Causality Visualization Using Animated Growing Polygons*

Niklas Elmqvist† Philippas Tsigas‡

Department of Computing Science
Chalmers University of Technology
SE-412 96 Göteborg, Sweden

Abstract

We present Growing Polygons, a novel visualization technique for the graphical representation of causal relations and information flow in a system of interacting processes. Using this method, individual processes are displayed as partitioned polygons with color-coded segments showing dependencies to other processes. The entire visualization is also animated to communicate the dynamic execution of the system to the user. The results from a comparative user study of the method show that the Growing Polygons technique is significantly more efficient than the traditional Hasse diagram visualization for analysis tasks related to deducing information flow in a system for both small and large executions. Furthermore, our findings indicate that the correctness when solving causality tasks is significantly improved using our method. In addition, the subjective ratings of the users rank the method as superior in all regards, including usability, efficiency, and enjoyability.

CR Categories: D.1.3 [Programming Techniques]: Concurrent Programming; D.2.5 [Software Engineering]: Testing and Debugging; H.5.1 [Information Systems]: Multimedia Information Systems—Animations; H.5.2 [Information Systems]: User Interfaces; I.3 [Computer Methodologies]: Computer Graphics

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.

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. A less traditional visualization technique is Growing Squares [Elmqvist and Tsigas 2003], which uses colors, textures, and animation to provide a graphical representation of the information flow in the system. This method addresses the major issues associated with Hasse diagrams, but instead suffers from a number of weak points; the original Growing Squares technique uses a simple color coding scheme for processes that does not scale well with system size, and the relative timing of process events is not readily visible using this visualization.

In this paper, we present a new visualization technique called Growing Polygons that attacks the problem of effective causality visualization with the use of animation, colors, and patterns to provide an accessible overview of a system. The basic idea is to assign each node in a system of n processes a color and 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 Growing Squares and the visualization is therefore more scalable. Internal markings on the process polygons serve as "age rings" which allow the user to assess the relative ordering of events. This visualization technique has been implemented as part of our application

*This work was partially supported by the Swedish Research Council (VR).

†e-mail: elm@cs.chalmers.se

‡e-mail: tsigas@cs.chalmers.se

platform for causality visualization, allowing us to compare Growing Polygons with existing methods.

The design of the Growing Polygons method was largely guided by the information we collected using a focus group of distributed systems researchers at the onset of the project. The new method builds on the philosophy of our previous work, but attempts to solve the specific problems associated with the Growing Squares technique. Identifying these problems was mainly done through the comments and observations of the subjects during our previous user study. We conducted a new user study to evaluate whether or not the Growing Polygons method is an improvement over the existing visualizations, and our results are very positive: the new 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). Subjects also have a much higher correctness ratio 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 original Growing Squares method, is perceived as more efficient as well as easier and more enjoyable to use than Hasse diagrams.

As discussed above, 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.

The structure of this paper is as follows: We first describe the existing work in the field, followed by a brief formal definition of causal relations. Then, we describe the Growing Polygons method, including details on design and implementation, and present the user study we conducted. 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 causal relation visualization, and the prevalent visualization method is still the traditional Hasse (also know as time-space) diagram. In this diagram, we assign the time parameter to one of the coordinate axes (usually the x axis) and distribute processes along the other axis. Horizontal lines in the diagram are used to denote the life time of individual processes, events (both internal and external) are marked using dots on the line, and arrows between them represent messages sent and received by the processes involved. This straightforward method is currently found in many platforms for visualization and debugging, especially those aimed at parallel and distributed systems. In fact, too many such platforms exist for us to mention them all; examples include [Socha et al. 1989; Bemmerl and Braum 1993; Moses et al. 1998; Koldehofe et al. 1999; Kraemer and Stasko 1998; Heath 1990; Heath and Etheridge 1991; Topol et al. 1998; 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.

Of more specific interest is our previous work on the Growing Squares [Elmqvist and Tsigas 2003] visualization technique that uses a metaphor of growing 2D squares to represent processes in the system. The interior of each square is color-coded to signify the influences it has received from other process squares, and the whole visualization is animated to show the dynamic execution of the system over time. The purpose of Growing Squares is to provide an easily accessible overview of the information flow in the system, including the causal dependencies of each process. By studying the colors in the patterned square of each process, users can at any time deduce the influences the process has received from other processes. This visualization has been shown to be significantly more efficient than Hasse diagrams for small systems, but regrettably not so for larger systems. However, the subjective ratings of users clearly name Growing Squares as superior to Hasse diagrams in every regard, and indicate that the method addresses most of the weaknesses of traditional visualizations.

As indicated by our earlier user study, the Growing Squares technique has a number of issues of its own. 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.

Finally, the *visual causality vector* (VCV) presented by Ware *et al.* [1999] is a motion-based visual and perceptual construct for representing causal relations in directed acyclic graphs. However, Ware's primary contribution is the investigation of timing concerns for the perception of causality for users, not the visualization technique *per se*.

