Pacers: Time-Elastic Objects

Steven H. Tang
Stanford University

Mark A. Linton
Silicon Graphics

Abstract

Current time-based presentation systems are rigid in that they assume the running time of all components of a presentation is constant. Furthermore, most systems offer little or no support for dynamically adapting the presentation quality to the (lack of) available system resources. In this paper, we introduce an object called a Pacer that is "time-elastic" in that it can adjust the quality of its presentation according to the amount of time available. We have implemented a direct manipulation graphical interface using pacers that can automatically degrade presentation quality in a controlled fashion depending on the user's input rate and the speed of the rendering system.

Keywords: Elastic objects, adaptive presentation, graphical user interfaces.

1 Introduction

The emergence of video and audio as desirable components of a user interface along with the wide variety of computer platforms increases the need for an adaptive presentation mechanism. Many systems have low-level adaptive algorithms. QuickTime[1] presentations, for example, give higher priority to audio than to video. As a result, video frames are dropped when the playback platform cannot support playing both the audio and video at full speed.

The adaptive strategy in QuickTime is built-in; applications have no control over how the presentation is degraded. One cannot specify, for example, that some portions of a video are less important than others, or that the audio portion can extend by fading out more slowly when the video cannot keep up.

Another motivation for adaptive presentation comes from the example of direct manipulation in a drawing editor. During a manipulation such as moving an object, many editors will draw a bounding box instead of the target object. In some cases, the graphics for the object could be rendered fast enough to present an accurate picture on-the-fly, but application developers cannot predict which objects will be fast enough on which platforms.

The result of a fixed approach to presentation is that meaningful information is often lost while useless information may be retained in the presentation. Since applications cannot accurately predict degradation behavior, they tend to underutilize hardware resources in order to guarantee acceptable performance.

Our goal is to define presentations in terms of the highest quality possible and let the system determine dynamically how to degrade the presentation to give acceptable performance in the absence of sufficient resources. To support this adaptive approach, we define a type of object called a Pacer for time-based composition.We call pacers "time-elastic" because they monitor and adjust the quality of a presentation depending on the time available. Pacers rely on an object-oriented composition model for controlling degradation behavior. Granularity of pacers is a policy issue; for example, one could have either a pacer per video frame or one per group of video frames.

Using a composition model significantly simplifies the design and implementation of adaptive presentations. We have used pacers to implement manipulation of graphical objects in a simple drawing editor called EDraw. The response of EDraw is independent of the complexity of the graphics and the running environment. The more the resources available, the more faithful the manipulation will be.

2 Background

Interactive applications are usually tuned for the least powerful platform on which they will run, resulting in the underutilization of resources on more powerful platforms. In addition, tuning during development does not work in a multi-user, networked environment where available resources can change dynamically.

Systems traditionally consider resource inadequacy using one of the following two approaches:

1) Avoid the situation where resources are insufficient.

2) Recover when resources are insufficient.

The two common strategies for avoidance are to adjust the presentation quality dynamically according to resource availability or to reserve the needed resources ahead of time. The most common technique for recovery is to discard out-of-date information.

2.1 Avoidance

Funkhouser and Sequin[2] have a scheme to adjust the presentation quality of complex virtual environments in order to maintain a constant frame rate. A special cost-benefit metric is used to decide how objects are rendered, and the final presentation attempts to maximize this metric. They use a mathematical model to simulate the graphics hardware and predict the cost of each graphical object, and they use heuristics to compute the benefit of an object. For example, the closer an object is to the eye the more benefit it contributes to the virtual environment.

This approach does not provide a general solution for handling resource inadequacy. In particular, the model is not extensible in that a new object with its own degradation behavior cannot be introduced into the system. In addition, the cost of rendering an object often cannot be predicted easily and the benefit an object contributes may depend more on semantics rather than heuristics. For example, players of a basketball game in a virtual environment may be more important than spectators.

The CM player[3] is a continuous media player that plays synchronized digital video and audio data streams read from a file server. CM scripts use the logical time system (LTS) from Anderson and Homsy[4] to synchronize media streams. The CM architecture includes a CM server which makes UDP connections to CM source processes running on the file server. In a typical scenario, the CM player sends commands to the CM server which then retrieves media data streams from CM sources. To handle degradation, the server applies an adaptive feedback algorithm to perform rate matching. Instead of discarding out of date information, the server sends penalties back to the video CM source periodically. The penalties are a gauge to the number of discarded video frames. The video CM source uses this information to readjust the frame rate. Though relatively low-level, this technique can significantly reduce frame rate variability. However, this approach does not apply generally to a presentation involving multiple media objects. Applications cannot specify various trade-offs between media streams.

