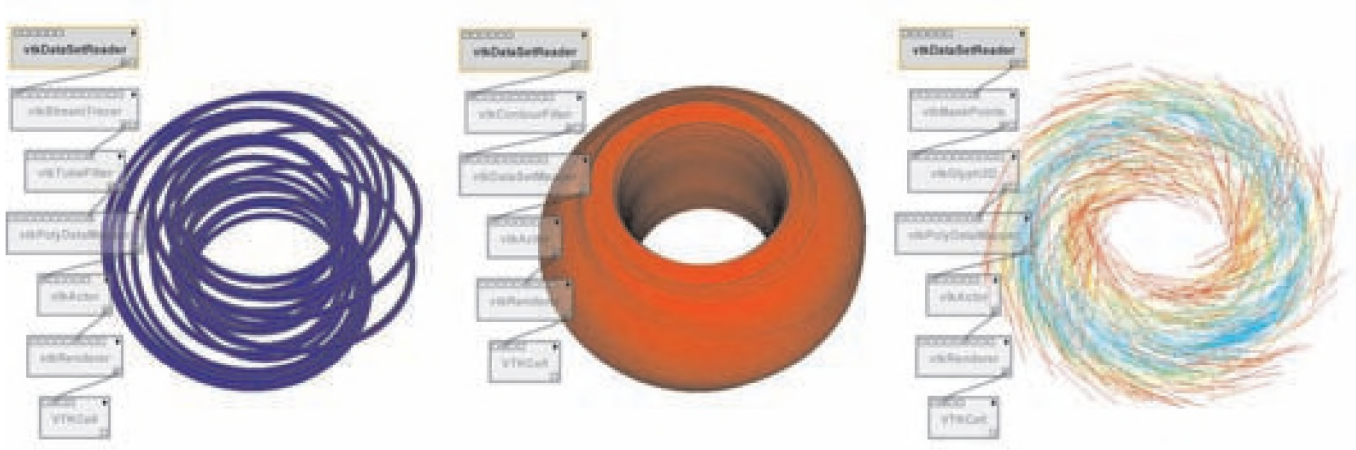


Fig. 2: Three of the first four suggested completions for a "vtkDataSetReader" are shown along with corresponding visualizations. The visualizations were created using these completions for a time step of the Tokamak Reactor dataset that was not used in the training data.

$G(M,C)$, where M consists of a set of modules and C is a set of connections between modules in M. A *module* is a complex object which contains a set of input and output ports through which data flows in and out of the module. A *connection* between two modules m_a and m_b connects an output port of m_a to an input port of m_b.

Problem Definition. The problem of deriving pipeline completions can be defined as follows. Given a partial graph G, we wish to find a set of completions $C(G)$ that reflect the structures that exist in a collection of completed graphs. A *completion* of G, G^c, is a supergraph of G.

Our solution to this problem consists of two main steps. First, we pre-process the collection of pipelines $\mathcal{G}$ and create $\mathcal{G}_{path}$, a compact representation of $\mathcal{G}$ that summarizes relationships between common structures (i.e., sequences of modules) in the collection (Section 3.1). Given a partial pipeline p, completions are generated by querying $\mathcal{G}_{path}$ to identify modules and connections that have been used in conjunction with p in the collection (Section 3.2).

3.1 Mining Pipelines

To derive completions, we need to identify graph fragments that co-occur in the collection of pipelines $\mathcal{G}$. Intuitively, if a certain fragment always appears connected to a second fragment in our collection, we ought to predict one of those fragments when we see the other.

Because we are dealing with directed acyclic graphs, we can identify potential completions for a vertex v in a pipeline by associating subgraphs downstream from v with those that are upstream. A subgraph S is *downstream* (*upstream*) of a vertex v if for every $v' \in S$, there exists a path from v to v' (v' to v). In many cases where we wish to complete a graph, we will know either the downstream or upstream structure and wish to complete the opposite direction. Note that this problem is symmetric: we can change one problem to the other by simply reversing the direction of the edges.

However, due to the (very) large number of possible subgraphs in $\mathcal{G}$, generating predictions based on subgraphs can be prohibitively expensive. Thus, instead of subgraphs we use paths, i.e., a linear sequence of connected modules. Specifically, we compute the frequencies for each path in $\mathcal{G}$. Completions are then determined by finding which path extensions are likely given the existing paths.

Generating the Path Summary. To efficiently derive completions from a collection of pipelines $\mathcal{G}$, we begin by generating a summary of all paths contained in the pipelines. Because completions are derived for a specific vertex v in a partial pipeline (we call this vertex the *completion anchor*), we extract all possible paths that end or begin with v and associate them with the vertices that are directly connected downstream or upstream of v. Note that this leads to many fewer entries than the alternative of extracting all possible subgraph pairs. And as we discuss in Section 6, paths are effective and lead to good predictions.

