Animated Visualization of Causal Relations Through Growing 2D Geometry[*]

Niklas Elmqvist[†] Philippas Tsigas[‡]

Department of Computer Science and Engineering
Chalmers University of Technology and Göteborg University
412 96 Göteborg, Sweden

Abstract

Causality visualization is an important tool for many scientific domains that involve complex interactions between multiple entities (examples include parallel and distributed systems in computer science). However, traditional visualization techniques such as Hasse diagrams are not well-suited to large system executions, and users often have difficulties answering even basic questions using them, or have to spend inordinate amounts of time to do so. In this paper we present the Growing Squares and Growing Polygons methods, two sibling visualization techniques that were designed to solve this problem by providing efficient 2D causality visualization through the use of color, texture, and animation. Both techniques have abandoned the traditional linear timeline and instead map the time parameter to the size of geometrical primitives representing the processes; in the Growing Squares case, each process is a color-coded square that receives color influences from other process squares as messages reach it; in the Growing Polygons case, each process is instead an n-sided polygon consisting of triangular sectors showing color-coded influences from the other processes. We have performed user studies of both techniques, comparing them with Hasse diagrams, and they have been shown to be significantly more efficient than old techniques, both in terms of objective performance as well as the subjective opinion of the test subjects (the Growing Squares technique is, however, only significantly more efficient for small systems).

Keywords: causal relations, information visualization, interactive animation

1 Introduction

It is part of human nature to not simply accept things as they are, but to search for reasons and to try and answer the question "why?". Thus, the concepts of *cause* and *effect* have always fascinated human beings, and also lie at the core of modern science. In order to fully understand the workings of a system, a scientist often needs to ascertain its underlying mechanisms by observing their visible effects. Or, as Aristotle puts it in *Physics II.3* [Aristotle 350 B.C.]:

> Since we believe that we know a thing only when we can say why it is as it is–which in fact means grasping its primary causes (*aitia*)–plainly we must try to achieve this [...] so that we may know what their principles are and may refer to these principles in order to explain everything into which we inquire.

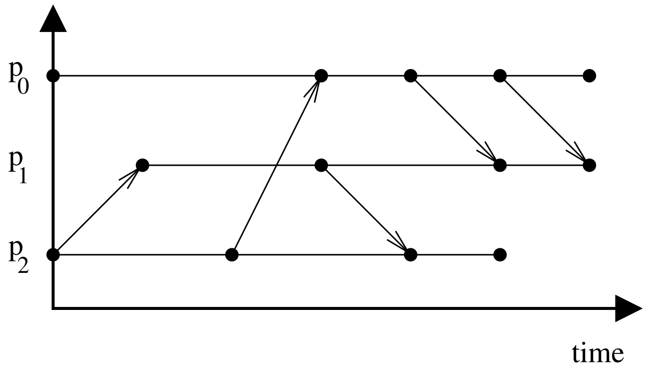


Figure 1: Hasse diagram visualization with 3 processes.

Humans are particularly apt at inferring the cause for simple physical processes merely by tracing its effects backwards, for instance by backtracking the path of a moving billiard ball on a pool table to identify the cue ball that struck it. However, as the number of action-reaction pairs grows, the human mind reaches a point when it is no longer able to cope. Continuing with the analogy above, fully comprehending the interactions, or *causal relations*, of all sixteen balls moving and colliding on the billiard table is impossible to do in real-time.

One way to allay this problem is to employ some kind of graphical visualization that presents the information in a more digestible format suitable for offline study. Simple directed-acyclic graphs (DAGs) or Hasse diagrams (also known as time-space diagrams) offer an intuitive view of these causal relations, but are unsuitable for studying the node dependencies and information flow in a system, especially when the number of nodes and interactions grow.

In this paper, we present two novel visualization techniques called Growing Squares and Growing Polygons, respectively, that attack the problem of effective causality visualization through the use of animation, colors, and patterns to provide an accessible overview of a system of causal relations. Both techniques abandon the traditional linear timeline of previous visualizations, and instead map the time parameter onto the size of the geometrical entities representing the processes (squares versus n-sided polygons, respectively). In the Growing Squares technique, we represent each process in the system as a color-coded square, laid out in a suitable way, and then intuitively "grow" these squares as time progresses. Events that causally relate the process squares influence their coloring, somewhat akin to how color pools would spread out on a piece of paper (see Fig-

[*]This is the authors' version of a journal paper that appeared in *Information Visualization*, Vol. 3 (2004) No. 3, pp. 154–172.
[†]e-mail: elm@cs.chalmers.se
[‡]e-mail: tsigas@cs.chalmers.se

ure 5). The Growing Polygons technique, on the other hand, is based on the idea of assigning each node in a system of n processes not only a color and but also a triangular sector in an n-sided polygon, and have each such process polygon grow and be subsequently filled with the colors of the processes influencing it. Since both the color and position of each process sector are invariant, distinguishing between individual processes is easier than for the Growing Squares technique and the visualization is therefore more scalable.

Chronologically, the Growing Squares method was devised as a first alternative to Hasse diagrams, and the Growing Polygons method was later designed to address some of the weak points of the Growing Squares. Both techniques have been implemented and tested as part of a visualization framework for causal relations we have developed, allowing us to compare the new methods with each other as well as with traditional techniques (see Figure 2). In addition, this framework allows the user to dynamically select different visualizations for the same system of causal relations, essentially making it possible for the user to harness the strengths of each technique dependent on the analysis task being performed.

A formative evaluation, using a focus group consisting of researchers working on distributed systems, was conducted at the onset of the project in order to identify the tasks associated with causal relations and to shape the design of the visualizations. The insights gained through these discussions were instrumental in guiding the development of both methods. Furthermore, formal user studies of both visualizations were performed to ensure the validity of our findings. The results from the Growing Squares study show that the Growing Squares method is significantly faster and more efficient than Hasse diagrams for sparse data sets. However, the new method is not significantly more efficient for dense data sets. Test subjects clearly favored Growing Squares over Hasse diagrams for all analysis tasks performed. Overall, the subjective ratings of the test subjects show that the Growing Squares method is easier, feels more efficient, and is more enjoyable to use than Hasse diagrams.

While the test subjects' opinion of the Growing Squares method were clearly favorable, the study revealed considerable room for improvement in the efficiency of the technique. Fortunately, the results from our study of the Growing Polygons method are much more positive: the improved method is significantly faster and more efficient than Hasse diagrams for *both* sparse and dense data sets when performing tasks related to information flow in a system (i.e. not only for sparse sets as for the Growing Squares method). In addition, subjects have a much higher correctness rate using our technique to solve tasks than when using Hasse diagrams. Furthermore, the subjective ratings of the subjects show that the new method, just as the previous Growing Squares method, is perceived as more efficient as well as easier and more enjoyable to use than Hasse diagrams.

The structure of this paper is as follows: We first describe the existing work in the field, followed by a background of causal relations, how to visualize them, and our software framework. Then, we describe the Growing Squares method, including details on design and implementation, and present the user study we conducted. After that, we introduce the improved Growing Polygons method and go through its design, implementation, and the user study we performed on it. The final sections of this paper deals with the results we obtained and our interpretation of them.

2 Related Work

There has been surprisingly little work performed in the area of causality visualization, and the prevalent visualization method is still the traditional Hasse (also known as time-space) diagram. Figure 1 shows an example of a time-space diagram for a system comprised of three processes, where the progress of each process is described by a directed horizontal line, the process line. Time is assumed to move from left to right. Events are symbolized by dots on the process lines, according to their relative order of occurrence. Messages are shown as arrows connecting send events with their corresponding receive events. Visualizations of causal relations in the form of such time-space diagrams are currently quite standard in visualization and debugging platforms for parallel and distributed systems, and the number of such platforms is too large to allow discussing them all; we will just focus on a few of the noteworthy systems. One of the first of the new generation of visualization tools to include the time-space diagram was the Voyeur [Socha et al. 1989] system, which provided a framework for defining various animation views for parallel algorithms. The TOPSYS [Bemmerl and Braum 1993] environment includes various standard concurrency visualizations (called VISTOP) integrated with the debugging and performance analysis tools of the system, with time-space visualization being one of them. Using this process-based concurrency view, users can identify synchronization and communication bugs. Going one step further, the conceptual visualization model of the VADE [Moses et al. 1998] system is based on the causal relation notion. VADE is also geared towards more general algorithm visualization, and supports not only communication events but also other algorithmic objects and events. Also of interest is LYDIAN [Koldehofe et al. 1999], an educational visualization system, which by default constructs the time-space diagram for every algorithm implemented in the system. Kraemer and Stasko [Kraemer and Stasko 1998] describe the essential characteristics of toolkits for visualization of concurrent executions, and introduce their own system, called Parade. Parade also includes an animation component called the Animation Choreographer that orders display events from a trace file in much the same way as the techniques described in this paper. Also, for the purpose of our study, the Hasse visualization used in Figure 1 is very similar to the time-space visualization view from the ParaGraph system [Heath 1990; Heath and Etheridge 1991] and its adaption in the PVaniM tool [Topol et al. 1998], as well as the Feynman or Lamport views from the Polka animation library [Stasko and Kraemer 1993].

While Hasse diagrams certainly are in widespread use, they have a number of deficiencies that lower their usefulness for realistic systems. First of all, a Hasse diagram offers only local dependency information for each process and not the transitive closure of all interactions involving it, making it difficult to gain an overview of the overall information flow in the system; in essence, the user is forced to manually backtrace every single message and process affecting a specific process to find its dependencies. Second, the fine granularity of the visualization makes Hasse diagrams difficult to use for large systems of ten or more involved nodes; the amount of intersecting message arrows simply becomes too overwhelming for complex executions. And third, Hasse diagrams are intrinsically static in nature and thus make little use of the interactiveness of the computer medium; animation and creative use of color are likely to be useful tools in this kind of visualization.

