Master's Project:

Applying Traditional Animation-Techniques to Shared 3D Virtual Worlds

Zornitza Atanassova

Abstract

To improve shared 3D environments as communication media, we propose to apply traditional animation techniques to them. We reason that in some aspects shared 3D environments are similar to traditional animation and that they therefore benefit from the highly developed techniques common in traditional animation. Unlike traditional animation, however, shared 3D environments require the generation of animation in real-time, on the fly, and without complete information. Generating animation for shared 3D environments is therefore challenging. We discuss the techniques of squash and stretch, motion blur, anticipation, and follow-through in detail and implement them. Based on this implementation, we then describe our findings.

1 Introduction

Today many applications are shared 3D environments. They constitute multi-user 3D virtual worlds, in which the participants work, communicate, or play with one another. Examples of such applications exist in the domains of CAD/CAM, simulation, visualization, and entertainment.

In the CAD/CAM area, for example, such shared multi-user applications enable several participants to distribute 3D models, visually inspect these models, share viewpoints, and point out and discuss features. Thus, even if separated by hundreds of miles, participants can collaboratively design and ultimately manufacture products. Multi-player games that employ the Internet to connect like-minded players to one another are another example of shared multi-user virtual environments.

The purpose of many of these shared environments, for example, shared CAD/CAM tools, is to facilitate communication between the participants. In that case, participants tolerate some amount of lag, slight changes of the motion path, and additional artistic touches such as motion-blur, anticipation, and stretching. We concentrate on these kinds of applications in this research. Specifically, we exclude simulation-based environments, e.g., military training applications, that find any additional lag or altered motion paths objectionable.

To follow ongoing discussions, participants in those shared environments need to track the movements and actions of the other participants. Technical limitations, such as limited user-input tracking, limited transmission capabilities, and limited rendering capacity, however, simplify the representation of the other participants and their movements considerably. Consequently, the movements and actions of other participants often become unexpected or confusing. For example, objects seemingly appear, disappear, or disappear from one place and appear in another. Thus following and understanding the actions of multiple other participants becomes challenging.

In comparison, good animation excels at making viewers comprehend even bizarre, simultaneous, and rapid action. Animation accomplishes this task by catching and guiding viewers' attention. Shared virtual environments are comparable with animations in that both present users with time-varying, non-realistic visual information. We therefore suggest to apply to shared virtual environments the techniques that make animations comprehensible. We thus hope to make shared virtual environments easier to observe, and thus ultimately easier to communicate with.

Applying these techniques, however, is not straightforward. Shared virtual worlds differ from animations in several aspects: for one, shared virtual worlds are interactive and real-time, while animations are not. In this paper, we therefore investigate if and how animation-techniques apply to shared virtual environments. We first review related work in Section 2. Then, we list cartoon-style animation techniques and discuss their applicability them remotely, the local user never knows when to expect them. In addition, variable transmission-delays further scramble the timing of these remotely generated animations. To help local users decipher the actions of remote participants, we suggest to automatically modify remote-participant actions to adhere to the language of animation.

Modifying remote-participant actions is hard. For example, to maintain reasonable lag-times, we must modify a remote-participant action while ongoing, i.e., we cannot wait for it to end. Thus, we are creating animations on the fly without knowing how they are going to develop or when they are going to end. In contrast, this difficulty is nonexistent with computer-controlled actions: a user's initiation of a computer-controlled action fully specifies all its parameters before it starts.

3 Review and Rational of Animation Techniques

We group the various animation techniques that constitute the animation language [LASS'87] [THOM'81] into three principles [CHAN'93]: solidity, exaggeration, and reinforcement. Each of the following sections discusses one of these principles. We examine a principle's relevance in general and to 3D shared environments in particular and elucidate on some of the particular techniques supporting it.

3.1 Solidity

Characters and objects appear real in animations only if they maintain an illusion of solidity. Traditional animation therefore draws characters and objects as if three-dimensional. Moreover, objects appear to have mass and move accordingly; all motion takes weight, mass, inertia, and balance of an object into account. In particular, objects squash, stretch, and deform to maintain their volume while reacting to forces. Also, fast moving objects, e.g., objects that move more than their size from frame to frame, motion-blur. If they would not motion-blur, they would appear to move jerkily, i.e., as if repeatedly and quickly starting and stopping from frame to frame, thus appearing to have no mass and destroying the illusion of solidity.