SRP[5] is a network management protocol that allows client applications to reserve the needed resources on the network a priori in order to obtain performance guarantees. SRP defines two classes of sessions: guaranteed and best-effort. For guaranteed sessions, resource reservations are made and maintained throughout the sessions. For best-effort sessions, no reservation is made, and workload parameters are only hints to the system. Shortcomings of reservation schemes include:

- Clients must be able to specify their resource requirement, which would be difficult for interactive graphics applications.
- Guaranteed sessions have no degradation, i.e. reservation either succeeds or fails. If clients want to relax the workload parameters they have to start a new session.
- Best effort sessions suffer from the same drawbacks as those using resource inadequacy recovery strategies.
- It is susceptible to deadlocks and starvations, problems faced by operating systems. Handling these problems introduces more overhead into the system.
- It could lead to extremely inefficient use of resources when a small fraction of the total allocated resources is used on average

2.2 Recovery

Audio devices typically have low I/O requirements. The bottleneck, therefore, is often caused by the video devices. QuickTime[1] solves this problem by employing a skipping and pausing algorithm at the operating system level, freeing all QuickTime-based applications of implementing their own rate matching and synchronization schemes. When the presentation is resource inadequate however, it exhibits unpredictable real-time behavior since the movie author has no control over what spatial information gets discarded. QuickTime presentations are also inflexible because their real running times are always constant. Video frames, for example, cannot be preserved by extending the audio track.

2.3 Other systems

Some multimedia synchronization models support coarse-grained adjustments of the presentation. Steinmetz[7] proposed synchronizing objects for the execution of multimedia presentations. At every synchronization point, objects send events to maintain certain synchronization properties of the presentation. Typical degradation behavior is *skipping and pausing*. Blakowski et al.[8] take this model one step further by defining three types of actions that are sent between the objects at each point. They are *Waiting* actions, *Acceleration* actions and *Skipping* actions. These actions tend to improve the awareness of the objects with their running environment and thus enable them to degrade more smoothly. Because of

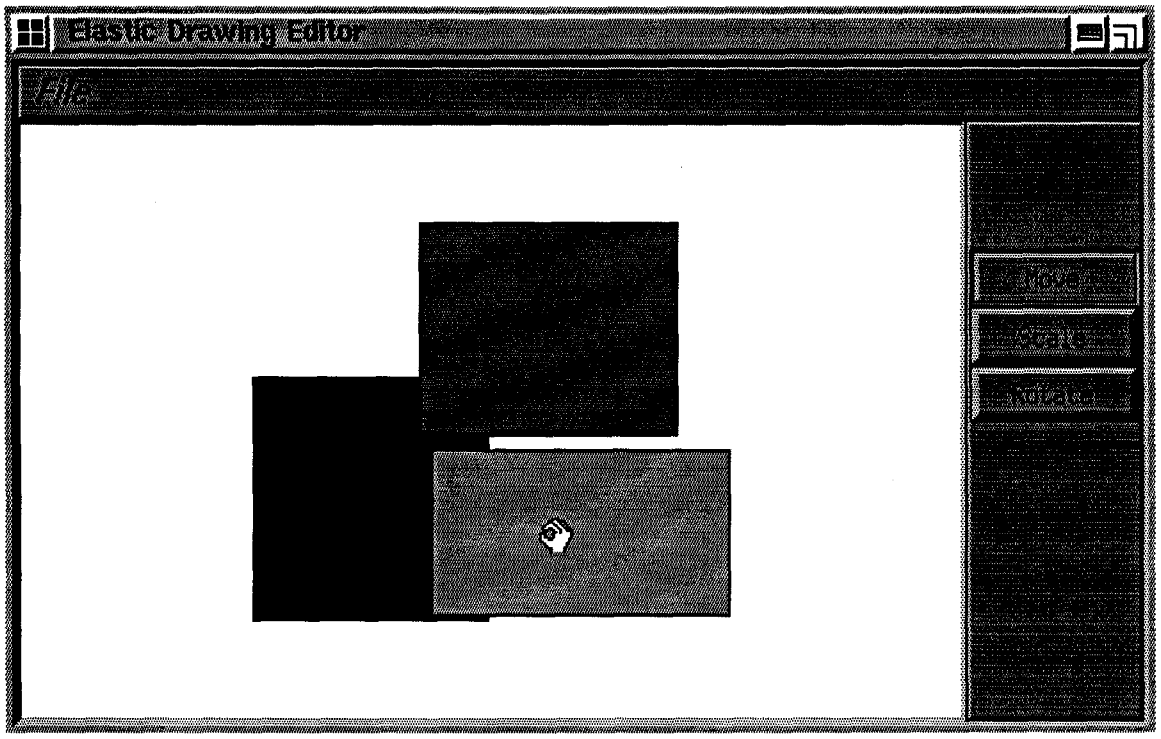


Figure 1: Simple manipulation

the lack of knowledge of time duration, media objects in these systems tend to degrade abruptly. In addition, only a few objects are involved per media stream, limiting the system to coarse-grained degradation behavior.