Ware *et al.* [Ware et al. 1999] presented a new visualization construct called a *visual causality vector* (VCV) that represents the perceptual impression of a causal relation and employed animation to emphasize this relation in a directed acyclic graph. Three different VCVs were introduced based on different metaphors: the pin-ball metaphor, where the VCV is a ball that moves from the source to the destination node, striking the destination and making it oscillate; the prod metaphor, where the VCV is a rod that extends from the source to prod the destination; and finally a wave metaphor, where the VCV accordingly is an animated wave that moves towards the destination node. However, while these constructs are certainly an improvement over a simple DAG representation of causal relations, they do nothing to battle the complexity of large systems with many nodes and relations. In fact, Ware's primary contribution is the investigation of timing concerns for the perception of causality for users, not the visualization technique *per se*. It might still be interesting to incorporate Ware's VCVs into our system in some form.

3 Background

In modern use, the notion of causality is associated with the idea of something (the cause) producing or bringing about something else (its effect). In general, the term "cause" has a broader meaning, equivalent to an explanatory or reasoning tool. Identifying causal relations in a complex system can be the first step towards understanding the underlying mechanisms that determine the system's laws. As such, causal relations cover a wide variety of software domains where causality are of importance.

More specifically, causal relations play a vital role in understanding how any kind of complex system works, especially those involving several concurrent processes interacting with each other. Our interest originates mainly from the viewpoint of distributed and parallel computing, where causal relations are used extensively for example (i) in distributed database management to determine consistent recovery points; (ii) in distributed software systems for determining deadlocks; (iii) in distributed and parallel debugging for detecting global predicates and detecting synchronization errors; (iv) in monitoring and animation of distributed and parallel programs to determine the sequence in which events must be processed so that cause and effect appear in the correct order; and (v) in parallel and distributed software performance to determine the critical path abstraction: the longest sequential thread, or chain of dependencies, in the execution of a parallel or distributed program. Improving the graphical visualization of causal relations will thus benefit all these activities.

In this section we give a brief background to the causality visualization problem, including a brief formal introduction to causal relations, a description of the various analysis tasks involved when studying causal relations, and a presentation of CausalViz, our software platform for causality visualization.

3.1 Causal Relations

A causal relation is the relation that connects or relates two items, called *events*, one of which is a cause of the other. Obviously, for an event to cause another, it is not sufficient that the second merely happens after the first; however, it is well accepted to state that this is necessary, and temporal order can be relied on to explain the asymmetrical direction of causal relations[1]. All events connected in the causal relation are part of a set of *processes*, labelled $P_1, \ldots, P_N$, each of which can be thought of as a disjoint subset of the set of all events in a system. Events performed by the same process are assumed to be sequential; if not, we can split the process into sub-processes. Thus, it is convenient to index the events of a process P_i in the order in which they occur: $E_i = e_1^i, e_2^i, e_3^i, \ldots$

For our purposes, it suffices to distinguish between two types of events; *external* and *internal* events. Internal events affect only the local process state. An internal event on process P_i will causally relate to the next event on the same process. External events, on the other hand, interconnect events on different processes. Each external event can be treated as a tuple of two events: a *send* event, and a corresponding *receive* event. A send event reflects the fact that an event, that will influence some other event in the future, took place and its influence is "in transit"; a receive event denotes the receipt of an influence-message together with the local state change according to the contents of that message. A send event and a receive event are said to correspond if the same message m that was sent in the send event is received in the receive event.

We now formally define the binary causal relation $\rightarrow$ over all the events of the system E ($\rightarrow \subseteq E \times E$) as the smallest transitive closure that satisfies the following properties [Lamport 1978]:

1. If e_k^i, $e_l^i \in E_i$ and $k < l$, then $e_k^i \rightarrow e_l^i$.
2. If $e^i = send(m)$ and $e^j = receive(m)$, then $e^i \rightarrow e^j$ where m is a message.

When $e \rightarrow e'$, we say e causally precedes e' or e caused e'. Causal relations are irreflexive, asymmetric, and transitive.

3.2 Analysis Tasks

At the onset of our investigation into visualization of causal relations, we organized a formative evaluation of these concepts using a focus group consisting of researchers from our university working on distributed systems. The evaluation took the shape of a panel discussion on questions related to causal relations and their use, and six researchers from the Distributed Computing & Systems group at the Department of Computing Science at Chalmers participated in the session. These discussions allowed us to identify the typical analysis tasks a user is interested in when studying a distributed system, and were vital in tailoring our visualization to these tasks. Below follows a short overview of these analysis tasks.

Lifecycle Analysis The lifecycle of individual processes are often of great interest when analyzing a system of causal relations. This includes aspects such as the duration of a process as well as its starting and stopping times (both in isolation as well as in relation to other processes), aspects that are vital in understanding how a system works.

[1]It has been argued that not even this is necessary, and that both simultaneous causation and "backwards causation" (effects preceding their causes) are at least conceptually possible. This, on the other hand, causes problems when considering the asymmetric nature of causal relations.

Influence Analysis The analysis of influences and dependencies in a distributed system was found to be one of the most important analysis tasks when studying the flow of information in a system. Designing, debugging, or trying to grasp the underlying mechanisms of a distributed system or algorithm all involve this task.

Inter-Process Causal Relations Often, a practitioner studying a system of causal relations needs to know whether two nodes, P_i and P_j, in the system are causally related, i.e. if there exists an event $e^i \in E_i$ and an event $e^j \in E_j$ such that $e^i \rightarrow e^j$. Of course, this causal relation can go through several levels of transitive indirection, and is therefore quite difficult to spot manually or by using Hasse diagrams (as we will see).

3.3 The CausalViz Framework

In order to test the Growing Squares and Growing Polygons techniques and to subsequently be able to perform user studies on their effectiveness, we implemented a general application framework for the visualization of causal relations called CausalViz (see Figure 2). The framework is implemented in C++ on the Linux platform and uses the Gtk+/Gtk– widget toolkits for user interface components as well as OpenGL for graphical rendering.

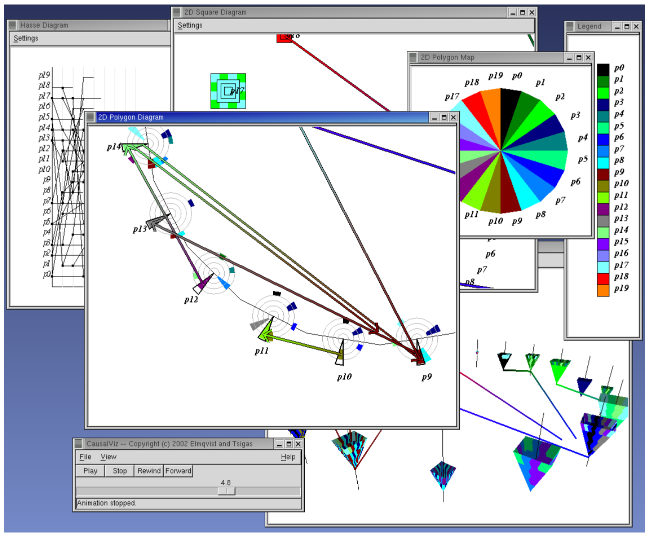


Figure 2: The CausalViz application.

3.3.1 System Architecture

The architecture of the CausalViz application (see Figure 3) is based around a single partially ordered set (*poset*) representing the execution data under study. A number of visualization components observe this set and present graphical representations of the data (potentially allowing for the set to change during run-time). There currently exists three different visualizations, i.e. traditional Hasse diagrams, the 2D Growing Squares, and the prototype 3D Growing Pyramids.

Central in the system architecture is the *application manager* that creates all the other components, manages the graphical user interface (GUI), and performs loading of data files into the application (stored in a general XML format for partially ordered sets). In order to allow for the animation of events in the visualizations, there also exists a general *animation manager* thread that the visualization components can use to smoothly interpolate values in the poset with respect to time.

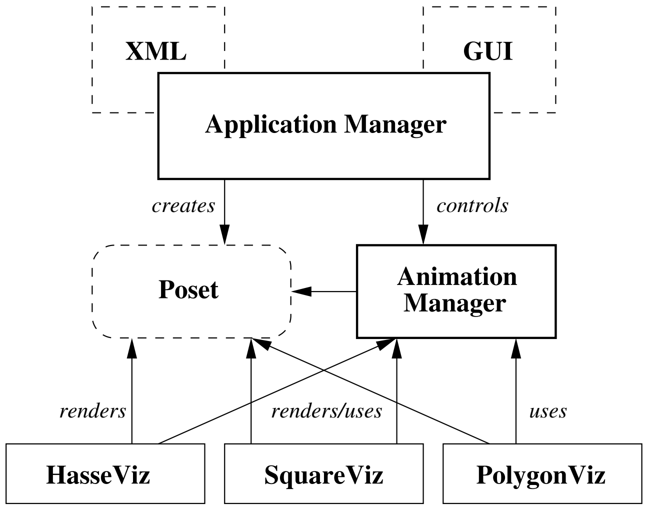


Figure 3: CausalViz system architecture.

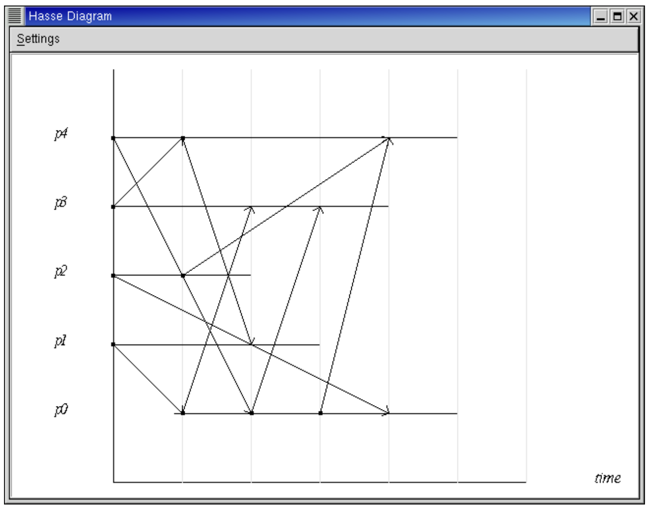


Figure 4: CausalViz Hasse visualization.