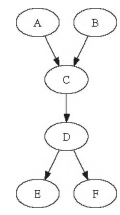
More concretely, we extract all possible paths of length N, and split them into a path of length $N-1$ and a single vertex. Note that we do this in *both* forward and reverse directions with respect to the directed edges. This allows us to offer completions for pipeline pieces when they are built top-down and bottom-up. The path summary $\mathcal{G}_{path}$ is stored as a set of (path, vertex) pairs sorted by the number of occurrences in the database and indexed by the last vertex of the path (the anchor). Since predictions begin at the anchor vertex, indexing the path summary by this vertex leads to faster access to the predictions.

As an example of the path summary generation, consider the graph shown in Figure 3. We have the following upstream paths ending with D: $A \rightarrow C \rightarrow D$, $B \rightarrow C \rightarrow D$, $C \rightarrow D$, and D. In addition, we also have the following downstream vertices: E and F. The set of correlations between the upstream paths and downstream vertices is shown in Figure 3. As we compute these correlations for all starting vertices over all graphs, some paths will have higher frequencies than others. The frequency (or support) for the paths is used for ranking purposes: predictions derived from paths with higher frequency are ranked higher.

Besides paths, we also extract additional information that aid in the construction of completions. Because we wish to predict full pipeline structures, not just paths, we compute statistics for the in- and out-degrees of each vertex type. This information is important in determining where to extend a completion at each iteration (see Figure 4). We also extract the frequency of connection types for each pair of modules. Since two modules can be connected through different pairs of ports, this information allows us to predict the most frequent connection type.

3.2 Generating Predictions

Predicting a completion given the path summary and an anchor module v is simple: given the set of paths associated with v, we identify the vertices that are most likely to follow these paths. As shown in the following algorithm, we iteratively develop our list of predictions by adding new vertices using this criteria.



path	vertex
$A \rightarrow C \rightarrow D$	E
$A \rightarrow C \rightarrow D$	F
$B \rightarrow C \rightarrow D$	E
$B \rightarrow C \rightarrow D$	F
$C \rightarrow D$	E
$C \rightarrow D$	F
D	E
D	F

Fig. 3: Deriving a path summary for the vertex D.

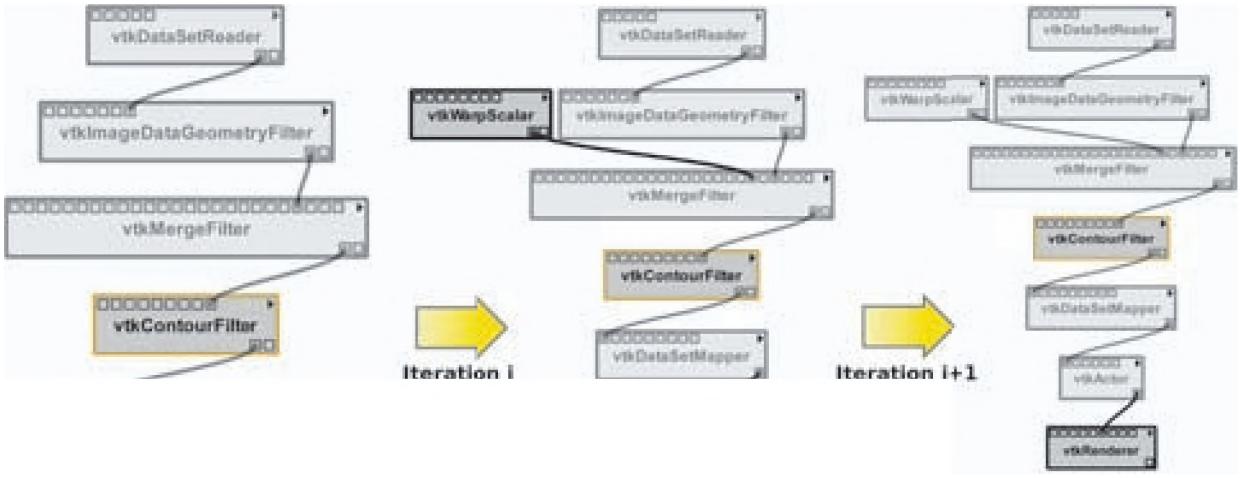


Fig. 4: Predictions are iteratively refined. At each step, a prediction can be extended upstream and downstream; in the second step, the algorithm only suggests a downstream addition. Also, predictions in either direction may include branches in the pipeline, as shown in the center.