Although shared 3D environments are inherently three-dimensional, they may nonetheless violate this principle of solidity. In particular, due to the required direct-manipulation metaphor, a participant who is manipulating objects has unconstrained control over their motion. A remote participant observing such a remotely manipulated object and its unconstrained movements thus perceives the object to have very little or no mass. Worse, objects in shared 3D environments are in general rigid, i.e., they never deform. The combination of low mass and extreme rigidity in a single object is unnatural and thus unbelievable – it severely undermines the solidity of a shared 3D environment.

Similarly, shared 3D environments do not commonly employ motion blur. Reasons for this exclusion are additional programming overhead, since graphics packages do not support motion blur as part of their standard API, and performance overhead. (Even though adding motion blur does not necessarily imply substantial performance overhead: just as traditional animators substitute correct and thus hard-to-generate motion blur with crude but fast and nonetheless effective approximations, we do the same in Section 4.2.) The result of excluding motion blur from shared 3D environments is the same as excluding it from traditional animations: fast moving objects move jerkily in apparent starts and stops.

Worse, the distributed nature of shared 3D environments often exacerbates jerky movements. A participant manipulating an object typically generates enough updates to generate smooth motion only locally. Due to differences in camera positions a remote participant who zooms in on the remotely manipulated object may not perceive these motions as smooth. This effect persists even if the zoomed-in participant receives all the generated updates in time. The problem only worsens if transmission additionally delays or loses updates.

To give shared 3D environments the appearance of solidity, we propose to apply techniques from traditional animation. The two most common techniques in traditional animation to enforce solidity are squash/stretch and motion blur.

3.1.1 Squash and Stretch

Squash and stretch refers to the deformation objects in animations undergo as they accelerate, decelerate, and move [LASS'87]. Accelerating and decelerating objects squash as the acceleration or deceleration force compresses them; the heavier or the more flexible the object or the higher its acceleration or deceleration, the more the object squashes. A moving object stretches to purvey the sense of speed; again, the more flexible an object or the faster its speed, the more it stretches.

Squashing and stretching objects in shared 3D environments reinforces the principle of solidity. Squashing and stretching applies to all forms of object-movement: direct-manipulation actions, computer-controlled actions, and remote-participant actions (see Section 2.3). In particular for remote-participant actions, however, squashing and stretching restores a sense of mass and thus of reality to remotely manipulated, rapidly accelerating, decelerating, and moving objects.

In addition, stretching a moving object provides important visual cues in a shared 3D environment. In a shared 3D environment, an observer is generally unable to follow all activities: multiple activities may be simultaneously ongoing or the chosen viewpoint may be restrictive. Therefore, an observer may only see part of an action, may start observing an action that is already ongoing, or may even only catch a momentary glimpse of the action. In these cases, the observer's goal is to extrapolate the witnessed movement and find the center of activity to determine whether it is of interest. Stretching a moving object in the direction of motion and proportionally to its speed makes such extrapolation easier, as it indicates in a single frame the direction and speed of the motion.

We describe our implementation of squash and stretch for shared 3D environments in Section 4.1.

3.1.2 Motion Blur

Traditional animation applies motion blur to fast-moving objects to avoid the otherwise apparent jerkiness. Proper motion blur is akin to temporal anti-aliasing [WLOK'96] that is expensive to generate accurately. Traditional animation therefore approximates motion blur in the form of speed-lines or vague color smears. These approximations, while orders of magnitude easier to generate, have the same effect of alleviating jerky movements.

As explained above (in Section 3.1), shared 3D environments require motion blur to maintain solidity. In addition, motion blur provides the same important visual cues in shared 3D environments as stretching (see Section 3.1.1): motion blur depicts the direction of moving objects in a single frame. Because motion-blur visually connects the current with the previous positions of objects, it also eases the task of tracking moving objects.

On the other hand, shared 3D environments already push their performance limits; applying proper motion blur is computationally too expensive to be practical. Instead, we propose a variety of approximations of motion blur modeled after techniques used in traditional animation. These approximations are orders of magnitude faster to generate than proper motion blur, and thus are practical even for shared 3D environments. We describe these approximations in detail in Section 4.2.