3 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 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 in 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.

4 Growing Polygons

Visualizing the causal relations in a system consisting of n processes using the Growing Polygons 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.

[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.

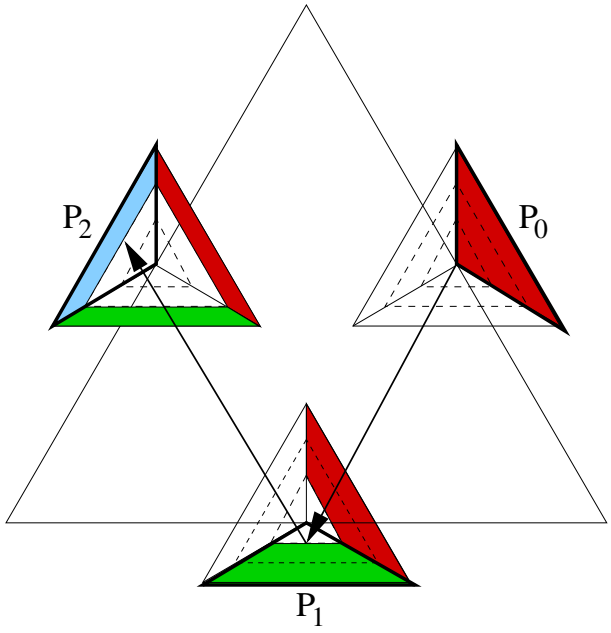

Figure 1: Growing Polygons visualization with $n = 3$ (i.e. the process polygons are triangles).

Figure 1 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.

As we ascertained earlier, causal relations are transitive, so if $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$. Figure 1 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 1 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 4 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.

4.1 Analysis Tasks

The analysis and design of the Growing Polygons technique was largely guided by input gained from discussions we conducted with a focus group of researchers of distributed systems working at our department. 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.

4.1.1 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.

4.1.2 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.

4.1.3 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.

4.2 Design Decisions

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. Thus, a continuous color spectrum such as LOCS [Levkowitz and Herman 1992] is not suitable. Instead, 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 ($\theta = 360^\circ/n$) 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 dependencies in the system. Animation allows the user to still see the dynamic execution of the system in an intuitive way.

4.3 Implementation

We implemented the Growing Polygons method as a separate visualization class in CausalViz, our existing visualization platform for general causal relations. CausalViz is written in C++ on the Linux platform and features a modularized system architecture that facilitates easy extension with new visualization classes. The application uses a general XML file format for partially ordered sets, and can then present multiple visualizations of the execution in separate windows. This provided us with a simple way of comparing

our new method with existing visualizations, such as Hasse diagrams and Growing Squares.

Our implementation of Growing Polygons uses OpenGL for rendering as well as the existing continuous zoom mechanism built into CausalViz. However, to avoid users getting lost inside the visualization (we observed this numerous times when evaluating the Growing Squares method), we constrained the 2D camera to the interior of the layout polygon. In addition, we provide the users with a small navigation window acting both as a key for mapping colors and process sectors to process names, and as a shortcut for quickly jumping to a specific process.

5 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.

5.1 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.

5.2 Equipment

The study was run on a high-end 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.

5.3 Procedure

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 1 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.

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 unpleasant, 5=very pleasant).	Likert

Table 1: Repeated tasks for each density and visualization type.

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 1). 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 2).

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.

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 2: Post-evaluation ranking questions.

6 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.

6.1 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 2 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 ($F(1, 19) = 26.943$, $p < .001$).

6.2 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 ($F(1, 19) = 0.322$, $p = .577$).

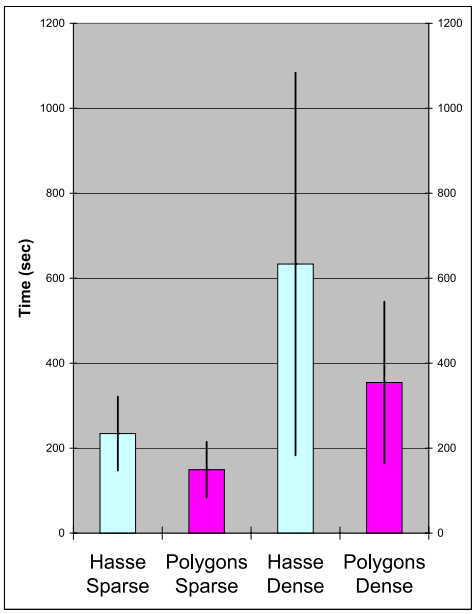

Figure 2: Mean task completion times for all tasks across the Hasse and Growing Polygons methods and across levels of density. Error bars show standard deviations.

6.3 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. See Table 3 for the complete data analysis table.

The subjects' response to the ease-of-use question (Q1, Table 3) 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 3) 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 3) 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 4, 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.

Question	Hasse diagrams sparse	Hasse diagrams dense	Growing Polygons sparse	Growing Polygons dense	Reliability 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 3: Mean (standard deviation) responses to 5-point Likert-scale questions. Reliability is defined as being significant at the .05 level.