GENERATE-PREDICTIONS(P)
  predictions ← FIRST-PREDICTION(P)
  result ← [ ]
  while |*predictions*| > 0
    do *prediction* ← REMOVE-FIRST(*predictions*)
      new-predictions ← REFINE(*prediction*)
      if |*new-predictions*| = 0
        then *result* ← *result* + *prediction*
        else *predictions* ← *predictions* + *new-predictions*

At each step, we refine existing predictions by generating new predictions that add a new vertex based on the path summary information. Note that because there can be more than one possible new vertex, we may add more than one new prediction for each existing prediction. Figure 4 illustrates two steps in the prediction process.

To initialize the list of predictions, we use the specified anchor modules (provided as input). At this point, each prediction is simply a base prediction that describes the anchor modules and possibly how they connect to the pipeline. After initialization, we iteratively refine the list of predictions by adding to each suggestion. Because there are a large number of predictions, we need some criteria to order them so that users can easily locate useful results. We introduce *confidence* to measure the goodness of the predictions.

Given the set of upstream (or downstream depending on which direction we are currently predicting) paths, the confidence of a single vertex $c(v)$ is the measure of how likely that vertex is, given the upstream paths. To compute the confidence of a single vertex, we need to take into account the information given by all upstream paths. For this reason, the values in $\mathcal{G}_{path}$ are not normalized; we use the exact counts. Then, as illustrated by Figure 5, we combine the counts from each path. This means we do not need any weighting based on the frequency of paths; the formula takes this into account automatically. Specifically,

$$c(v) = \frac{\sum_{P \in upstream(v)} count(v \mid P)}{\sum_{P \in upstream(v)} count(P)}$$

Then, the confidence of a graph G is the product of the confidences of each of its vertices:

$$c(G) = \prod_{v \in G} c(v)$$

While each vertex confidence is not entirely independent, this measure gives a reasonable approximation for the total confidence of the graph. Because we perform our predictions iteratively, we calculate the confidence of the new prediction p_{i+1} as the product of the confidence of the old prediction p_i and the confidence of the new vertex v:

$$c(p_{i+1}) = c(p_i) \cdot c(v)$$

For computational stability, our implementation uses log-confidences so the products are actually sums.

Because we wish to derive predictions that are not just paths, our refinement step begins by identifying the vertex in the current prediction that we wish to extend our prediction from. Recall that we computed the average in- and out-degree for each vertex type in the mining step. Then, for each vertex, we can compute the difference between the average degree for its type and its current degree for the current prediction direction. We choose to extend completions at vertices where the current degree is much smaller than the average degree. We also incorporate this measure into our vertex confidence so that predictions that contain vertices with too many edges are ranked lower:

$$c_d(v) = c(v) + degree\text{-}difference(v)$$

We stop iteratively refining our predictions after a given number of steps or when no new predictions are generated. At this point, we sort all of the suggestions by confidence and return them. If we have too many suggestions, we can choose to prune our set of predictions at each step by eliminating those which fall below a certain threshold.

3.3 Biasing the Predictions

The prediction mechanism described above relies primarily on the frequency of paths to rank the predictions. There are, however, other factors that can be used to influence the ranking. For example, if a user has been working on volume rendering pipelines, completions that emphasize modules related to that technique could be ranked higher than those dealing with other techniques. In addition, some users will prefer certain completions over others because they more closely mirror their own work or their own pipeline structures. Again, it makes sense to bias completions toward user preferences. We can adapt our algorithm to include such bias by incorporating a weighting factor in the confidence computation. Specifically, we adjust our counts by weighting the contribution of each path according to a pipeline importance factor determined by a user's preferences.

4 IMPLEMENTATION

Our implementation is split into three specific steps: determining when completion should be invoked, computing the set of possible completions, and presenting these suggestions to the user. Computing the possible completions requires the machinery developed in the previous section. The other steps are essential to make the approach usable. The interface, in particular, plays a significant role in allowing users to make use of suggestions while also being able to quickly dismiss them when they are not desired.

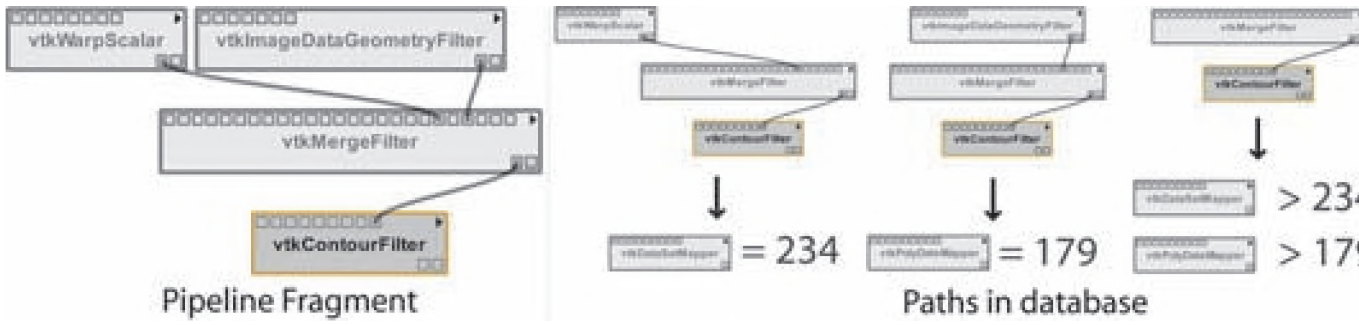


Fig. 5: At each iteration, we examine all upstream paths to suggest a new downstream vertex. We select the vertex that has the largest frequency given all upstream paths. In this example, "vtkDataSetMapper" would be the selected addition.