Hamakawa et al.[9] describe a time authoring model based on boxes and glue in the time domain. The execution objects for the presentation are inelastic. Guimaraes et al.[10] also describe an authoring model based on series and parallel composition, with an execution model that provides several types of actions for synchronization. Gibbs[11] uses a timeline model for authoring. The model has execution objects with synchronization commands similar to Hamakawa[9]. None of these systems addresses the general problem of media degradation.

2.4 Open problems

Though previous systems offer support for adaptive presentation, none provides a mechanism by which application-level objects can control how, when and where degradation is performed. One of our goals was to define a model where objects are actively involved in selecting the appropriate quality of presentation that satisfies the frame time. Unlike the scheme used by Funkhouser and Sequin, we want to be able to change the frame rate dynamically to maximize both response and quality of a presentation. The model should not stipulate the kinds of resources available from the running environment or the kinds of resources required by the application. Rather, objects should be able to adapt to the environment.

3 Elastic drawing editor

Our experimental application is a drawing editor called EDraw, which attempts to present the manipulated state of the original graphics as faithfully as possible while also maintaining reasonable response. The amount of degradation depends on the complexity of the graphics and the rate of user input. In Figure1, three rectangles are being manipulated. Since these graphics are simple, they are presented without any degradation Figure 2 shows a scene with two dancing ducks and a popping wine bottle consisting of about 80 splines and ellipses. When the wine bottle is moved, it may be drawn with a semi-transparent color to keep up with user input. When the manipulation slows down, all the damaged regions are repaired, as shown in Figure 3.

Figure 4 shows the drawing of a car made up with about 100 splines, ellipses and polygons. When the wheel is moved, it degrades to become semi-transparent like before. However, because of the complexity of the wheel itself, when it is moved faster, even the wheel itself degrades such that only some of its components get drawn. If the manipulation