3.2 Exaggeration

Animation is a non-interactive communications medium. As such, it presents ideas and actions without its audience being able to influence the happenings. It must therefore communicate its ideas and actions so that an audience grasps them as they appear – if an audience misses an action, it and its idea are lost.

Animators therefore attempt to communicate all actions as clearly as possible via two main means. First, because large and oversimplified movements are more easily and more readily understood than subtle and nuanced ones, animators exaggerate movements. Second and last, an audience follows actions more readily if it knows where the action is heading. Accordingly, animators continuously telegraph the next movement before it happens; they continuously provide cues in the current movement about the following ones. Anticipation and follow-through are the most common examples of this foreshadowing technique [LASS'87].

Shared 3D environments are interactive. In principle, participants could therefore interrupt other participants to ask them to repeat actions. In practice, and especially if such repetitions are frequent or if many participants try to follow the same action, this mode of communication quickly becomes cumbersome. Thus, communicating actions as clearly as possible is also important in shared 3D environments.

We do not attempt the first form of exaggeration: we do not simplify remote-participant actions to make them more easily understood for two reasons. First, to simplify remote-participant actions one would have to recognize their essence on the fly. Distilling the essence of arbitrary, human-generated movements automatically and in real-time is difficult – after all, the reason for solving this task is that it escapes the human observers. Second, because these actions are human-generated, it is possible, even likely, that the nuances they include are essential; stripping actions of their nuances may alter their meaning.

We do incorporate the second form of exaggeration into shared 3D environments: we modify remote-participant actions by adding hints about the imminently following movements. In particular, we apply anticipation to all remote-participant actions.

3.2.1 Anticipation

The animation technique of anticipation applies to objects that are at rest, but about to move. Normally, such objects would simply start to accelerate towards their destination. When applying anticipation, however, these objects first move in the opposite direction, i.e., away from the destination, pause, and only then accelerate towards their destination. The anticipatory drawing-back and pausing is longer the faster the ultimate acceleration is.

Anticipation catches the attention of observers, leads them to expect a following action, and therefore readies them for the imminent movement. It provides information about the direction and speed of the anticipated movement and is therefore similar to squash and stretch (see Section 3.1.1). Using anticipation in conjunction with squash and stretch reinforces both techniques: since anticipation adds lead-time to movements, it makes the squashing and stretching more apparent; squashing and stretching the anticipating objects adds weight to the anticipation.

On the other hand, anticipation has two disadvantages for use in shared 3D environments. First, applying anticipation to an object alters its motion path. Depending on the application domain, such motion-path modifications may be unacceptable. Second and last, applying anticipation to an object also increases its lag-time. This second disadvantage is particularly relevant, since shared 3D environments are distributed, thus already exposed to lag, and therefore highly sensitive to any additional lag. Applying anticipation to objects in shared 3D environments delays the displaying of actual object movements by the amount of time it takes to display the anticipatory movement. Because the shared 3D environment generates this lag explicitly, however, it may be bound, and it may therefore be acceptable in certain applications (see Section 1). In addition, the benefit of adding anticipation may make up for the incurred delay.

We describe our implementation of anticipation in Section 4.3.

3.2.2 Follow Through

Follow-through is similar to anticipation. Unlike applying to objects about to move, however, it instead applies to objects in motion about to come to rest. While normally an object just stops, an object with follow-through overshoots its final destination, comes back, and oscillates into position. The magnitude and frequency of this oscillation depends on the deceleration of the object before stopping. The faster the deceleration, i.e., the more abrupt the stop, the higher is the magnitude and frequency of the oscillation.

Shared 3D environments benefit from follow-through, since it draws attention and thus prepares an observer for the imminent change in animation state. It exaggerates an object's behavior with respect to its weight, mass, inertia, and balance. Follow-through therefore extends and complements the effects of squash and stretch to make objects seem more real (see Section 3.1.1).

The disadvantages of follow-through are the same as for anticipation. Follow-through alters the motion path of an object, and it introduces additional lag (see Section 1).

We describe our implementation of follow-through in Section 4.4.