4.1 Triggering a Completion

We want to provide an environment where suggestions are offered automatically but do not interfere with a user's normal work patterns. There are two circumstances in pipeline creation where it makes sense to automatically trigger a completion: when a user adds a new module and when a user adds a new connection. In each of these cases, we are given new information about the pipeline structure that can be used to narrow down possible completions. Because users may also wish to invoke completion without modifying the pipeline, we also provide an explicit command to start the completion process.

In each of the triggering situations, we begin the suggestion process by identifying the modules that serve as anchors for the completions. For new connections, we use both of the newly connected modules, and for a user-requested completion, we use the selected module(s). However, when a user adds a new module, it is not connected to the rest of the existing pipeline. Thus, it can be difficult to offer meaningful suggestions since we have no surrounding structure to leverage. We address this issue by first finding the most probable connection to the existing pipeline, and then continue with the completion process.

Finding the initial connection for an added module may be difficult when there are multiple modules in the existing pipeline than can be connected to the new module. However, because visual programming interfaces allow users to drag and place new modules in the pipeline, we can use the initial position of the module to help infer a likely connection. To accomplish this, we compute the user's layout direction based on the existing pipeline, and locate the module that is nearest to the new module and can be connected to it.

4.2 Computing the Suggestions

As outlined in the previous section, we compute possible completions that emanate from a set of anchor modules in the existing pipeline using path summaries derived from a database of pipelines, and rank them by their confidence values. Depending on the anchor modules, a very large set of completions can be derived and a user is unlikely to examine a long list of suggestions. Therefore, we prune our predictions to avoid rare cases. This both speeds up computation and reduces the likelihood that we provide meaningless suggestions to the user. Specifically, because our predictions are refined iteratively, we prune a prediction if its confidence is significantly lower than its parent's confidence. Currently, this is implemented as a constant threshold, but we can use knowledge of the current distribution or iteration to improve our pruning.

VisComplete provides the user with suggestions that assist in the creation of the pipeline *structure*. Parameters are also essential components in visualizations, but because the choice of parameters is frequently data-dependent, we do not integrate parameter selection with our technique. Instead, we focus on helping users complete pipelines, and direct them to existing techniques [17, 16, 22, 2] to explore the parameter space. Note that it might be beneficial to extend VisComplete to identify commonly used parameters that a user might consider exploring, but we leave this for future work.

4.3 The Suggestion Interface

In concert with our goal of unobtrusiveness, we provide an intuitive and efficient interface that enables users to explore the space of possible completions. Auto-complete interfaces for text generally show a set of possible completions in a one-dimensional list that is refined as the user types. For pipelines this task is more difficult because it is not feasible to show multiple completions at once, as this would result in visual clutter. The complexity of deriving the completion is also greater. For this reason, our interface is two-dimensional: users can select from a list of full completions and then increase or decrease the extent of the completion.

Current text completion interfaces defer to the user by showing completions but allowing the user to continue to type if he does not wish to use the completions. We strive for similar behavior by automatically showing a completion along with a simple navigation panel when a completion is triggered. The user can choose to interact with the completion interface or disregard it completely by continuing to work, which will cause the completion interface to automatically disappear. The navigation interface contains a set of arrows for selecting different completions (left and right) and depths of the current completion (up and down). In addition, the rank of the current completion is displayed to assist in the navigation and accept and cancel buttons are provided (see Figure 1(c)). All of these completion actions, along with the ability to start a new completion with a selected module, are also available in a menu and as shortcut keys.

The suggested completions appear in the interface as semi-transparent modules and connections, so that they are easy to distinguish from the existing pipeline components. The suggested modules are also arranged in an intuitive way using a set of simple heuristics that respect the layout of the current pipeline. The first new suggested module is always placed near the anchor module. The offset of the new module from the anchor module is determined by averaging the direction and distance of each module in the existing pipeline. The offset for each additional suggested module is calculated by applying this same rule to the module it is appended to. Branches in the suggested completion are simply offset by a constant factor. These heuristics keep the spacing uniform and can handle upstream or downstream completions whether pipelines are built top-down or left-right.