3.3.2 Poset Management

System execution traces are stored in a general XML file format for partially ordered sets. Here, a process P_i is represented by the subset $E_i \subseteq E$ of all the events in the system belonging to the process and a set of messages M_i. Messages are partial orderings between events in different subsets (processes), and can thus be represented by pairs of events, i.e. $M_i \subseteq E \times E$. It is then up to the application to compute the minimal transitive closure for the poset.

In the CausalViz application, the transitive closure is computed using a modified topological sort [Cormen et al. 2001].

The objective of the algorithm is two-fold: (i) to derive the transitivity information for each event (i.e. the processes which have influenced it so far) and (ii) to assign the event to a discrete time slot. This is done by greedily consuming sequential events in each subset (i.e. process) of the poset until reaching an event with unresolved dependencies (i.e. a partial ordering to a previously unvisited event). When this happens, the algorithm moves on to the next process to continue from where it last left off. This is repeated until all events in the system have been visited. The current influence of each event is easily maintained and updated during this process, and illegal cyclic dependencies are trivially detected by checking whether the algorithm has cycled through all process without visiting any new events.

4 Growing Squares

As described earlier, there is surprisingly little work on visualizations of causal relations besides various implementations of Hasse diagrams, a fact which is especially curious in light of the shortcomings of Hasse diagrams for understanding a distributed system. The fine granularity of Hasse diagrams defeat their use as overview tools, and they transfer the burden of maintaining transitive relations to the user herself. This means that a user studying the information flow in a distributed systems visualized using a Hasse diagram might potentially have to backtrace every single message and process in order to get a clear picture of the influences in the system.

The Growing Squares visualization technique (first presented in [Elmqvist and Tsigas 2003b]) was designed to help the user quickly get an overview of the causal relations in a system by making use of animation, color and patterns in an intuitive way. The visual metaphor of the technique is that of "pools" of color spreading on a piece of paper as time progresses, each color and pool representing a specific process or node in the system. Messages in the system are shown as "channels" from one pool to another. Each color pool will start growing at the time its corresponding process is started, and accordingly stop growing when the process stops executing events. The channels representing messages from one process to another intuitively carry the color of its source with it, resulting in the destination pool receiving this color as well. However, like age rings on a tree, the color of the new influencing process will only be present in the destination process starting from when the message was received.

Figure 5 gives an example of a system with two processes, P_0 and P_1, colored blue and white, respectively. The color pools are represented as 2D squares which grow over time. At a certain time t, P_0 sends a message to P_1 (denoted by the arrow in the figure), establishing a causal relation between P_1 and P_0. For all times $t' > t$, the color pool of process P_1 now shows this influence from the blue P_0 by means of a checkered pattern combining the two colors.

In order to visualize the transitive property of the causal relation (see the previous section), a similar color pattern scheme is used. In Figure 6, process P_1 is sending a message to P_2 (colored red) *after* having been influenced by a message from P_0. Now, both the color of the source process (white from P_1 itself) and any of its existing influences at the time of sending the message (blue from P_0) are transferred to P_2, making its texture from this time and onwards be a checkered pattern of all of the three colors. It is now easy to see that P_2 is causally related to both P_0 and P_1.

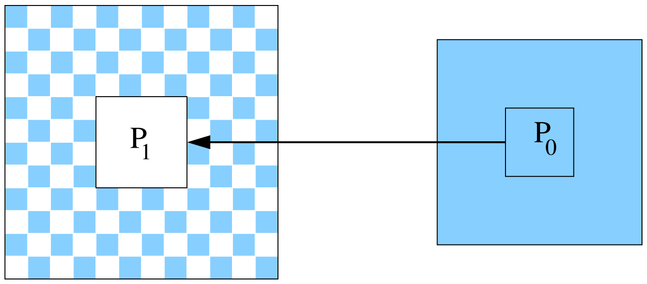


Figure 5: Simple example of the Growing Squares technique with two processes.

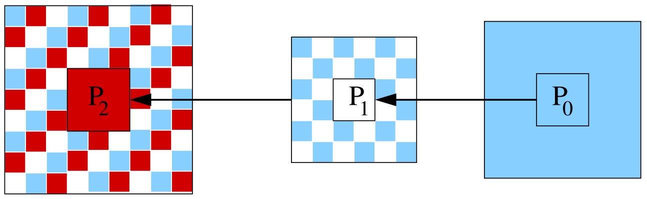


Figure 6: Transitivity property of causal relations using Growing Squares.

Multiple influences from the same source process will increase the amount of the source process's color in the texture of the destination process. Even if the checkered pattern makes it difficult to see the exact ratio, this fact can nevertheless be used as a visual indication that multiple influences have occurred.

Having foregone a traditional timeline, the Growing Squares method is dependent on animation to allow the user to view the entire execution of the system under study. Starting at $t = 0$, the user can advance the time in the system to observe the system execution in chronological order, or choose to view the situation at specific points in time. This is another radical difference from Hasse diagrams; Hasse diagrams are static in nature and do not benefit much from animation, whereas Growing Squares are dynamic and rely on animation to present the full data set to the user.

Figure 14 shows an example sequence consisting of 5 processes in a distributed system visualized using the Growing Squares technique. The state of the visualization is here shown for each discrete time unit (in practice, the animation is fluid and continuous between the time steps) starting at $t = 1$ and ending at $t = 5$, the end of the execution. Processes are laid out in a clock-wise fashion with P_0 at the top. Screenshot (a) at $t = 1$ shows how P_1 sends a message to P_0, starting it (it has zero size up until this time), and (b) at $t = 2$ depicts the two colors (green and black) in the process square of P_0. In the same screenshot, P_4 sends a message to P_1, causing P_1 in (c) at $t = 3$ to hold influences from both P_4 as well as indirectly from P_3 (i.e. an example of the visualization of transitivity in the Growing Squares visualization). In (d) at $t = 4$, two messages originating from the otherwise isolated P_2 reach P_0 and P_4, its blue color showing in the outer square of these two processes in snapshot (e) at $t = 5$.

4.1 Design

In order for the Growing Squares visualization to be effective, users must be able to easily distinguish between the individual process colors in the system under study. Selecting a suitable color scale is thus an important aspect of the method, and we investigated the use of perceptually uniform color scales such as LOCS [Levkowitz and Herman 1992; Levkowitz et al. 1992] for this purpose. However, we found that the continuous nature of LOCS was not well-suited to our problem since it made distinguishing between adjacent colors difficult, and the scale itself included an inordinate amount of dark colors. Instead, we opted for a simple color scale with the individual colors uniformly distributed over the RGB spectrum[2].

One of the central features of the presented visualization technique is that it draws process squares with the checkered patterns containing all the colors of the processes that have influenced the process. If the number of influences is large, the on-screen space allocated for each color will be very small and thus hard to distinguish (see [Wyszecki and Stiles 1991] for in-depth information on color perception). In order to still allow the visualization to be effective, we need a zoom function that allows the user to effortlessly view the graphical representation at different magnification levels. We have implemented a simple continuous zoom mechanism for this purpose; in the future, it may be extended to borrow techniques from the Pad [Perlin and Fox 1993] zoomable user interface and its descendants [Bederson et al. 1996; Bederson et al. 2000].

It might be argued that using circles instead of squares would have been more in keeping with the metaphor of color pools spreading on a piece of paper. Our original intention was also to use circles, but we ultimately chose squares for a number of reasons: (i) the larger area of squares facilitates color recognition better than circles, (ii) the layout of process squares into grids is easier (no wasted space), and (iii) squares are faster to render and easier to texture map (besides, we felt it was more logical to have checkered squares rather than checkered circles).

The Growing Squares visualization makes use of animation to display the dynamic execution of the system under study. While it certainly is possible to maintain all message arrows and just draw the visualization at full time, this would result in many of these messages coinciding (as in Hasse diagrams) and thus being hard to separate from other messages, as well as being impossible to associate with a specific time. Animation solves these issues in a natural way.

Another design aspect of the Growing Squares visualization is finding suitable layout methods for arranging the individual processes. Many such layout strategies exist. For instance, if the data set represents the execution of a distributed algorithm in a network, the geographical location of the individual nodes can be used to position the squares in the visualization. Other alternatives include simple grid and circular layouts (see Figure 7) which may serve to minimize the amount of coinciding message arrows to greater or lesser extent. In this paper, we chose to ignore this aspect and selected a simple circular layout scheme that has the advantage of avoiding message arrows coinciding with each other or passing over processes.

[2]The RGB color model was chosen for simplicity, while a color model like HSV might be more suitable to human perception.

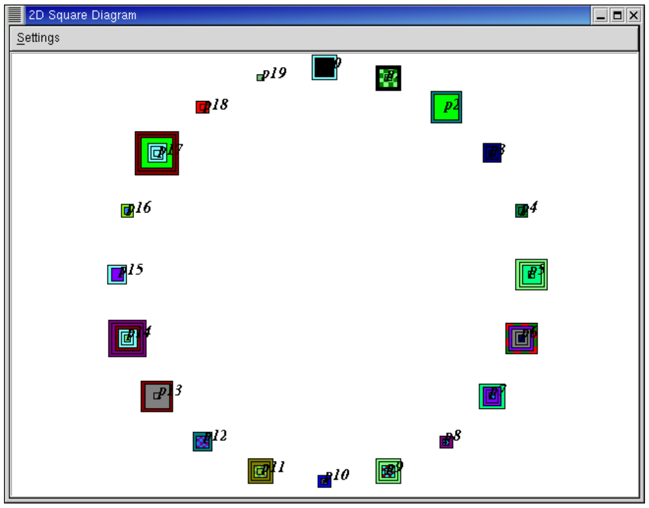


Figure 7: Growing Squares visualization with 20 processes.

4.2 User Study

Our hypothesis was that the Growing Squares technique is faster and more efficient at quickly providing an overview of the causal relations in a distributed system, and that the new technique scales better with system size than traditional methods. To test this, we conducted a formal comparative user study of the old Hasse diagram visualization and our new Growing Squares technique. The focus of this user study was to evaluate user performance of the "overview tasks", i.e. tasks associated with the general comprehension of how a system works. We also wanted to get a subjective assessment of the two methods.