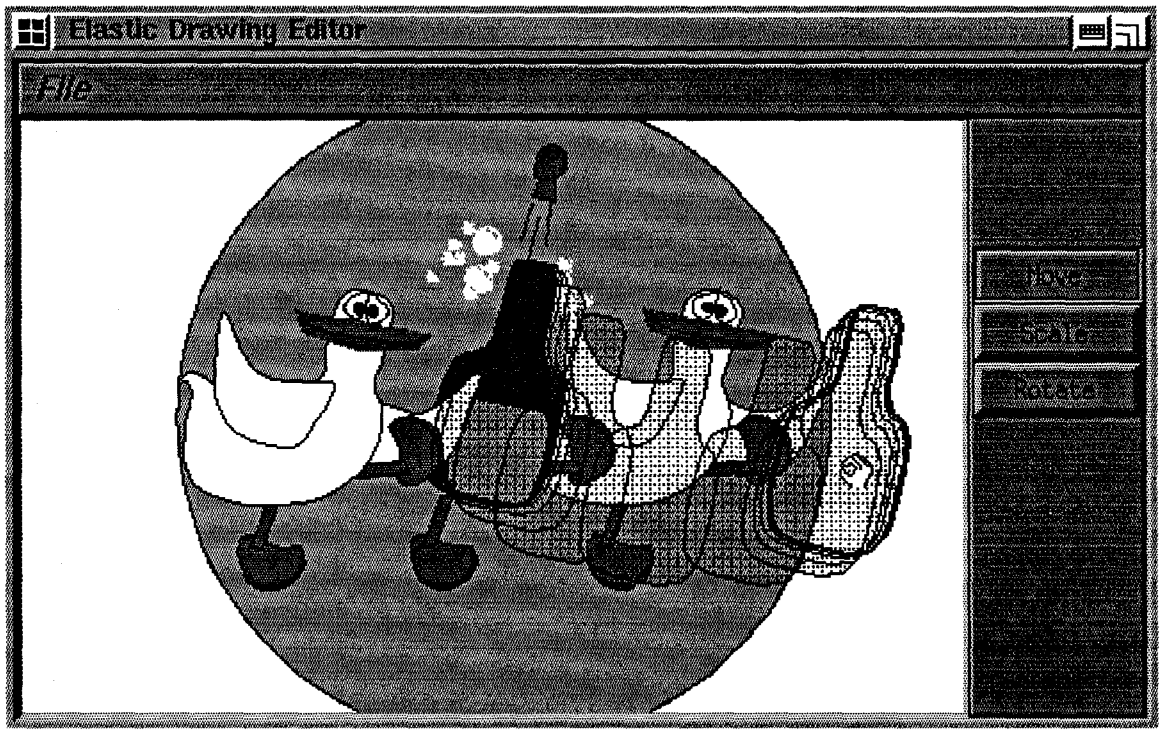


Figure 2: EDraw with ducks

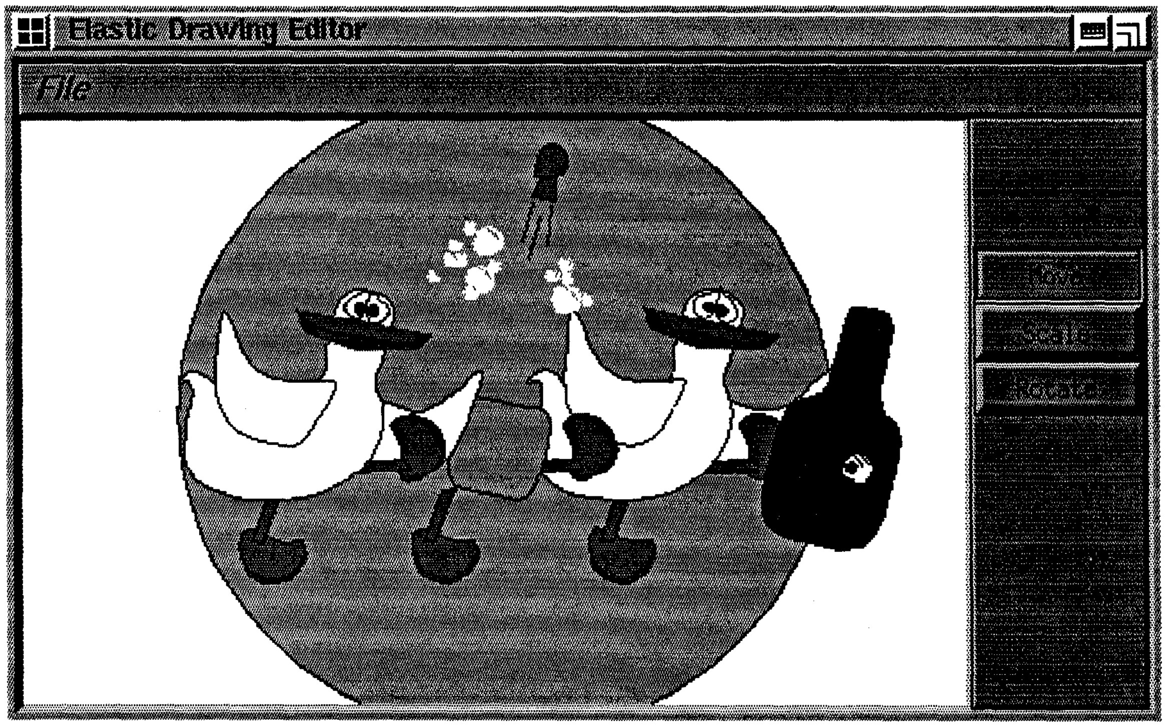


Figure 3: Ducks get fully repaired when manipulation slows down

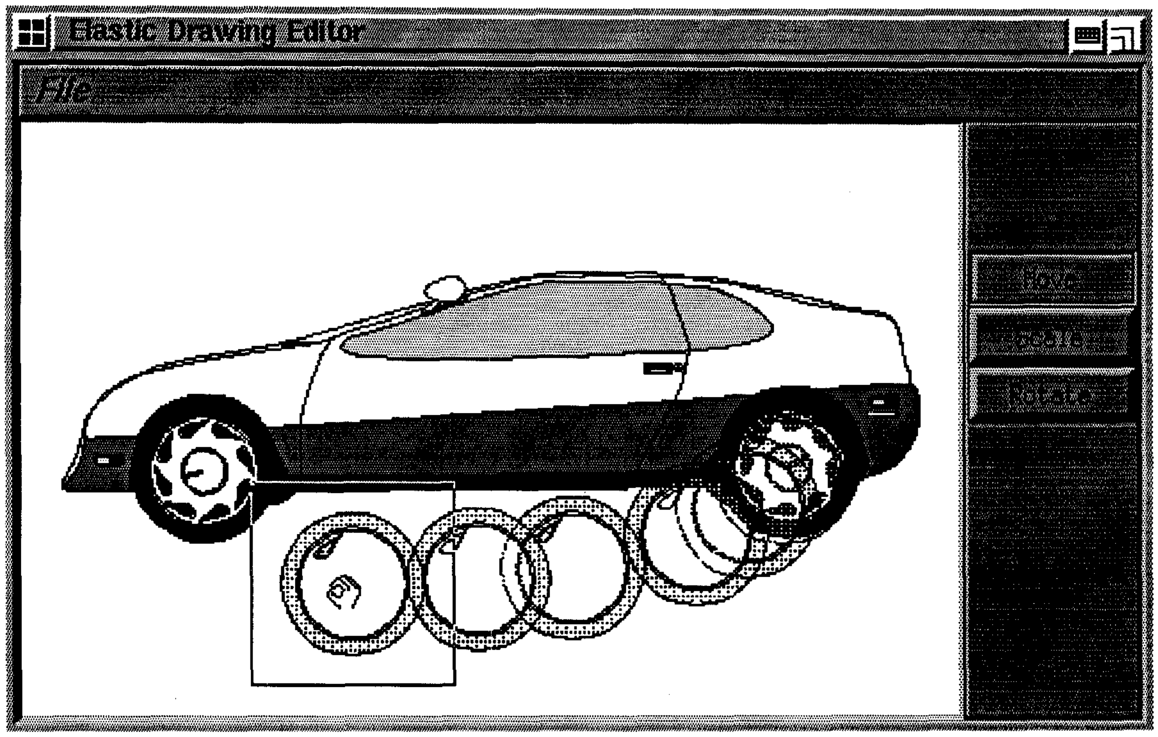


Figure 4: EDraw with a car

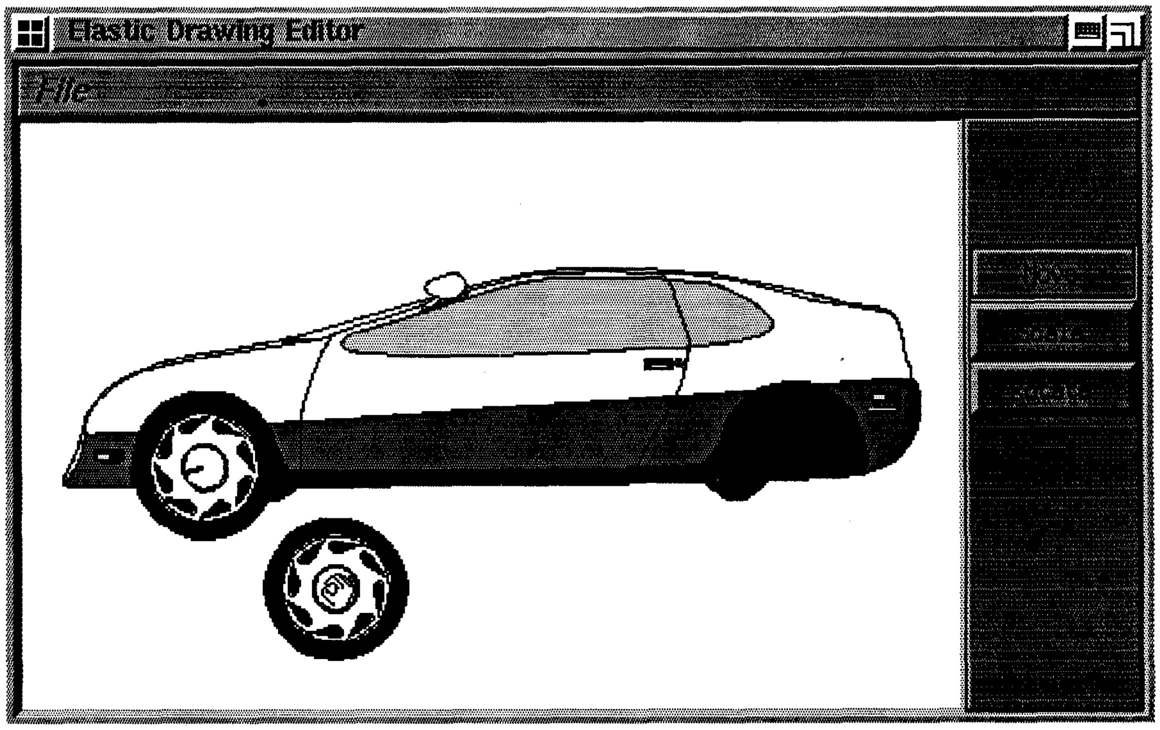


Figure 5: The car gets fully repaired when manipulation slows down

gets even faster, the wheel degrades into a sliding xor rectangle. When the user slows down, the wheel upgrades incrementally until a full repair occurs, as shown in Figure 5. Though simple, these examples demonstrate the important aspects of our architecture.

4 Architecture

The pacer architecture attempts to solve three inherent problems in elastic presentations:

- Unpredictable grade levels
- Oscillation
- Abrupt transitions

When a presentation cannot sustain at a certain quality level, it can degrade to the next lower level. However, it is more difficult to know when to upgrade. If an upgrade attempt is done periodically, the presentation quality is prone to oscillation. The pacer architecture therefore provides a mechanism to adopt different grade adjustment policies. The user can then select the most appropriate one for a specific presentation.

The pacer architecture must also notify objects during grade transitions, giving objects an opportunity to perform the appropriate transitions effects. For example, newly performing graphical objects may prefer to fade-in. Transitions must be handled carefully since they may be more expensive than the normal presentation.

The fundamental object classes in our architecture are Pacer, PaceMaker and PaceKeeper. A pacer performs in sampled increments of time at a certain grade level. Pacer composites define how grade boundaries of their children are assembled. A pace maker defines the clock for an application. A pace keeper defines specific upgrade and downgrade policies for a presentation.

4.1 Pacer

Figure 6 shows the pacer C++ class interface. Pacer::play executes the presentation given the runtime information, which encapsulates current logical time, duration and the old and new grade levels. Logical time is a measure of time relative to the beginning of a presentation. If the old and new grade levels are different, a grade transition is in progress and interested pacers have the option of responding appropriately. In EDraw, this grade change is used by the xor rectangle to clean up the previously drawn rectangle.