5 Use Cases

We envision VisComplete being used in different ways to simplify the task of pipeline construction. In what follows, we discuss use cases which consider different types of tasks and different user experience levels. The types of tasks performed by a user can range from the very repetitive to the unique. Obviously, if the user performs tasks that are very similar to those in the database of pipelines, the completions that are suggested are very full—almost the entire pipeline can be created using one or two modules (see Figure 2 for examples). On the other hand, if the task that is being performed is not often repeated and nothing similar in the database can be found, VisComplete will only be able to assist with smaller portions of the pipeline at a time.

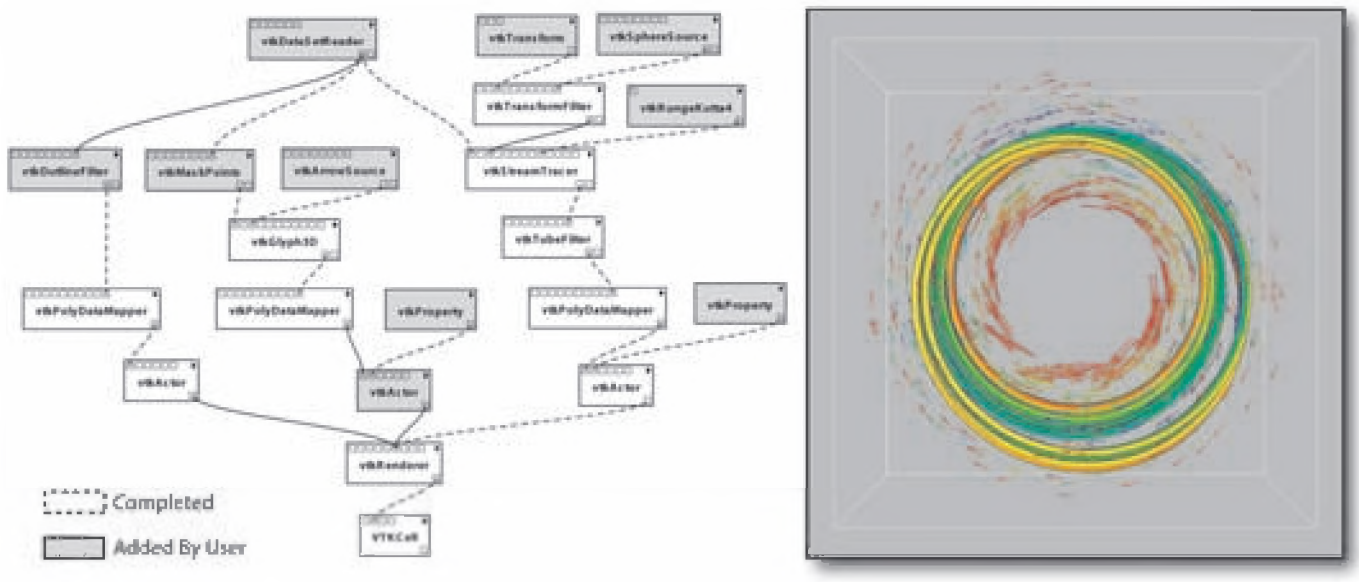


Fig. 6: One of the test visualization pipelines applied to a time step of the Tokamak Reactor dataset. VisComplete could have made many completions that would have reduced the amount of time creating the pipeline. In this case about half of the modules and completions could have been completed automatically.

This can still aid the user by showing the possible directions to proceed with pipeline construction, albeit at a smaller scale.

The experience level of users that could take advantage of VisComplete also varies. For a novice user, VisComplete replaces the process of searching for and tweaking an example that will perform their desired visualization. For example, a user who is new to VTK and desires to compute an isosurface of a volume might consult documentation to determine that a "vtkContourFilter" module is necessary and then search online for an example pipeline using this module. After downloading the example, they may be able to manipulate it to produce the desired visualization. Using VisComplete, this process is simplified—the user needs only to start the pipeline by adding a "vtkContourFilter" module and their pipeline will be constructed for them (see Figure 1). Multiple possible completions can easily be explored and unlike examples downloaded from the Web, VisComplete can customize the suggestions by providing completions that more closely reflect a specific user's previous or more current work.

For experienced users, VisComplete still offers substantial benefits. Because experts may not wish to see full pipelines as completions, the default depth of the completions can be adjusted as a preference so that only minor modifications are suggested at each step. Thus, at the smallest completion scale, a user can leverage just the initial connection completion to automatically connect new modules to their pipeline. The user could also choose to ignore suggested completions as they add modules until the pipeline is specific enough to shrink the number of suggestions. Unlike the novice user who may iterate through many suggestions at each step, the experienced user will likely choose to ignore the suggestions until they provide the desired completion on the first try.