4.2.1 Subjects

Twelve users, four of which were female, participated in this study. All users were carefully screened to have good computer skills and basic knowledge of distributed systems and general causal relations. In particular, knowledge of Hasse diagrams was required. Subject ages ranged from 20 through 50 years old, and all had normal or corrected-to-normal vision.

4.2.2 Equipment

The study was run on a Intel Pentium III 866 MHz laptop with 256 MB of memory and a 14-inch display. The machine was equipped with a NVidia Geforce 2 GO graphics accelerator and ran Redhat Linux 7.2.

4.2.3 Procedure

The experiment was a two-way repeated-measures analysis of variance (ANOVA) for independent variables "visualization type" with two levels (Hasse diagrams versus Growing Squares), and "data density", also with two levels. The two levels of data density were "sparse" and "dense" with 5 processes sending 15 messages and 30 processes sending 90 messages, respectively. The visualization type was a within-subjects factor, as was the data density. Each subject received the various task sets in different order to avoid systematic effects of practice.

The same set of four different data sets were used for all subjects. Two were geared at the sparse case with 5 processes and 15 messages (one for each visualization type), and two for the dense case with 30 processes and 90 messages (see Table 1). The traces were all generated using a heuristic algorithm to avoid users taking advantage of special knowledge about real system traces. In the case of deducing inter-node causal relations, care was taken to ensure that the complexity of this was equivalent for both task sets of each density.

The evaluation procedure consisted of repeating overview tasks using Hasse diagrams and Growing Squares for first the sparse and then the dense data densities. The order of the visualization types was different for each subject to minimize the impact of a learning effect. The repeated tasks for each density and visualization type is summarized in Table 2. Prior to starting work on each task set, subjects were given the chance to adjust the window size and placement to their liking. Subjects were informed that they should solve the tasks quickly and focus on using the visualization to get an overview of the system trace. The completion of each task was separately timed, except for the tasks Causality 1-3, which were timed together.

We enforced an 8 minute (480 seconds) time cap on the completion of each task in order to avoid excessive times skewing the results of the user study. Uncompleted or skipped tasks were set to the time cap for that particular task.

Since we were targeting overview tasks, it was not necessary for subjects to find a precise answer to each exercise. Instead, it was deemed sufficient if subjects named one of the processes in the top 20 %[3] for each category; i.e. for 30 processes, it was enough to pick one of the six processes that were most the influential, long-lived or influenced ones for the answer to be counted as correct. Only the Causality 1-3 tasks required a totally accurate answer.

After having performed each task set for a density and visualization type, subjects were asked to give a subjective rating of the efficiency, ease-of-use, and enjoyability of the visualization technique. When all of the tasks were completed, the subjects responded to a final questionnaire comparing the two visualization techniques based on the previously-stated criteria (see Table 3).

Each evaluation session lasted approximately one hour. Subjects were given a training phase of ten minutes to familiarize themselves with the CausalViz application and the two visualization techniques. During this time, subjects were instructed in how to use the visualizations to solve various simple tasks.

Data Density	**Processes**	**Messages**
Sparse	5	15
Dense	30	90

Table 1: Experimental design. Both density and visualization factors were within subjects for all 12 subjects.

[3]This number was somewhat arbitrarily chosen, partly because it was felt to be an acceptable margin of error, and partly because 20 % out of 5 processes for the sparse data set translates to finding the single correct process for each task.

Task	**Comments**	**Measure**
Duration	Find the process with the longest duration.	Time
Influence 1	Find the process that has had the most influence on the system.	Time
Influence 2	Find the process that has been influenced the most.	Time
Causality 1-3	Is process x causally related to process y?	Time
Q1	Rate the visualization w.r.t. ease-of-use (1=very hard, 5=very easy).	Likert
Q2	Rate the visualization w.r.t. efficiency (1=very inefficient, 5=very efficient).	Likert
Q3	Rate the visualization w.r.t. enjoyability (1=very boring, 5=very enjoyable).	Likert

Table 2: Repeated tasks for each density and visualization type.

Task	**Comments**
PQ1	Rank the visualizations w.r.t. ease of use.
PQ2	Rank the visualizations w.r.t. efficiency.
PQ3	Rank the visualizations w.r.t. enjoyability.

Table 3: Post-evaluation ranking questions.

4.3 Results

After having conducted the user study, we analyzed the resulting test data. The results can be divided into two parts; the objective performance measurement, and the subjective ratings of the test subjects.

4.3.1 Performance

The mean times of performing a full task set (i.e. four tasks) using the Hasse diagrams and the Growing Squares visualizations were 416.58 (s.d. 268.99) and 334.79 (s.d. 230.86) seconds respectively. This, however, is not a significant difference ($F(1, 11) = 2.54$, $p = .139$). The main effect for density was strongly significant ($F(1, 11) = 30.99$, $p < .001$), with means for the sparse and dense conditions of 222.96 (s.d. 77.24) and 528.42 (s.d. 272.94) seconds. Figure 8 summarizes the mean task results for the two visualizations across the two densities; error bars show one standard deviation above and below the mean. The figure also shows that the mean time for the task set was higher for the Hasse method across all densities. For the sparse conditions the visualization type was significant ($F(1, 11) = 15.82$, $p = .002$), with mean values of 259.50 (s.d. 75.23) and 186.42 (s.d. 62.46) seconds for the Hasse and Growing Squares visualizations. The Growing Squares method also gave better results for dense conditions; the mean times in Hasse and Growing Squares were 573.67 (s.d. 302.96) versus 483.17 (s.d. 243.94) seconds. This, however was not a significant difference ($F(1, 11) = 1.03$, $p = .332$).

The only exception where Hasse diagrams performed better than Growing Squares is the Duration subtask for dense systems, while our technique performed better than Hasse diagrams in all other subtasks across both densities. For the Duration subtask, the mean completion times for the sparse data set using Hasse diagrams were 30.92 seconds (s.d. 9.99) versus 21.17 seconds (s.d. 17.93) for the Growing Squares method, while the mean times for the dense set were 37.00 (s.d. 15.72) and 54.75 (s.d. 28.08), respectively. This, however, was not a significant difference for this subtask ($F(1, 11) = 0.492$, $p = 0.498$). For the Influence 1 subtask the mean completion times for the sparse data set using Hasse diagrams were 75.33 seconds (s.d. 50.71) versus 65.33 seconds (s.d. 35.93) for the Growing Squares method, while the mean times for the dense set were 234.67 (s.d. 141.81) and 157.17 (s.d. 66.85), respectively. The visualization type did not have a significant effect on the completion time for this subtask ($F(1, 11) = 2.80$, $p = 0.122$). Similarly, the Influence 2 subtask yielded mean completion times of 76.17 (s.d. 31.94) versus 47.17 (s.d. 31.65) for the sparse data set, and 165.58 (s.d. 159.21) versus 136.00 (s.d. 114.83) for the dense case. Again, the type of visualization did not have a significant effect to the completion time for this subtask ($F(1, 11) = 1.062$, $p = 0.325$). Finally, the Causality 1-3 subtask resulted in sparse mean completion times of 77.08 (s.d. 31.04) for Hasse diagrams and 52.75 (s.d. 13.32) for Growing Squares, whereas the dense means were 136.42 (s.d. 88.25) and 135.25 (s.d. 77.87), respectively. The type of visualization did not have a significant effect to the completion time for this subtask ($F(1, 11) = 0.707$, $p = 0.418$).

The subjects' comments revealed that one of the reasons for the absence of a statistically significant difference between visualizations in the dense condition was because of color similarities. Much time was spent by subjects matching colors to each other and looking up process numbers in the color legend.

Subjects made little use of the animation controls in the Growing Squares visualization except to play it through once at the beginning of each task to gain a picture of the data set. Only a few of the subjects actively moved the timeline back and forth to solve various subtasks, and most preferred to leave the time setting at the end of the execution.

The fixed (circular) layout algorithm used in the user study turned out to be limiting when it came to comparing the size (i.e. duration) of individual processes. Users remarked that it would have been useful to be able to click and drag processes to arbitrary positions to facilitate comparison as well as to group processes into semantic clusters (i.e. clusters of the same perceived type).

4.3.2 Subjective Ratings

The subjects consistently rated Growing Squares above Hasse diagram with respect to efficiency, ease-of-use and enjoyment. The mean response values to the five-point Likert-scale questions are summarized in Figure 9. The complete data analysis table is presented as Table 5.

The subjects' responses to the efficiency question (Q2, Table 5) showed a higher rating for the Growing Squares visualization than Hasse diagrams in both sparse (means 3.83 (s.d. .39) and 2.75 (s.d. .97)) and dense data densities (means 3.13 (s.d. .68) and 1.58 (s.d. .67)). Both higher

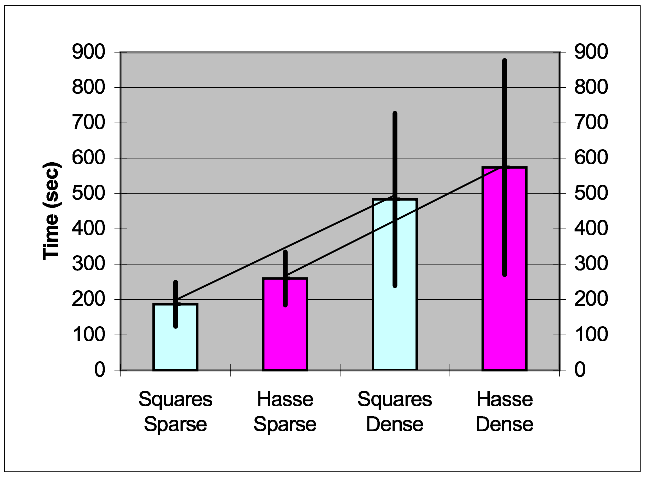


Figure 8: Mean task completion times for all tasks across the Hasse and Growing Squares methods and across levels of density. Error bars show standard deviations.

rating readings were significant (Friedman Tests, $p = .0209$ for the sparse case and $p = .0039$ for the dense case). The subjects' response to the ease-of-use question (Q1, Table 5) also showed a higher rating for the Squares visualization in both sparse (means 3.92 (s.d. .67) and 2.67 (s.d. .89)) and dense data densities (means 2.79 (s.d. .78) and 1.46 (s.d. .66)). Both higher rating readings were significant (Friedman Tests, $p = .0094$ for the sparse case and $p = .0015$ for the dense case). The subjects' response to the enjoyment question (Q3, Table 5) also showed a higher rating for the Squares visualization in both sparse (means 3.92 (s.d. .79) and 3.00 (s.d. .43)), and dense data densities (means 3.25 (s.d. .85) and 1.92 (s.d. .67)). Both higher rating readings were significant (Friedman Tests, $p = .0094$ for the sparse case and $p = .0015$ for the dense case).