Pacers define discrete sampling of performance. In order for a presentation to be continuous, a pacer subclass called Scheduler is introduced that defines periodicity. For example, a video scheduler might define its period to be

```
class Pacer {
public:
    virtual Pacer* get_parent();
    virtual void set_parent(Pacer*);

    virtual void play(RunTime&);
    virtual void pick(SchedulerList&);

    virtual void get_grades(Grade& low,Grade& high);
    virtual void regrade();
};
```

Figure 6 : Pacer class interface

1/30 sec. Pacer::pick is used to find the schedulers in a pacer hierarchy. In EDraw, a UIScheduler is used to set the period of the presentation according to the speed of user manipulation. The mapping between speed and period is completely programmable. This way, the faster the manipulation, the smaller the period, and therefore the presentation will be more degraded and more responsive to the user.

Pacer::get_grades returns the grade boundaries of a pacer. Grade boundaries are useful in culling the interested pacers at a particular grade efficiently. Pacer::regrade is called when the grade boundaries need to be recomputed. Figure 7 shows the subclasses of pacer.

Graders, which are composite pacers, determine how grades of their children are assembled. For example, a tiling composite computes its own grade boundaries by tiling the grades of its children. It plays by selecting the child in the tiling sequence corresponding to the current grade to play. Graders use GradeAssemblers to assemble their children's grades. The rationale behind this approach is that it is much cheaper to change the grade assembling scheme of a presentation by switching assemblers used by the composites than switching in a new set of composites. The later involves significantly more overhead in setting up the parent child relationships of the new composites.

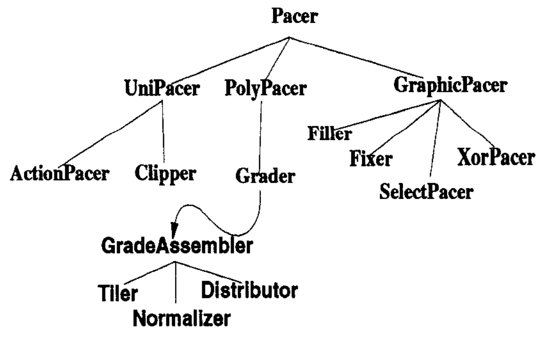


Figure 7 : Subclasses of Pacer

The only media specific pacers we have implemented to date are GraphicPacers, which perform by displaying their corresponding graphics. Subclasses define different rendering techniques. A Filler for instance, simply fills the damaged region with a predefined background color. A Fixer on the other hand, repairs the damaged region by redrawing all the intersecting graphics.

4.2 PaceMaker

A pace maker is a central clock that coordinates all pacer activities with a pace keeper. It defines a sampling period for the related pace keeper and a sampling period per scheduler. For example, a video scheduler, an audio scheduler and a user interface scheduler can all have different periods but they may present simultaneously.

Schedulers use a pace maker to play continuously. A pace keeper decides what grade to play, and it uses a pace maker to define the frequency of grade adjustment.

4.3 PaceKeeper

A pace keeper attempts to adjust the quality of a presentation such that all participating schedulers meet their periodic timing constraints. Subclasses of PaceKeeper implement specific downgrade and upgrade policies. To achieve this, a pace maker delegates the playing of schedulers to the pace keeper, thus letting it monitor the actual time used by the schedulers relative to their periods. When a scheduler overuses, the pace keeper has two options. It can either register the amount overused and delay any grade change decision until its evaluation time, or it can downgrade right away. Similarly, if a scheduler underuses, the pace keeper can attempt to upgrade immediately or delay until evaluation occurs.

Pace keepers typically use a damped oscillation approach for upgrades, and a conservative approach for downgrades. This means whenever a scheduler overuses time, the pace keeper downgrades immediately. When all schedulers underuse time, the pace keeper performs an upgrade attempt. If it is successful, no adjustment is performed. If it fails, the next upgrade attempt will take longer, until a limit is encountered. The advantage of this policy is simplicity. The obvious disadvantage is that it is prone to oscillation.

Another approach involves the use of the amount of time underused or overused to determine when the next upgrade attempt should be performed. This means the more unused time at a certain grade level, the sooner an upgrade occurs. When an upgrade attempt fails, the penalty is determined by how much time was overused. The higher the penalty, the longer the next upgrade attempt will be.

The problem with this approach is that if a presentation has grade levels with widely varying costs, the pace keeper will tend to oscillate between those levels. The pace keeper implemented in EDraw uses a coherence based approach that relates the cost of performance between adjacent grade levels. This policy assumes that if the cost at a certain level varies, it is very likely that the adjacent levels will vary in a similar fashion. A coherence factor is dynamically adjusted to reflect how related the adjacent levels are. This mechanism solves the oscillation problem because an upgrade attempt will not be performed simply because of accumulated unused time at a certain grade level. Upgrades are only performed if there is reason to believe that the next level can be sustained.