3.3 Reinforcement

Reinforcement techniques, such as arcs and slow in/out, help strengthen the effects that primary animation techniques (described in Sections 3.1 and 3.2) are designed to achieve. These reinforcement techniques, while subtle, are nonetheless key in traditional animation, since they largely determine an animation's particular style or feel.

We feel that reinforcement techniques are ultimately going to be important in shared 3D environments. Just as they differentiate animation styles, they could define and differentiate shared 3D environments. For now, however, the mechanics of applying animation techniques in general and reinforcement techniques in particular prove difficult enough to prevent any original artistic designs.

We discuss two common reinforcement techniques in more detail in the next two sections, arcs and slow-in/out. In particular, we show the problems of applying these techniques to shared 3D environments. While we are unable to solve these problems (and accordingly have no corresponding implementation descriptions in Section 4), these discussions might nonetheless provide useful insight to the reader.

3.3.1 Arcs

Due to gravity, most natural objects move in slight curves. Consequently, objects with unbend, straight motion paths seem unnatural and possibly weightless to an observer. Accordingly, objects in traditional animation rarely move in straight lines; rather, they arc to reinforce solidity.

Applying the arcing reinforcement technique in shared 3D environment is problematic for two reasons. First, changing straight-line motion paths to arced ones alters the original motion. As mentioned in Section 3.2, depending on the application, such alteration might be unacceptable.

Second, arcing of remote-participant actions lacks vital information. Visually pleasing arcs are of constant or nearly constant curvature. To transform an object's straight-line path from point A to point B to a single, constant-curvature arc that connects the same two points, one requires knowledge of both A and B. For remote-participant actions, however, we need to begin to transform the straight-line path of an object from A to B upon receipt of A only. If waiting for an additional update (and thus introducing lag) is acceptable, we at least know A and the tangent of the straight-line motion before we must start moving the object along its motion path. Even this additional information, however, is insufficient to determine a constant-curvature arc that ultimately connects point A to point B.

3.3.2 Slow In and Slow Out

Slow in and slow out refers to the timing of movement of an object. Roughly, slow in and slow out correspond to an object's acceleration and deceleration, respectively. Including slow in and slow out in the timing of the motion of an object corresponds to that object overcoming its inertia; it is therefore more natural than moving objects with infinite acceleration or deceleration. Slow in and slow out therefore also reinforce the solidity of objects and they are common in traditional animation.

The problems of applying slow in and slow out to shared 3D environments are similar to those of applying arcs. First, changing the timing of motion is similar to changing its path: altering the timing of motion may be unacceptable for certain types of shared 3D environments.

Second, natural looking motion has constant or nearly constant acceleration and deceleration. Adding constant acceleration to a constant velocity motion is straightforward: one simply accelerates the object until it reaches the original velocity. Nevertheless, choosing the acceleration constant is under-constrained. In particular for remote-participant actions, one could conceivably choose an acceleration constant that prevents an object to ever reach its target velocity before its motion ends, as the time the motion ends is unknown when acceleration starts.

Worse, smoothly decelerating remote-participant actions seems impossible. Because remote-participant actions originate as interactive actions, predicting when one ends is impossible. The earliest we know of an action ending is therefore just after its end. Other participants in the shared 3D environments thus also know when a remote-participant action terminates only after its ending. At this time, the current position of an object under the remote-participants control is typically at or at least very close to its final resting position. Adding deceleration to remote-participant actions would therefore not only add delay (due to postponing the termination of the motion), but also alter the final resting-place of the object in motion.

4 Implementation Notes and Findings

Below we describe our implementation of the various animation techniques. We use an existing shared 3D environment, VR-APP [CONN'97], to embed and test our implementations. Accordingly, we then compare the original VR-APP to our modified version that displays the effects of the animation techniques. While we list the advantages of animation techniques in Section 3, here we concentrate on shortcomings and difficulties both of the application of an animation technique to VR-APP and of our particular implementation.

4.1 Stretching

Because VR-APP supports neither the notion of mass nor the notion of flexibility of objects, our implementation of stretching considers only an object's direction and speed. Objects stretch along their motion vector proportionally to their velocity (see Color Figure 1). In essence, we treat all objects as having the same mass and flexibility.