Figure 9 shows, not surprisingly, that the density of the data set strongly influenced the subjects' responses to each question for both visualizations. This difference is reliable for all but the enjoyability question (Friedman Tests). The subjects' response to this question (Q2, Table 5) when using the Growing Squares visualization shows a higher rating when small data sets are considered (means 3.92 (s.d. .79) for sparse sets and 3.25 (s.d. .75) for large sets), but on the other hand, this is not a significant difference ($p > .05$).

The final ranking questionnaire shows that most subjects preferred the Growing Squares technique over Hasse diagrams with regard to ease of use, efficiency, and enjoyment (Table 4). Overall, the results from this ranking are very favorable for the Growing Squares method.

Question	Prefer GS?	
PQ1	Rank visualizations w.r.t. ease-of-use.	92 %
PQ2	Rank visualizations w.r.t. efficiency.	83 %
PQ3	Rank visualizations w.r.t. enjoyability.	92 %

Table 4: Subject responses to ranking the two visualization.

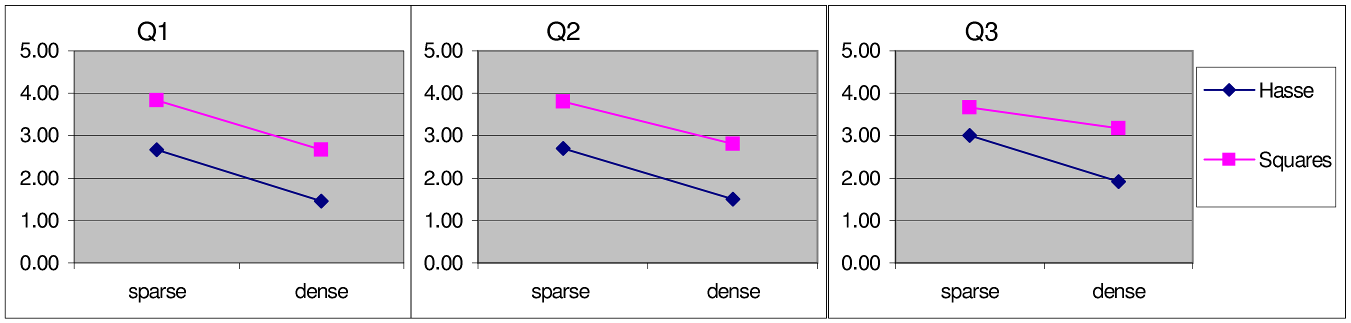


Figure 9: Responses to Q1-Q3 5-point Likert-scale questions across sparse and dense data densities for the Hasse and Growing Squares methods.

Question	**Hasse diagrams**		**Growing Squares**		**Reliability**	
	sparse	**dense**	**sparse**	**dense**	**Hasse/GS**	**Density**
Q1. Rate the visualization w.r.t. ease-of-use.	2.67 (.89)	1.46 (.66)	3.92 (.67)	2.79 (.78)	yes	yes
Q2. Rate the visualization w.r.t. efficiency.	2.75 (.97)	1.58 (.67)	3.83 (.39)	3.13 (.68)	yes	yes
Q3. Rate the visualization w.r.t. enjoyability.	3.00 (.43)	1.92 (.67)	3.92 (.79)	3.25 (.75)	yes	yes/no*

* Density does not significantly influence the enjoyability of the Growing Squares animation.

Table 5: Mean (standard deviation) responses to 5-point Likert-scale questions. Reliability is defined as being significant at the .05 level.

5 Caveats of Growing Squares

The Growing Squares technique is based on animation, colors and patterns to improve the perception of causality in distributed systems, and the results from the user study show that the technique is consistently faster and more efficient than Hasse diagrams. This difference, however, is not statistically significant for the general case, although it is significant for the sparse data set case. While there clearly is room for improvement, the Growing Squares visualization technique is nevertheless an improvement over conventional Hasse diagrams.

However, as indicated by the user study, the Growing Squares technique has a number of issues. First and foremost, since the method is dependent on a simple color coding for each process in a system, it is often very difficult to distinguish individual processes in a large system due to the similarity of the colors. This problem is exacerbated by the fact that Growing Squares presents the influences of a single process as colored pixels in a checkered pattern on each square, meaning that each influence can become arbitrarily small due to limited screen space (this problem is partially solved using a continuous zoom mechanism, however). And finally, a Growing Squares visualization does not explicitly communicate the absolute timing of events or process startup or shutdown; this must be manually deduced by studying the animated execution of the system.

6 Growing Polygons

Visualizing the causal relations in a system consisting of n processes using the Growing Polygons [Elmqvist and Tsigas 2003a] technique is done by placing n-sided polygons (so-called *process polygons*) representing the individual processes on the sides of a large n-sided polygon (the *layout polygon*). Instead of using a linear timeline, as in Hasse diagrams, the time parameter is mapped to the size of each process polygon so that they grow from zero to maximum size as time proceeds from the start to the end of the execution, just like in the Growing Squares technique. The visualization is animated to allow the user to study the dynamics of the execution, and the discrete time steps are shown as dashed or greyed-out "age rings" in the interior of each polygon. In addition to this, each process polygon is divided into triangular sections, with every process in the system being assigned a color and a specific sector in the polygon. This sector also corresponds to the side where the process polygon is positioned on the layout polygon. Whenever the process represented by a particular polygon is active, the appropriate time segments of the associated sector in the polygon will be filled in with the process color. Messages between processes in the system are shown as arrows travelling from the source polygon to the destination, and will activate the corresponding sector in the destination polygon with the color of the source process. In other words, a message sent from process A to process B will contaminate A's sector in B starting from the time the message was received.

Figure 10 shows an example of a simple 3-process system (consisting of processes P_0, P_1, and P_2) where each process is represented by a triangle partitioned into three sections, and with the process triangles positioned on the sides of a larger layout triangle. For each process triangle, the process's own sector has been marked with a thick black outline, and the internals of each polygon has also been segmented to show the discrete time steps of the execution. In addition, the processes have been assigned the colors red, green, and blue, respectively. In this example, we see how P_0 sends a message to P_1 at $t = 0$ that reaches the destination process at time $t = 1$, establishing a causal relation between the two nodes.

Notice how for all times $t \geq 1$, P_0's sector within P_1's process triangle is now filled, signifying this influence. By studying the polygons at $t = t_{end}$, i.e. the end of the execution, we can get a clear picture of the flow of information within the system.

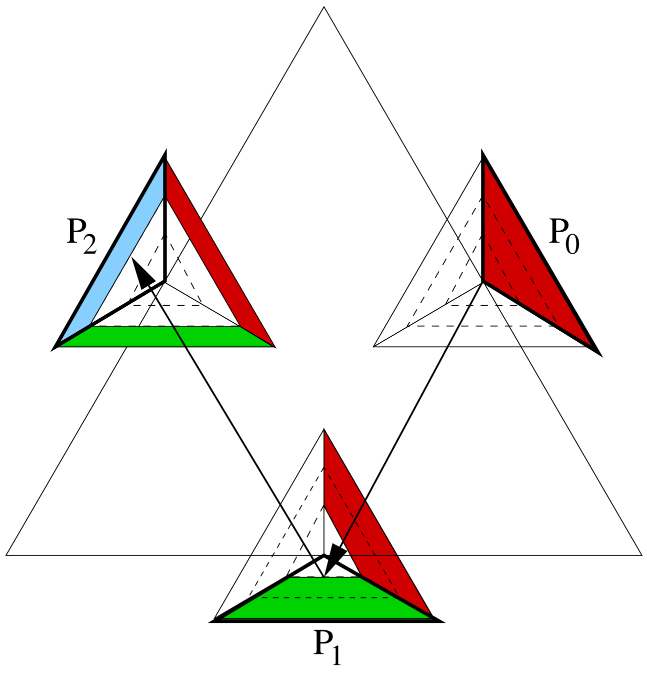


Figure 10: Growing Polygons visualization with $n = 3$ (i.e. the process polygons are triangles).

As we ascertained earlier, causal relations are transitive, so if $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$. Figure 10 also shows how this is expressed in the Growing Polygons visualization. At time $t = 2$, process P_2 receives a message from P_1. P_1 has already been influenced by P_0 in the previous interaction (in other words, there is already a causal relation between P_0 and P_1). Thus, the process triangle of P_2 now shows causal influences in *all* of its process sectors, including the transitive dependency to P_0, not just the direct dependency to P_1 which sent the actual message.

The simple execution in Figure 10 also gives information about the absolute lifecycles of the three processes. By studying the filled segments of each process triangle's own sector, we note that only process P_0 executed from the start to the end of the system trace; processes P_1 and P_2 were kickstarted by external messages at times $t = 1$ and $t = 2$, respectively. In fact, unlike the Growing Squares technique, the new method allows users to deduce the exact timing of all events in a system since the age rings in the interior of each polygon are fixed to absolute times.

Just like the Growing Squares technique, the Growing Polygons technique offers a view of the transitive closure of the node dependencies and influences, facilitating analysis of global information flow in the system (and not just locally, as for Hasse diagrams). The visualization is animated and can thus also avoid many of the message intersection problems of Hasse diagrams. In addition to this, by assigning not only a color but also a fixed polygon sector to each process, the Growing Polygons method largely remedies the difficulties of distinguishing colors that plague the Growing Squares technique. Thus, the new method is considerably more scalable than the old one since it is now enough that two similar colors are not placed in adjacent sectors for a user to be able to separate them.