5 EDraw implementation

EDraw is implemented using the InterViews[6] toolkit. We also modified the InterViews drawing editor, idraw, so that degraded levels of a drawing can be specified graphically. Figure 8 shows the idraw user interface. After reading in the description of the graphical objects from a file, EDraw creates a pacer hierarchy as shown in Figure 9.

Some properties specific to graphics editors made the pacer composition more complicated. Damage control for instance, requires that the background and all intersecting graphics be redrawn. If damaged regions are repaired at degraded levels, objects may appear to be inconsistent. To accommodate this, three forms of degradation behavior are used.

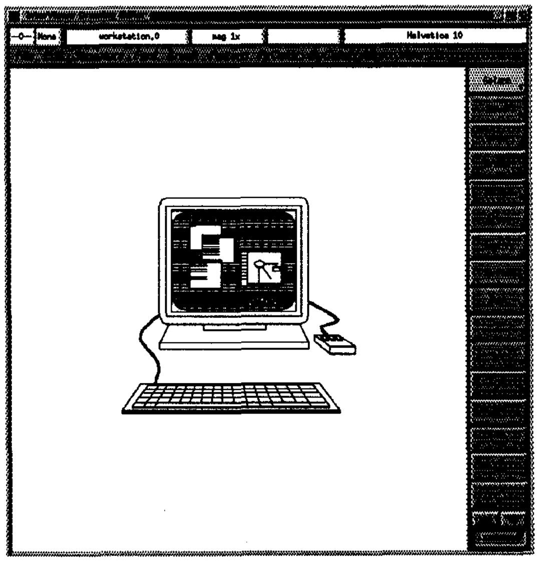
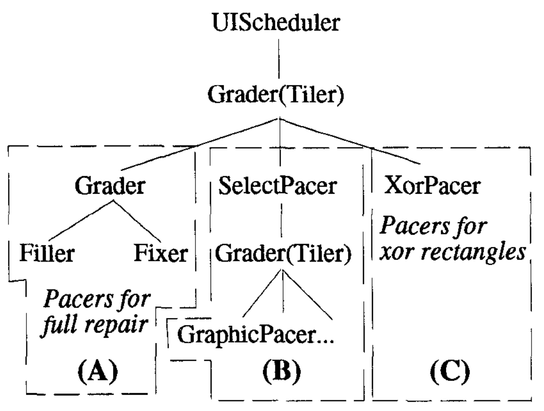


Figure 8 : InterViews drawing editor



Pacers specified by idraw

Figure 9 : Pacer instance hierarchy

Composition A represents a full repair. At this grade level, EDraw always completely repairs all the damaged regions.

Composition B represents a partial repair. In this form of degradation, EDraw only redraws the manipulated object. However, the intersecting background objects frequently get obscured without repair. This behavior may force the user to slow down and wait for a complete repair during manipulation. To avoid this situation, a SelectPacer object changes the selected graphics to use partially-transparent rendering as in Figure 4. The exact number of grade levels in this form can be specified by the creator of the drawing.

Composition C represents the xor grade level. This form of degradation is commonly used by drawing editors.

The top-level scheduler is responsible for mapping input events into refresh frame rate. Currently, when two events occur very close to each other, the scheduler will attempt to maintain a redraw rate of 8 fps. On the other hand, if two events are far apart on the screen, the redraw rate can go as high as 30 fps. The pace keeper takes this information and adjusts the quality of the presentation accordingly.

Figure 10 describes how pacer objects work together with an application. The pacer composite is the same as the one described in Figure 9.

6 Performance

In order to obtain consistent results for the costs of various pacers, a global function called *next* was created to generate events automatically up to a predefined limit set at 40. The before and after times of the next function were monitored, and a cost factor in terms of time usage was computed. We focused on the cost in terms of speed since pacers use relatively little memory (a typical graphic pacer is 24 bytes).

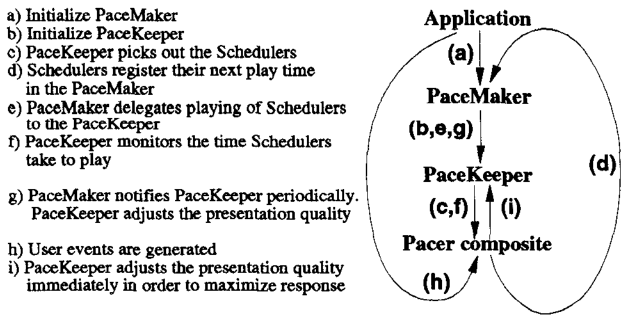
We picked the car example for this experiment because it exhibits more elastic behavior than the rectangle or duck examples. The experiment consisted of three parts. In Part A, we forced presentation of grader subtree in Figure 9 in order to compute the cost of a full repair. In Part B, we forced presentation of the semi-transparent graphics. In Part C, we forced the use of xor rectangles. In each part, the right wheel of the car was selected and the move tool was engaged. The wheel then moved to the upper left, with most of the redraws occurring over the body of the car.