6 Evaluation

6.1 Data and Validation Process

To evaluate the effectiveness of our completion technique, we used a set containing 2875 visualization pipelines along with logs of the actions used to construct each pipeline. These pipelines were constructed by 30 students during a scientific visualization course.[1] Throughout the semester, the students were assigned five different tasks and carried them out using the VisTrails system, which captures detailed provenance of the pipeline design process: the series of the actions a user followed to create and refine a set of related pipelines [7].

[1] http://www.vistrails.org/index.php/SciVisFall2007

The first four tasks were straightforward and required little experimentation, but the final task was open-ended; users were given a dataset without any restrictions on the use of available visualization techniques. As these users learned about various techniques over the semester, their proficiency in the area of visualization presumably progressed from a novice level toward the expert level.

To predict the performance gains VisComplete might attain, we created user models based on the provenance logs captured by VisTrails. User modeling has been used in the HCI community for many years [4, 5], and we employed a low-level model for our evaluation. Specifically, we assumed that at each step of the pipeline construction process, a VisComplete user would either modify the pipeline according to the current action from the log or select a completion that adds a part of the pipeline they would eventually need. We assumed that a user would examine at most ten completions and could select a subgraph of any of these suggestions.

Because VisComplete requires a collection of pipelines to derive suggestions, we divided our dataset into training and test sets. The training sets were used to construct the path summaries while the test sets were used with the user models to measure performance.

We note that this model presumes a user's foreknowledge of the completed pipeline, and this certainly is not always the case. Still, we believe this simple model approximates user behavior well enough to gauge performance. We also assumed a greedy approach in our model; a user would always take the largest completion that matched their final pipeline. Note that this might not always yield the best performance because the quality of the suggestions may improve as the pipeline is further specified.

6.2 Results

Figure 6 shows one of the test pipelines with the components that VisComplete could have completed highlighted along with its resulting visualization. To evaluate the situation where a set of users create pipelines that all tend to follow a similar template, we performed a leave-one-out test for each task in our dataset. Figure 7 shows that our suggestion algorithm could have eliminated over 50%, on average, of the pipeline construction operations for each task. Because Task 1 was more structured than the other tasks, it achieved a higher percentage of reduction. Because Task 4 was more open-ended, although the average percentage is also high, the results show a wider variation (between 30% and 75%). This indicates that the completion interface can be faster and more intuitive than manually choosing a template.

Fig. 7: Box plot of the percentages of operations that could be completed per task (higher is better). The statistics were generated for each user by taking them out of the training data.

Fig. 8: Box plot of the percentages of operations that could be completed given two types of tasks, novice and expert. The statistics were generated by evaluating the novice tasks using the expert tasks as training data (novice) and by evaluating the expert tasks using the novice tasks as training data (expert).

Because it is much more likely that our collection will contain pipelines from a variety of tasks, we also evaluated two cases that examined the type of knowledge captured by the pipelines. Since Task 5 was more open-ended and completed after the other four tasks, we expected that most users would be proficient using the tool and closer to the expert user described in Section 5. We ran the completion results using Tasks 1 through 4 as the training data (2250 pipelines) and Task 5 (625 pipelines) as the test data to represent a case where novice users are helping expert users, but we also ran this test in reverse to determine if pipelines from expert users can aid beginners. Figure 8 shows that both tests achieved similar results; this implies that the variety of pipelines from the four novice tasks balanced the knowledge captured in the expert pipelines.

Our testing assumed that users would examine up to ten full completions before quitting. In reality, it is likely that users would give up even quicker. To evaluate how many predictions a user might need to examine before finding the desired completion, we recorded the index of the chosen completion in our tests. Figure 9 shows that the the chosen completion was almost always among the first four. Note that we excluded completions that only specified the connection between the new module and the existing pipeline because these trivial completions are possible at each prediction index.

Our results show that VisComplete can significantly reduce the number of operations required during pipeline construction. In addition, the completion percentages might be higher if our technique were available to the users because it would likely change user's work patterns. For example, a user might select a completion that contains most of the structure they require plus some extraneous components and then delete or replace the extra pieces. Such a completion would almost certainly save the user time but was not captured with our user model. Finally, the parameters (e.g., pruning threshold, degree weighting) for the completion algorithms were not tuned. We plan to evaluate these settings to possibly improve our results.

Fig. 9: Box plot of the average prediction index that was used for the completions in Figure 7 (lower is better). These statistics provide a measure of how many suggestions the user would have to examine before the correct one was found.

The completion examples shown in the figures of this paper, with the exception of Figure 6, used the entire collection of pipelines to generate predictions. Figure 6 used only the pipelines from Tasks 1–4.

7 Discussion

To our knowledge, VisComplete is the first approach for automatically suggesting pipeline completions using a database of existing pipelines. As large volumes of data continue to be generated and stored and as analyses and visualizations grow in complexity, the creation of new content by consensus and the ability to learn by example are essential to enable a broader use of data analysis and visualization tools.