Now let us study a full example to see the Growing Polygons visualization in action. Figure 15 shows a sequence of screenshots taken at the discrete time steps of the execution of a 5-process system of (in the real visualization, these images are smoothly animated). The processes are laid out in clockwise order with P_0 at the top right. In (a), at $t = 1$, we see that all processes except P_0 are executing and sending messages (the process sector of P_0 is empty). However, a message from P_1 is just about to reach P_0 and will activate it starting from this point in time. Screenshot (b) shows the subsequent situation at $t = 2$, where P_0 now has begun executing and exhibits a causal dependence to the green process (P_1) that started it, and where P_4 similarly shows a dependence to P_3 (P_3's sector in P_4's process polygon is filled in from time step 1 and onwards). Moving to $t = 3$ in (c), we see more causal dependencies appearing in the process polygons of the various nodes, the transitive dependencies in both P_1 (cyan from P_3) and P_3 (green from P_1) being of special interest. We can also observe that process P_2 appears to have stopped executing since it is no longer filling up its own process sector. Image (d) displays the situation one time step later ($t = 4$), where the two messages from the inactive P_2 finally reach P_0 and P_4 respectively, and image (e) shows the final situation at $t = 5$, with the causal dependencies in the system plainly visible.

6.1 Design

One of the weaknesses of the original Growing Squares method that limited its scalability was the difficulties of distinguishing between different process colors. To remedy this problem, the Growing Polygons technique also assigns a unique triangular sector to each process. Nevertheless, for our method to work efficiently, adjacent process sectors should not have similar colors, or users can easily mistake one process for another. Just like in the Growing Squares case, we opted for a straightforward non-continuous distribution of colors across the RGB spectrum.

While our new method does not exhibit the same congestion of screen space that plagues Growing Squares, where a much-influenced process square simply cannot convey all of its influences in its limited screen space, there are instances where even Growing Polygons fail at this. For example, when visualizing a large system with many processes, the angle ($\theta = 360°/n$) assigned to each process sector will be small, making it difficult to distinguish events early on in the execution. The same is also true if the time span of the execution is long, since the layout algorithm will then have to scale each time step to fit inside the allocated maximum size of each polygon. To cope with these two situations, the Growing Polygons visualization retains the simple continuous zoom mechanism of the Growing Squares technique, allowing users to zoom in arbitrarily close in order to distinguish details in the visualization.

The decision to use animation in the Growing Polygons technique was mainly grounded on the wish to avoid a maze of cris-crossing message arrows (like in Hasse diagrams). At the end of the system execution, no message arrows at all are visible, facilitating easy study of the inter-process depen-dendencies in the system. Animation allows the user to still see the dynamic execution of the system in an intuitive way, just like in the Growing Squares technique.

6.2 User Study

Our intention with the Growing Polygons technique was to provide an efficient way of viewing the flow of information and the node dependencies in a system of communicating processes. In order to check whether our method performs better than existing methods, we conducted a comparative user study between Hasse diagrams and Growing Polygons. The study involved test subjects that were deemed representative of the target audience, and consisted of having them solve problems using the two techniques. Timing performance and correctness were measured, as well as the subjective ratings of individual users.

6.2.1 Subjects

Twenty users, fifteen of which were male, participated in this study. All users were screened to have good computer skills and at least basic knowledge of distributed systems and general causal relations. Subject ages ranged from 20 through 50 years old, and all had normal or corrected-to-normal vision (one person claimed partial color blindness but was still able to carry out the test). Ten of the subjects had participated in our earlier user study of the Growing Squares technique.

6.2.2 Equipment

We used the same equipment that was used for the Growing Squares user study for this study as well.

6.2.3 Procedure

As before, the experiment was a two-way repeated-measures analysis of variance (ANOVA) for the independent variables "visualization type" (Hasse diagrams versus Growing Polygons) and "data density" (sparse versus dense). The sparse data density consisted of system executions involving 5 processes and 15 messages, while the dense data density involved 20 processes and 60 messages. All subjects were given the same four task sets split into the two density classes. The system trace for each task set was generated using a simple randomized heuristic algorithm to avoid subjects taking advantage of special knowledge about the behavior of a particular distributed system. In addition, care was taken to ensure that the complexity of both system traces for a specific data density was roughly equivalent by removing ambiguities and ensuring that the number of indirect relations was the same.

The procedure consisted of solving two of the four task sets using conventional Hasse diagrams, and the other two using the Growing Polygons technique. Sparse task sets were solved first, followed by the respective dense sets. In order to minimize the impact of learning effects, half of the subjects used the Hasse diagrams first, while the other half used the Growing Polygons first. The task sets themselves consisted of four tasks that were directly based on our previous user study of Growing Squares (see Table 2 for an overview). Subjects were given the opportunity to freely adjust window size and placement prior to starting work on each task set. Furthermore, subjects were instructed to solve each task quickly but thoroughly, and were allowed to ask questions during the course of the procedure. Each individual task in a task set was timed separately, except for the tasks Causality 1-3, which were timed together. In addition, answers were checked and the correctness ratio was recorded for each task.

In order to avoid run-away times on troublesome tasks, completion times were limited to 10 minutes (600 seconds). If a test subject chose for some reason to skip a task, the completion time for that task was set to this cap.

After each completed task set, each subject was given a short questionnaire of three 5-point Likert-scale questions asking for their personal opinion on the usability, efficiency, and enjoyability of the visualization method they had just used (see tasks Q1 to Q3 in Table 2). The purpose of this questionnaire was to measure how users' ratings of the visualizations changed depending on the data density. In addition, users also filled out a post-evaluation questionnaire after having completed all of the task sets, where they were asked to rank the two visualizations on the above criteria (see Table 7).

Each evaluation session lasted approximately 45 minutes. Prior to starting the evaluation itself, subjects were given a training phase of up to ten minutes where they were given instructions on how to use both visualization methods to solve various simple tasks.

Data Density	**Processes**	**Messages**
Sparse	5	15
Dense	20	60

Table 6: Experimental design. Both density and visualization factors were within subjects for all 20 subjects.

Task	**Comments**
PQ1	Rank the visualizations w.r.t. ease of use.
PQ2	Rank the visualizations w.r.t. efficiency for solving the following tasks: (a) Duration analysis (b) Influence importance (most influential) (c) Influence assessment (most influenced) (d) Inter-node causal relations
PQ3	Rank the visualizations w.r.t. enjoyability.

Table 7: Post-evaluation ranking questions.

6.3 Results

The analysis of the results we obtained from the aforementioned user study can be divided into the timing performance, the correctness, and the subjective ratings of the test subjects.

6.3.1 Performance

The mean times of solving a full task set (i.e. all four tasks) using Hasse diagrams and the Growing Polygons visualizations were 433.90 (s.d. 378.59) and 251.85 (s.d. 174.88) seconds respectively. This is also a statistically significant difference ($F(1, 19) = 20.118$, $p < .001$). The main effect for density was significant ($F(1, 19) = 26.932$, $p < .001$), with means for the sparse and dense conditions of 191.80 (s.d. 87.57) and 493.95 (s.d. 359.35) seconds.

Figure 11 summarizes the mean task results for the two visualizations across the two densities; error bars show the standard deviation above and below the mean. The figure also shows that the mean time for the task set was higher for the Hasse method across all densities. For the sparse condition, the mean completion times were 234.40 (s.d. 87.09) and 149.20 (s.d. 65.85) seconds for the Hasse and Growing Polygons visualizations. The Growing Polygons method also gave better results for dense conditions, with mean values of 616.05 (s.d. 550.60) seconds for the Hasse visualization versus 354.50 (s.d. 190.41) seconds for Growing Polygons.

The one exception where Hasse diagrams performed better than Growing Polygons was for the Duration subtask across both densities, with sparse set mean times of 25.75 (s.d. 10.39) for Hasse diagrams versus 33.95 (s.d. 17.47) for Growing Polygons, and for the dense set, 34.40 (s.d. 18.54) versus 72.35 (s.d. 36.06) seconds. This difference was also significant ($F(1,19) = 26.943$, $p < .001$).

For the Influence 1 subtask, on the other hand, the mean completion times for the sparse data set using Hasse diagrams was 58.50 seconds (s.d. 22.25) versus 36.65 seconds (s.d. 17.93) for the Growing Polygons method, while the mean times for the dense set were 270.60 (s.d. 180.XX) and 169.70 (s.d. 140.72), respectively. This was a significant difference ($F(1,19) = 14.614$, $p = 0.001$). Similarly, the Influence 2 subtask yielded mean completion times of 77.64 (s.d. 53.58) versus 34.35 (s.d. 30.47) for the sparse data set, and 184.10 (s.d. 207.05) versus 50.85 (s.d. 26.61) for the dense case. Again, this was a significant difference in favor of the Growing Polygons method ($F(1,19) = 14.170$, $p = 0.001$). Finally, the Causality 1-3 subtask resulted in sparse mean completion times of 72.50 (s.d. 29.28) for Hasse diagrams and 44.25 (s.d. 19.68) for Growing Polygons, whereas the dense means were 144.30 (s.d. 116.37) and 61.60 (s.d. 40.88), respectively. This was also a significant difference ($F(1,19) = 18.896$, $p < 0.001$).

6.3.2 Correctness

The average number of correct answers when solving a full task set (i.e. six tasks) using Hasse diagrams and the Growing Polygons visualization was 4.375 (s.d. 1.148) versus 5.625 (s.d. 0.667) correct answers, respectively. This is a significant difference ($F(1,19) = 46.57$, $p < .001$). For the sparse data set, the mean correctness was 4.70 (s.d. 1.218) for Hasse diagrams and 5.75 (s.d. 0.716) for Growing Polygons, versus 4.05 (s.d. 0.999) and 5.50 (s.d. 0.607) for the dense case. In fact, the mean correctness of the Growing Polygons visualization is significantly better than for Hasse diagrams for all individual subtasks except for the Duration subtask, where Hasse performs better with a correctness ratios of 0.975 versus 0.950 for Growing Polygons. This, however, is not a significant difference ($F(1,19) = 0.322$, $p = .577$).