In the second trial, we short-circuited the pace maker's ticking mechanism and all the events were generated in succession. A pacer called a ByPasser avoided the costs associated with the pace maker and pace keeper.

We measured the elapsed times for both trials. The measurements in each trial were repeated 10 times to get average results. All tests were done on a Sun Sparc10 workstation with 32 Mb of memory. The results are shown in Table 1.

	Trial 1 With original pacers	Trial 2 With a bypasser
Part A	4.972 sec	4.905 sec
Part B	3.722 sec	3.653 sec
Part C	0.103 sec	0.087 sec

Table 1: Performance measurements


a) Initialize PaceMaker
b) Initialize PaceKeeper
c) PaceKeeper picks out the Schedulers
d) Schedulers register their next play time in the PaceMaker
e) PaceMaker delegates playing of Schedulers to the PaceKeeper
f) PaceKeeper monitors the time Schedulers take to play

g) PaceMaker notifies PaceKeeper periodically. PaceKeeper adjusts the presentation quality

h) User events are generated
i) PaceKeeper adjusts the presentation quality immediately in order to maximize response

Figure 10 : Pacer logic flow

Intuitively, the more expensive the presentation, the less expensive the relative cost of managing it. The results in Table 1 confirms this intuition. In Part A, five pacers were involved for grade management. However, 35 graphical objects on the average were involved with each redraw. In this case only 1.35% of the total time was lost as overhead. In Part B, five pacers were used to control a total of 28 graphical objects drawn using transparent rendering. The overhead in this case raised to 1.85%. In Part C, three pacers were involved for presenting one xor rectangle. The overhead in this case is 15.5%, which is still pretty low considering this example uses more pacers than media objects.

7 Future work

Pacers control the sampling aspect of presentations. As a result, their APIs are low level. For example, a programmer has to specify information like sampling periods and their readjustments. Given a high level description of the desired degradation behavior, these details should be generated automatically.

Applying multi-threading to pacers is also an interesting problem. Currently in EDraw, if the scheduler increases its period due to lack of input, it may trigger the pace keeper to promote the presentation to a full repair, which may take a significant amount of time. Any user events that arrive after the full repair has started will not get processed until the repair finishes. Using a separate thread to play a pacer, the event dispatcher could preempt an unfinished repair to improve response time.

8 Conclusions

Pacers use timing constraints and composition as uniform mechanisms to define elastic presentations. We have demonstrated the power of pacers in a simple drawing editor that adapts the direct manipulation feedback according to the graphics performance and complexity. Measurements indicate the overhead of pacers is acceptably low, even for complex elastic behavior.

The use of timing information to evaluate the cost of a presentation allows pacers to work effectively across platforms and in the presence of load changes. Authors of time-based presentations can use pacers to specify the elasticity of media components, giving the highest quality presentation possible on the user's platform.

9 Acknowledgments

This project has been supported by a grant from Fujitsu America Inc.

10 References

[1] S. Miller and A. Harris, "The QuickTime How-To Book", Sybex Macintosh Library 1992.

[2] Thomas A. Funkhouser and Carlo H. Sequin, "Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments", Proc. 20th SIGGRAPH, Anaheim, California, Aug. 1993.

[3] L. Rowe and B. Smith, "A Continuous Media Player", Proc. 3rd Int. Workshop on Network and OS Support for Digital Audio and Video, San Diego CA, Nov. 1992.

[4] D. Anderson and G. Homsy, "A Continuous Media I/O Server and its Synchronization Mechanism", IEEE Computer, Oct 1991.

[5] D. Anderson, R. Herrwich, and C. Schaefer, "SRP: A Resource Reservation Protocol for Guaranteed-Performance Communication in the Internet", International Computer Science Institute Technical Report 90-006

[6] P. Calder and M. Linton, "Glyphs: Flyweight objects for user interfaces," Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology, Snowbird, Utah, October 1990, pp. 92-101.

[7] R. Steinmetz, "Synchronization Properties in Multimedia Systems", IEEE Journal on Selected Areas in Communications, Vol 8, No 3, Apr 1990.

[8] G. Blakowski, J. Hubel, and U. Langrehr, "Tools for Specifying and Executing Synchronized Multimedia Presentations", Proc. 2nd Int. Workshop on Network and OS Support for Digital Audio and Video, Heidelberg, Nov 1991.

[9] R. Hamakawa, H. Sakagami, and J. Rekimoto, "Audio and Video Extensions to Graphical User Interface Toolkits", Proc. 3rd Int. Workshop on Network and OS Support for Digital Audio and Video, San Diego CA, Nov 1992.

[10] N. Guimaraes, N. Correia, and T. Carmo, "Programming Time in Multimedia User Interfaces", Proc. 5th ACM Symposium on User Interface Software and Technology.

[11] S. Gibbs, "Composite Multimedia and Active Objects", Proc. OOPSLA '91, Phoenix, Arizona, pp. 97-112.