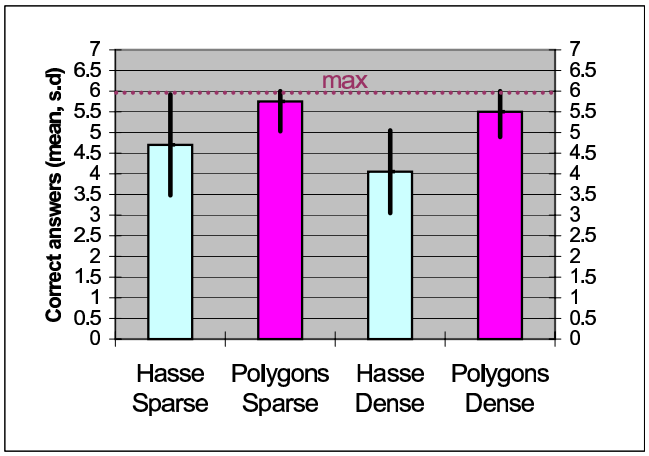

Figure 3: Mean correctness for all tasks across the Hasse and Growing Polygons methods and across levels of density. Error bars show standard deviations.

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 4: Subject responses to ranking the two visualizations.

7 Discussion

The results obtained from our user study quite comfortably show that the Growing Polygons method is superior to Hasse diagrams in terms of performance, correctness, and the subjective opinion of the test subjects across all data densities. The test subjects consistently ranked our technique before Hasse diagrams in all aspects except one. 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 surprising, 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.

Our intention with the design of the Growing Polygons technique was to provide a better alternative 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, our previous work on the Growing Squares technique already improved on Hasse diagrams, so the question is naturally where the Growing Polygons technique stands in relation to Growing Squares. While we have not performed a direct comparison between the two techniques, it is clear that our new technique is superior to the old one. First of all, we have achieved 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 did not manage for the dense data sets. Second, the comments from the test subjects who also participated in the previous user study clearly indicate that Growing Polygons is vastly superior to the Growing Squares method.

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.

8 Conclusions

We have presented a new visualization technique for the graphical representation of causal relations in systems of interacting processes. The method, called Growing Polygons, has abandoned the linear timeline of conventional methods such as Hasse diagrams, and instead represents processes as n-sided polygons partitioned into triangular sectors that 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 a comparative user study of Growing Polygons in relation to traditional Hasse diagrams, and our results give conclusive evidence that our method not only is more efficient and gives better correctness, but that test subjects also tend to prefer our method over Hasse diagrams.

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.

References

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

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.

ELMQVIST, N., AND TSIGAS, P. 2003. Growing squares: Animated visualization of causal relations. In *Proceedings of the ACM Symposium on Software Visualization 2003.*

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.

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.

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 IEEE Information Visualization 99.*

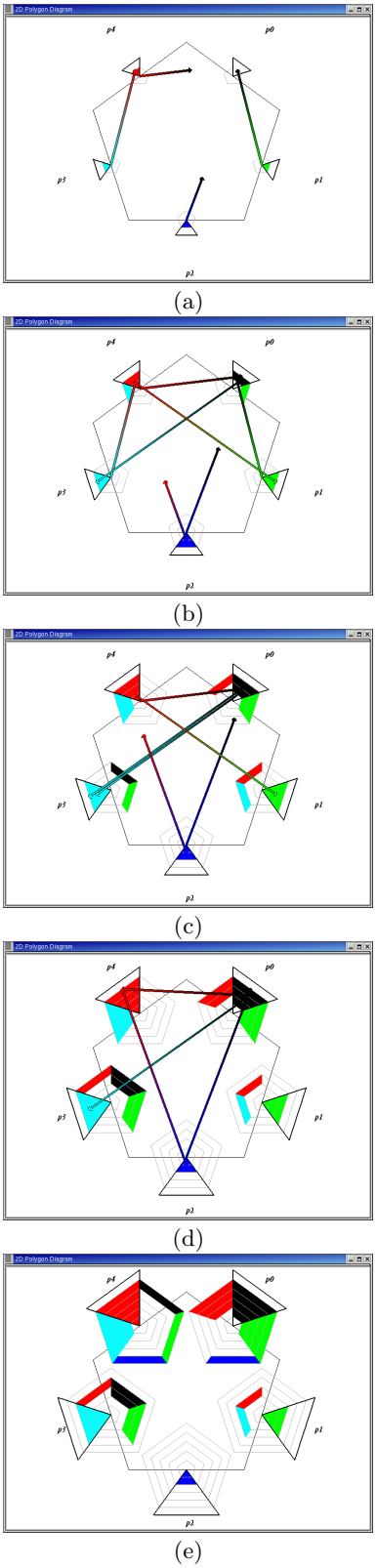

Figure 4: Growing Polygons visualization of the dynamic execution of a 5-process distributed system.