We measure the velocity of an object at the site where the motion originates. Broadcasting time-stamps along with positional updates is a sufficient solution. In contrast, we find that measuring velocity of an object at the receiving end, e.g., to save the communications bandwidth needed to transmit the time-stamps, is unsatisfactory. It is equivalent to time-stamping the positional updates when they arrive. Varying delays during transmission, however, distort the time-stamps and therefore the velocity of an object. The observed velocity becomes artificially irregular. Stretching, since it emphasizes object velocity, further amplifies these distortions.

Varying transmission-delays nonetheless affect the perceived velocity, even when measuring velocity at the originating site. VR-APP updates the position of a remote-controlled object when and only when it receives a positional update from the network. Therefore, if transmission is in bursts, a receiving site does not receive updates for moving objects for several rendering cycles as transmission holds them up – the object appears to be static. Then, as transmission delivers all held-up updates during a single rendering cycle to the receiving site, each update overwrites the previous one and only the last one renders – the object jumps to the latest position. An originally smoothly moving object appears to move in fits and jumps. Because our implementation of stretching uses and therefore depicts the original velocity of an object, it alleviates these fits and jumps, even though update-delivery and thus update-rendering is as irregular as before.

4.1.1 Adding Delay

To further smooth object movement, we implement the option of artificially delaying all incoming updates. Instead of always rendering the latest received update, only updates, that are at least a minimum number of milliseconds old, render. An update's age is the difference between its creation (not its arrival time) and the current time. Adding delay to all updates buffers the rendering from the irregularities of transmission and, therefore, preserves the original velocity of objects. It decreases and possibly eliminates artificial jerkiness, because most of the time it is able to provide an update at the time of rendering. It unfortunately also adds lag. The additional lag, however, is user-selectable, bound, and hopefully small in comparison to the overall network-delay. For example, the user may choose this additional lag to be the maximum of all but the top 10% of all observed network delays. Finally, adding delay also incurs storage overhead, since the receiving site stores multiple received updates until each renders.

4.2 Motion Blur

We implement several versions of motion blur; the three main variations are parallel speed-lines, tapered speed-lines, and comet tail. Each major variations, however, has several sub-variations. We therefore describe our general construction of motion-blur and, along the way, point out the various options available.

In general, all our versions construct motion blur in the following manner. First, they define the contour of an object. Second, they construct an interpolating motion spline based on the motion of the object, its previously defined contour, and a user-defined motion-blur length. Third and last, they render a set of motion splines using a variety of attributes. All generated blurs are two-dimensional, i.e. they have no volume. The following subsections describe these three steps in more detail.

4.2.1 Motion Contour

The contour of an object is the set of all edges constituting the outside border of the film-plane projection of the object. The trailing-contour consists of the edges of the contour that trail the object with respect to the film-plane projection of the motion vector of the object. This trailing contour represents a good motion-contour for constructing the motion blur of an object.

Finding this trailing-contour of an object, however, is time-consuming. To speed up computation, we simplify the motion-contour considerably. We project the vertices of an object into the film-plane, then find, and store only three points. Two of these points are the top- and bottom-points: they are the two vertices furthest apart in the direction perpendicular to the projected motion vector (see Figure 2). The third point is the trail-point: it is the vertex with the largest distance from the object center in the direction of the negative motion vector (see Figure 2). Our motion-contour is either the line connecting the top- and bottom-points, let us call it the line-contour, or the triangle formed by all three points, the triangle-contour.

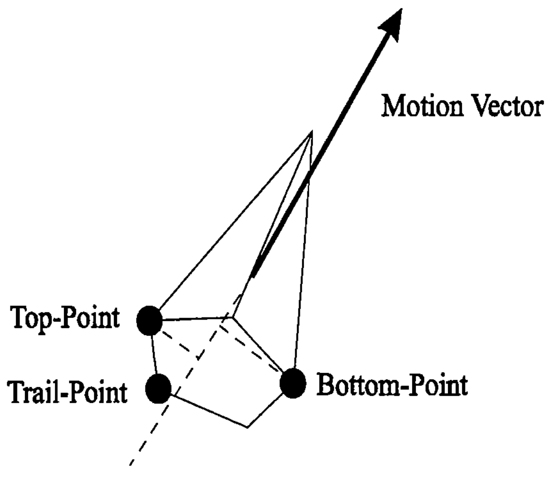


Figure 2: The top-, bottom-, and trail-points of an object.