The major difference between our automatic pipeline completion technique and the related work on creating pipelines by analogy [31] is that instead of using a single, known sequence of pipeline actions, our method uses an entire database of pipelines. Thus, instead of completing a pipeline based on a single example, VisComplete uses *many* examples. A second important difference is that instead of predicting a new set of actions, our method currently predicts new structure regardless of the ordering of the additions. This also means that VisComplete only adds to the structure while analogies will delete from the structure as well. By incorporating more provenance information, as in analogies, VisComplete might be able to leverage more information about the order in which additions to a pipeline are made. This could improve the quality of the suggested completions.

We note that there will be situations where data about the types of completions that should occur are not available. Also, some suggestions might not correspond to the user's desires. If there are no completions, VisComplete will not derive any suggestions. If there are completions that do not help, the user can dismiss them by either continuing their normal work or by explicitly canceling completion. Currently we determine the completions in an offline step (by precomputing the path summary, Section 3). We could update the path summary as new pipelines are added to the repository, incorporating new pipelines as they are created. In addition, we could learn from user feedback by, for example, allowing users to remove suggestions that they do not want to see again. Completions could be further refined by assigning greater weight to those that more closely mirror the current user's actions, even if they are not the most likely in the database.

One important aspect of our technique is that it leverages the visual programming environment available in many visualization systems. In fact, it would be difficult to offer suggestions without a visual environment in which to display the structural changes. In addition, the information for the completions comes from the fact that we have structural pipelines from previous work. Without an interface to construct pipeline structures, it would be more difficult to process the data used to generate completions. However, we should note that turnkey applications that are based on workflow systems, such as ParaView [18], may also be able to take advantage of completions in a more limited

way by providing a more intelligent set of default settings for the user during their explorations.

8 Conclusions and Future Work

We have described VisComplete, a new method for aiding in the design of visualization pipelines that leverages a database of existing pipelines. We have demonstrated that suitable pipeline fragments can be computed from the database and used to complete new pipelines in real-time. Furthermore, we have shown how these completions can be presented to the user in an intuitive way that can potentially reduce the time required to create pipelines. Our results indicate that substantial effort can be saved using this method for both novice and expert users.

There are several areas of future work that we would like to pursue. As described above, we would like to update the database of pipelines incrementally, thus allowing the completions to be refined based on current information and feedback from the user. We plan to refine the quality of the results by formally investigating the confidence measure and its parameters. We would also like to explore suggesting finished pipelines from the database in addition to the constructed completions we currently generate. For finished pipelines, we could display not only the completed pipeline structure but also a thumbnail of the result from an execution of that pipeline. Finally, we plan to conduct a user study to further evaluate our technique.

Acknowledgments

We acknowledge the generous help of many colleagues and collaborators. Specifically, we wish to thank the VisTrails team for their support, the VTK developers for their software, and the students of CS 5630/6630 at the University of Utah for their data. Our research has been funded by the Department of Energy SciDAC (VACET and SDM centers), the National Science Foundation (grants IIS-0746500, CNS-0751152, IIS-0713637, OCE-0424602, IIS-0534628, CNS-0514485, IIS-0513692, CNS-0524096, CCF-0401498, OISE-0405402, CCF-0528201, CNS-0551724), IBM Faculty Awards (2005, 2006, 2007, and 2008), and a Graduate Research Fellowship.

References

[1] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, 1993.

[2] L. Bavoil, S. Callahan, P. Crossno, J. Freire, C. Scheidegger, C. Silva, and H. Vo. Vistrails: Enabling interactive, multiple-view visualizations. In *IEEE Visualization*, pages 135–142, 2005.

[3] W. Bethel, C. Siegerist, J. Shalf, P. Shetty, T. J. Jankun-Kelly, O. Kreylos, and K.-L. Ma. VisPortal: Deploying grid-enabled visualization tools through a web-portal interface. In *Third Annual Workshop on Advanced Collaborative Environments*, 2003.

[4] S. K. Card, T. P. Moran, and A. Newell. The keystroke-level model for user performance time with interactive systems. *Commun. ACM*, 23(7):396–410, 1980.

[5] S. K. Card, A. Newell, and T. P. Moran. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, 1983.

[6] H. Childs, E. S. Brugger, K. S. Bonnell, J. S. Meredith, M. Miller, B. J. Whitlock, and N. Max. A contract-based system for large data visualization. In *IEEE Visualization*, pages 190–198, 2005.

[7] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo. Managing rapidly-evolving scientific workflows. In *International Provenance and Annotation Workshop (IPAW)*, LNCS 4145, pages 10–18, 2006. Invited paper.