6.3.3 Subjective Ratings

For the post-task questionnaire, the test subjects consistently rated Growing Polygons above Hasse diagram in all regards, including efficiency, ease-of-use and enjoyment. The mean response values to the five-point Likert-scale questions are summarized in Figure 13. See Table 8 for the complete data analysis table.

The subjects' response to the ease-of-use question (Q1, Table 8) showed a higher rating for the Growing Polygons visualization than Hasse diagrams in both sparse (means 4.20 (s.d. .70) and 2.75 (s.d. .85), respectively) and dense data densities (means 3.75 (s.d. .79) and 1.90 (s.d. .91)). Both higher ratings were significant (Friedman Tests, $p < .001$ for both the sparse and dense cases). The subjects' responses to the efficiency question (Q2, Table 8) showed a higher rating for the Growing Polygons visualization in both sparse (means 4.20 (s.d. .62) and 2.40 (s.d. .88)) and dense data densities (means 3.95 (s.d. .51) and 1.55 (s.d. .51)). Both higher ratings readings were significant (Friedman Tests, $p < .001$ for the sparse case and $p < .001$ for the dense case). The subjects' response to the enjoyment question (Q3, Table 8) also showed a higher rating for the Growing Polygons visualization in both sparse (means 4.20 (s.d. .62) and 2.95 (s.d. .39)), and dense data densities (means 4.10 (s.d. .64) and 2.00 (s.d. .73)). Both higher ratings were significant (Friedman Tests, $p < .001$ for the sparse case and $p < .001$ for the dense case).

The results from the post-task summary questionnaire can been found in Table 9, and clearly show that test subjects regard the Growing Polygons technique as superior to Hasse diagrams in all aspects except for duration analysis (task PQ2 (a)). However, as can be seen from the this table, the overall user rankings are very convincingly in favor of our method.

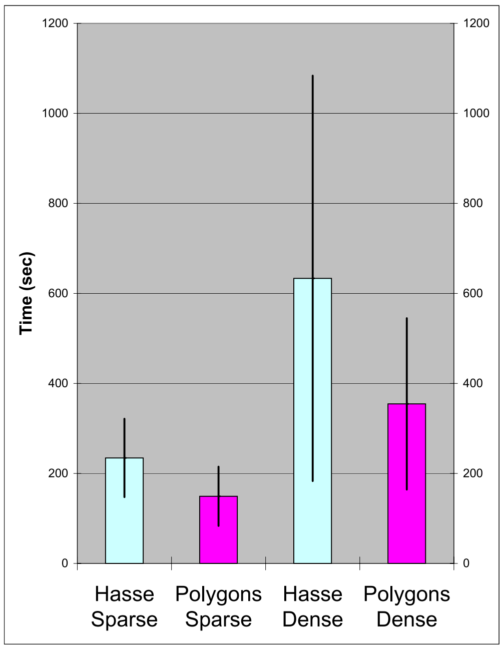


Figure 11: Mean task completion times for all tasks across the Hasse and Growing Polygons methods and across levels of density. Error bars show standard deviations.

7 Discussion

The results obtained from our user studies quite comfortably show that the Growing Squares and the Growing Polygons methods are both superior to Hasse diagrams in terms of performance, correctness, and the subjective opinion of the test subjects across all data densities (although Growing Squares are only significantly more efficient to use for the sparse density). The test subjects consistently ranked both techniques

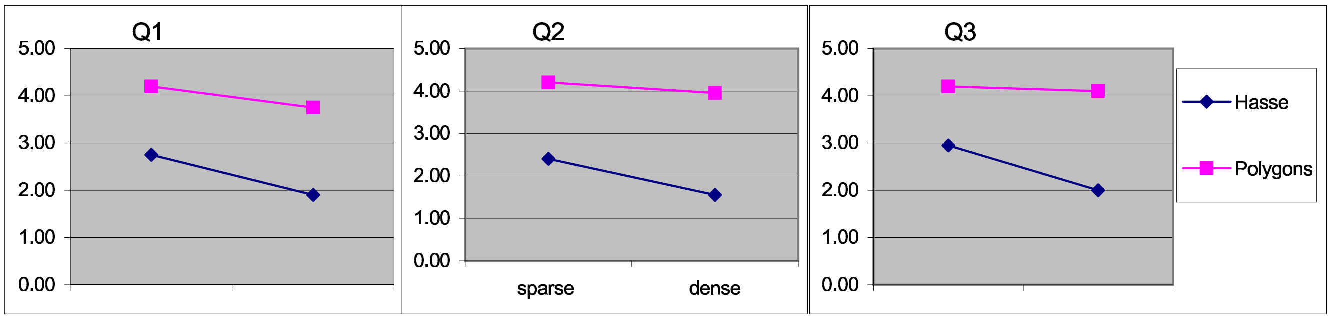


Figure 13: Responses to Q1-Q3 5-point Likert-scale questions across sparse and dense data densities for the Hasse and Growing Polygons methods.

Question	**Hasse diagrams**		**Growing Polygons**		**Reliability**
	sparse	**dense**	**sparse**	**dense**	**Hasse/GP**
Q1. Rate the visualization w.r.t. ease-of-use.	2.75 (.85)	1.90 (.91)	4.20 (.70)	3.75 (.79)	yes
Q2. Rate the visualization w.r.t. efficiency.	2.40 (.88)	1.55 (.51)	4.20 (.62)	3.95 (.51)	yes
Q3. Rate the visualization w.r.t. enjoyability.	2.95 (.39)	2.00 (.73)	4.20 (.62)	4.10 (.64)	yes

Table 8: Mean (standard deviation) responses to 5-point Likert-scale questions. Reliability is defined as being significant at the .05 level.

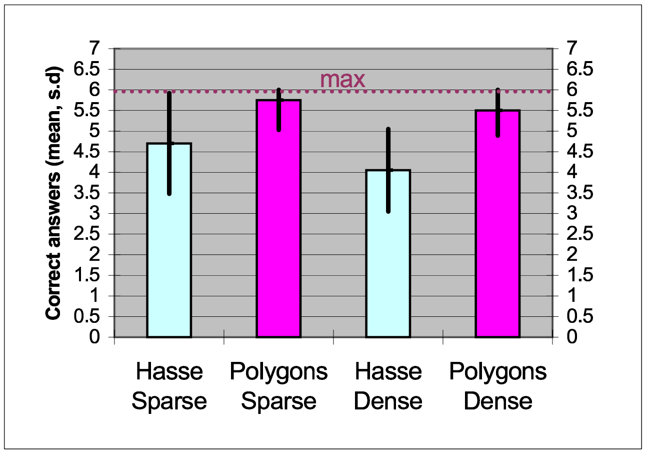


Figure 12: Mean correctness for all tasks across the Hasse and Growing Polygons methods and across levels of density. Error bars show standard deviations.

before Hasse diagrams in all aspects except for measuring process duration. Our findings show that users are significantly more efficient and correct when using Growing Polygons to analyze the influences and check inter-process causal relations in a system (both sparse and dense).

The only subtask where Hasse diagrams perform significantly better is duration analysis, where users were asked to find the most long-lived process in the system. However, while the correctness for this subtask is also better using Hasse diagrams, this is not a significant difference. The fact that Hasse diagrams perform better in this regard is not sur-

Task	**Comment**	**GP**	**Hasse**	**Undec.**
PQ1	Ease-of-use	95 %	0 %	5 %
PQ2	Efficiency (avg)	80 %	11 %	9 %
	(a) Duration	35 %	40 %	25 %
	(b) Importance	90 %	5 %	5 %
	(c) Assessment	95 %	0 %	5 %
	(d) Causality	100 %	0 %	0 %
PQ3	Enjoyability	100 %	0 %	0 %

Table 9: Subject responses to ranking the two visualizations.

prising, given that the visual design of Hasse diagrams allows for easy length comparison of the parallel process lines. This fact is also reflected in the user rankings, where 40 % of the subjects stated that they preferred Hasse diagrams whereas only 35 % preferred Growing Polygons (no similar question was asked in the Growing Squares study).

Our intention with the design of the Growing Squares and Growing Polygons techniques was to provide better alternatives to causality visualization than existing techniques. We used Hasse diagrams as the basis for our comparative user study on the basis that it is still the standard way of visualizing causal relations. However, the question is naturally where the Growing Polygons and Growing Squares techniques stand in relation to each other. While we have not performed a direct comparison between the two techniques, the Growing Polygons method is likely superior to the Growing Squares method. First of all, the Growing Polygons method achieves statistically significant improvement over Hasse diagrams in *all* subtasks (except the duration analysis subtask, which the Growing Squares method also failed at) and across all densities, something which the Growing Squares method does not manage for dense data sets. Second, the comments from the test subjects who also participated in the previous user study clearly indicate that the new method is significantly superior to the older one. Unfortunately, the nature of the work we conducted means that we cannot compare the two techniques directly.

We have already discussed how the the human eye's limited capabilities of color distinction hampered the scalability of the original Growing Squares method. Color is similarly used to encode processes in the Growing Polygons method, but here processes are also assigned a unique sector in the process polygons, so this issue should be less of a concern. However, we have not yet performed any stress tests with very high numbers of involved processes to explore the boundaries of the hybrid approach that the Growing Polygons uses.

Scalability is a relative measure, and even if the results from the Growing Polygons study are favorable, it is clear that displaying every single involved process in a system will not be feasible in the extreme cases. For very large systems of causal relations, some kind of hierarchical clustering scheme needs to be used to group sets of processes into process groups, preferably in a dynamical and self-adjusting way. In addition, executions spanning a long period of time probably require a non-linear time scale to allow for efficient visualization.

In our user studies, all test subjects were well-familiar with Hasse diagrams prior to carrying out the experiments whereas they knew nothing of the new visualizations in beforehand, yet performed consistently better using the new techniques in almost all cases. This, we think, suggests that the Growing Squares and Growing Polygons methods are intuitive and easily accessible, and that the methods with practice might become even more efficient to use. The subjective ratings also support this belief.

Finally, the positive feedback that we have received from the subjects suggests that these kinds of alternate visualization methods of causal relations are indeed useful and worthwhile avenues for future research. By combining them with traditional methods such as Hasse diagrams, users will be able to use the strengths of different methods to solve different problems. In addition, the ability to view systems of causal relations from different perspectives will greatly aid in understanding the mechanics of such a system.

8 Conclusions and Future Work

We have presented two visualization techniques for the graphical representation of causal relations in systems of interacting processes. The methods, called Growing Squares and Growing Polygons, respectively, both abandon the linear timeline of conventional methods such as Hasse diagrams, and instead visualize the execution using color, texture, and animation. The Growing Squares technique, on the one hand, represents processes as color-coded squares that grow in size as time progresses. Messages between processes carry across the source color to the destination, thus showing the casual influences of each process. The Growing Polygons technique, on the other hand, uses n-sided polygons partitioned into triangular sectors to represent processes, analogously allowing them to grow from zero to full size over time. Each sector is assigned to a specific process and given a unique color, and is filled in for each process polygon that receives an influence from the process it represents. We have performed comparative user studies of both techniques in relation to traditional Hasse diagrams, and our results give conclusive evidence that our methods not only are more efficient and give better correctness, but that test subjects also tend to prefer our methods over Hasse diagrams.

As mentioned earlier, while the Growing Polygons technique seems to perform well for small and medium-sized system executions, we have yet to perform any form of stress testing for very large systems (upwards of hundreds or even thousands of processes potentially spanning a very long period of time). In the future, we will explore hierarchical clustering techniques as well as "time windows" and non-linear time scales for adressing these concerns.

Acknowledgements

The authors would like to thank the employees and students of Chalmers University of Technology who participated in the user study. Thanks to the members of the DCS group for their help on reviewing and commenting on the paper as well as participating in the panel discussion at the beginning of the project. Thanks to David Modjeska for reviewing and commenting on the Growing Squares paper in its early stages. Thanks to the anonomyous reviewers, whose comments helped us greatly improve the paper. This work was partially supported by the Swedish Research Council (VR).

References

Aristotle. 350 B.C. *Physics: Book II.* Translated by Richard Hooker (1993).

Bederson, B. B., Hollan, J. D., Perlin, K., Meyer, J., Bacon, D., and Furnas, G. W. 1996. Pad++: A zoomable graphical sketchpad for exploring alternate interface physics. *Journal of Visual Languages and Computing 7*, 3–31.

Bederson, B. B., Meyer, J., and Good, L. 2000. Jazz: An extensible zoomable user interface graphics toolkit in Java. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2000)*, 171–180.

Bemmerl, T., and Braum, P. 1993. Visualization of message passing parallel programs with the TOPSYS parallel programming environment. *Journal of Parallel and Distributed Computing 18*, 2 (June), 118–128.

Cormen, T. H., Lesierson, C. E., Rivest, R. L., and Stein, C. 2001. *Introduction to Algorithms*, second ed. MIT Press.

Elmqvist, N., and Tsigas, P. 2003. Causality visualization using animated growing polygons. In *Proceedings of the IEEE Symposium on Information Visualization 2003*, 189–196.

Elmqvist, N., and Tsigas, P. 2003. Growing squares: Animated visualization of causal relations. In *Proceedings of the ACM Symposium on Software Visualization 2003*, 17–26.

Heath, M. T., and Etheridge, J. A. 1991. Visualizing the performance of parallel programs. *IEEE Software 8*, 5 (Sept.), 29–39.

Heath, M. T. 1990. Visual animation of parallel algorithms for matrix computations. In *Proceedings of the Fifth Distributed Memory Computing Conference*, 1213–1222.

Koldehofe, B., Papatriantafilou, M., and Tsigas, P. 1999. Distributed algorithms visualisation for educational purposes. In *Proceedings of the 4th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education*, 103–106.

Kraemer, E., and Stasko, J. T. 1998. Creating an accurate portrayal of concurrent executions. *IEEE Concurrency 6*, 1 (Jan./Mar.), 36–46.

Lamport, L. 1978. Time, clocks and the ordering of events in distributed systems. *Communications of the ACM 21*, 7, 558–564.

Levkowitz, H., and Herman, G. T. 1992. Color scales for image data. *IEEE Computer Graphics and Applications 12*, 1 (Jan.), 72–80.

Levkowitz, H., Holub, R. A., Meyer, G. W., and Robertson, P. K. 1992. Color versus black and white in visualization. *IEEE Computer Graphics and Applications 12*, 4 (July), 20–22.

Moses, Y., Polunsky, Z., and Tal, A. 1998. Algorithm visualization for distributed environments. In *Proceedings of the IEEE Symposium on Information Visualization 1998*, IEEE, 71–78.

Perlin, K., and Fox, D. 1993. Pad: An alternative approach to the computer interface. In *Proceedings of Computer Graphics (SIGGRAPH 93)*, vol. 27, 57–64.

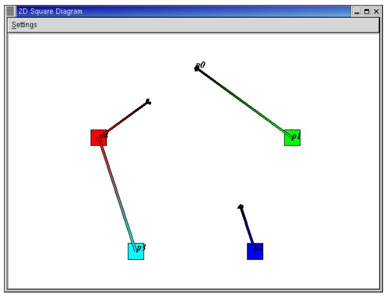
Socha, D., Bailey, M. L., and Notkin, D. 1989. Voyeur: Graphical views of parallel programs. In *Proceedings of the ACM SIGPLAN/SIGOPS Workshop on Parallel and Distributed Debugging, ACM SIGPLAN Notices 24*, 206–215.

Stasko, J. T., and Kraemer, E. 1993. A methodology for building application-specific visualizations of parallel programs. *Journal of Parallel and Distributed Computing 18*, 2 (June), 258–264.

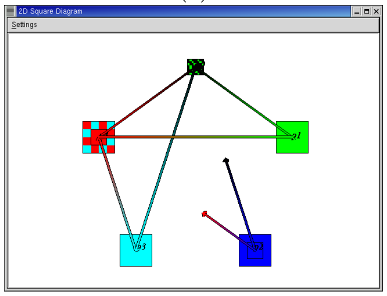
Topol, B., Stasko, J. T., and Sunderam, V. 1998. PVaniM: a tool for visualization in network computing environments. *Concurrency: Practice and Experience 10*, 14 (Dec.), 1197–1222.

Ware, C., Neufeld, E., and Bartram, L. 1999. Visualizing causal relations. In *Proceedings of the IEEE Symposium on Information Visualization 1999 (Late Breaking Hot Topics)*, 39–42.

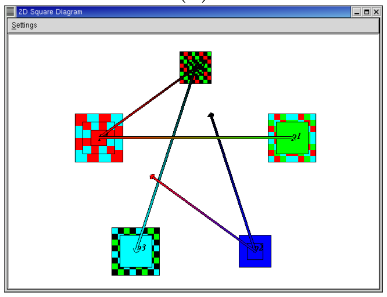
Wyszecki, G., and Stiles, W. S. 1991. *Color Science: Concepts and Methods, Quantitative Data and Formulae*, second ed. John Wiley & Sons.



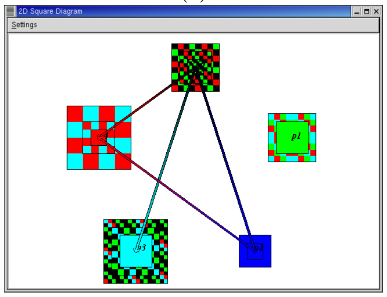
(a)



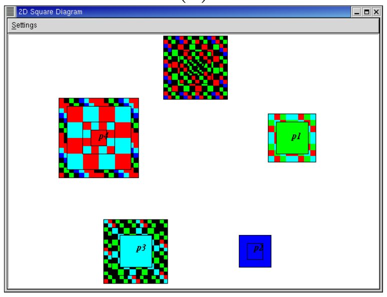
(b)



(c)

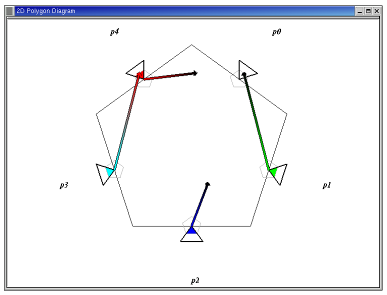


(d)

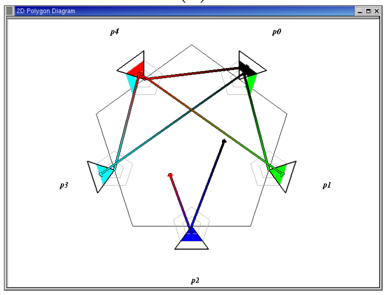


(e)

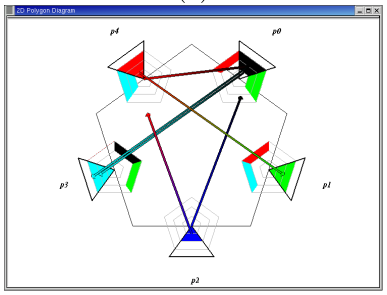
Figure 14: Growing Squares visualization of the dynamic execution of a 5-process distributed system.



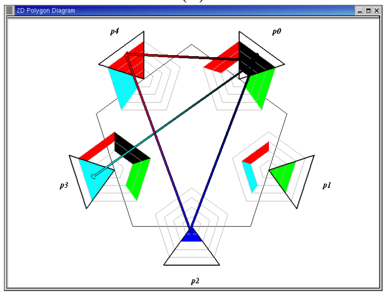
(a)



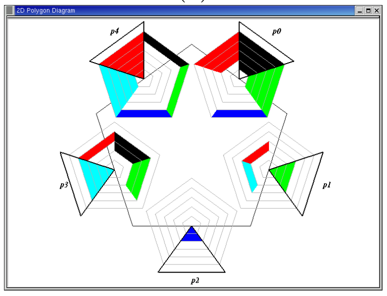
(b)



(c)



(d)



(e)

Figure 15: Growing Polygons visualization of the dynamic execution of a 5-process distributed system.