[8] X. Fu, J. Budzik, and K. J. Hammond. Mining navigation history for recommendation. In *IUI '00: Proceedings of the 5th international conference on Intelligent user interfaces*, pages 106–112, 2000.

[9] O. Gilson, N. Silva, P. Grant, M. Chen, and J. Rocha. VizThis: Rule-based semantically assisted information visualization. Poster, in Proceedings of SWUI 2006, 2006.

[10] J. Hays and A. A. Efros. Scene completion using millions of photographs. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 26(3), 2007.

[11] J. Heer, F. B. Viégas, and M. Wattenberg. Voyagers and voyeurs: supporting asynchronous collaborative information visualization. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI)*, pages 1029–1038, 2007.

[12] H. Hirsh, C. Basu, and B. D. Davison. Learning to personalize. *Communications of ACM*, 43(8):102–106, 2000.

[13] IBM. OpenDX. http://www.research.ibm.com/dx.

[14] T. Igarashi and J. F. Hughes. A suggestive interface for 3d drawing. In *ACM symposium on User interface software and technology (UIST)*, pages 173–181, 2001.

[15] T. J. Jankun-Kelly, O. Kreylos, K.-L. Ma, B. Hamann, K. I. Joy, J. M. Shalf, and E. W. Bethel. Deploying web-based visual exploration tools on the grid. *IEEE Computer Graphics and Applications*, 23(2):40–50, 2003.

[16] T. J. Jankun-Kelly and K.-L. Ma. Visualization exploration and encapsulation via a spreadsheet-like interface. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):275–287, July/September 2001.

[17] T. J. Jankun-Kelly, K.-L. Ma, and M. Gertz. A model and framework for visualization exploration. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):357–369, March/April 2007.

[18] Kitware. Paraview. http://www.paraview.org.

[19] Kitware. VTK. http://www.vtk.org.

[20] B. Korvemaker and R. Greiner. Predicting unix command lines: Adjusting to user patterns. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 230–235, 2000.

[21] J.-F. Lalonde, D. Hoiem, A. A. Efros, C. Rother, J. Winn, and A. Criminisi. Photo clip art. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 26(3), 2007.

[22] K.-L. Ma. Visualizing visualizations: User interfaces for managing and exploring scientific visualization data. *IEEE Comput. Graph. Appl.*, 20(5):16–19, 2000.

[23] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, W. Ryall, J. Seims, and S. Shieber. Design galleries: A general approach to setting parameters for computer graphics and animation. In *ACM SIGGRAPH*, pages 389–400, 1997.

[24] Mercury Computer Systems. Amira. http://www.amiravis.com.

[25] MeVis Research. MeVisLab. http://www.mevislab.de.

[26] Microsoft. Intellisense. http://msdn.microsoft.com/en-us/library/hcw1s69b(VS.80).aspx.

[27] T. Munzner, C. Johnson, R. Moorhead, H. Pfister, P. Rheingans, and T. S. Yoo. NIH-NSF visualization research challenges report summary. *IEEE Computer Graphics and Applications*, 26(2):20–24, 2006.

[28] NAG. Iris Explorer. http://www.nag.co.uk/welcome_iec.asp.

[29] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos. A data mining algorithm for generalized web prefetching. *IEEE Transactions on Knowledge and Data Engineering*, 15(5):1155–1169, Sept.-Oct. 2003.

[30] S. G. Parker and C. R. Johnson. SCIRun: a scientific programming environment for computational steering. In *Supercomputing*, 1995.

[31] C. E. Scheidegger, H. T. Vo, D. Koop, J. Freire, and C. T. Silva. Querying and creating visualizations by analogy. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of Visualization)*, 13(6):1560–1567, 2007.

[32] H. Senay and E. Ignatius. A knowledge-based system for visualization design. *IEEE Comp. Graph. and Appl.*, 14(6), 1994.

[33] Swivel. http://www.swivel.com.

[34] S. Takahashi, I. Fujishiro, Y. Takeshima, and T. Nishita. A feature-driven approach to locating optimal viewpoints for volume visualization. In *IEEE Visualization*, pages 495–502, 2005.

[35] S. Tsang, R. Balakrishnan, K. Singh, and A. Ranjan. A suggestive interface for image guided 3d sketching. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI)*, pages 591–598, 2004.

[36] University of Utah. VisTrails. http://www.vistrails.org.

[37] C. Upson et al. The application visualization system: A computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):30–42, 1989.

[38] F. B. Viegas, M. Wattenberg, F. van Ham, J. Kriss, and M. McKeon. ManyEyes: a site for visualization at internet scale. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis)*, 13(6):1121–1128, 2007.

[39] U. D. Vordoloi and H.-W. Shen. View selection for volume rendering. In *IEEE Visualization*, pages 487–494, 2005.