Although the triangle-contour models the shape of an object better, in practice we find little difference to the line-contour when applied to parallel and tapered speed lines (see Section 4.2.3 below). For the comet-tail (see Section 4.2.3), however, applying the triangle-contour creates the unwanted illusion that the motion-blur folds in the middle, i.e., it appears to have a sharp crease, especially if transparency increases towards the edges (see Section 4.2.3).

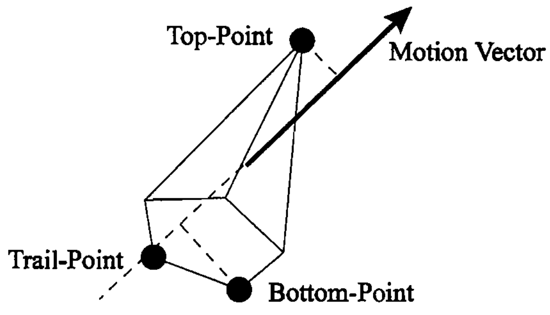
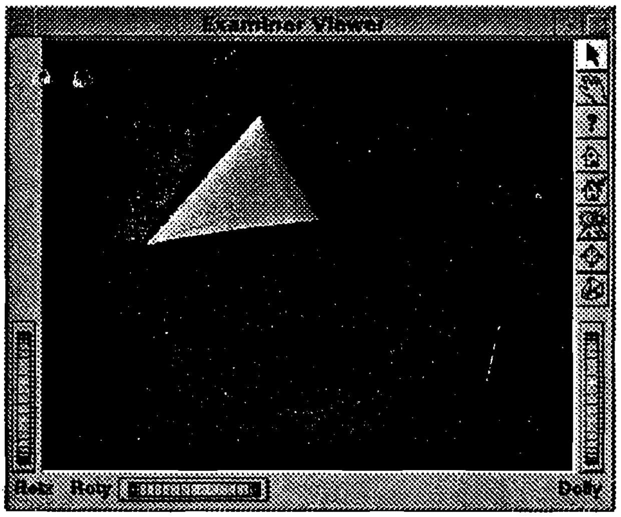


Figure 3: A small change in the direction of the motion vector (compare to Figure 2) drastically changes the contour-points.

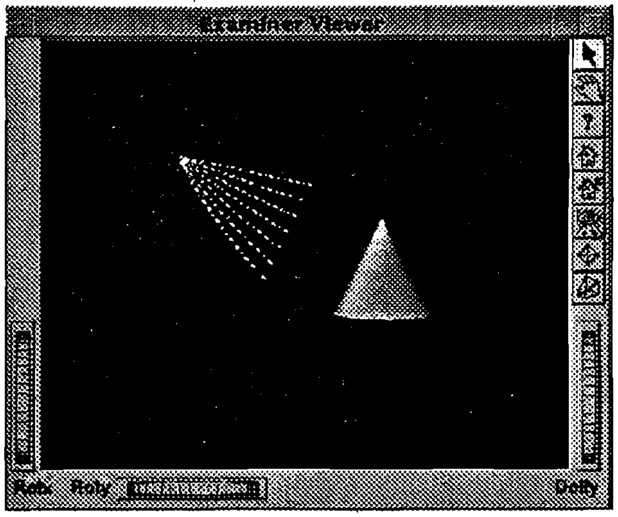
A final variation of constructing the motion-contour attempts to repair a defect of the line- and triangle-contours. Because the line- and triangle-contours construct the top- and bottom-points from vertices originating with the polyhedral representation of the object, these contours change discretely as the motion vector changes with respect to the orientation of the object. Therefore, whenever the object changes direction without also rotating into that new direction, the consequent top-, bottom-, and trail-points may be vastly different from the previous ones, even if the motion-vector change is only small (see Figures 2 and 3).

To counteract these discrete jumps we choose different top-, bottom-, and trail-points for the contours. While the method of computing these points remains unchanged, we do change the set of points from which to choose. We replace the previously used set of vertices stemming from the polyhedral representation of an object with the set of points representing the surface of a sphere. We inscribe this sphere into the object and maximize its radius while still fully contained within the object. We call the resulting motion-contour the sphere-contour.

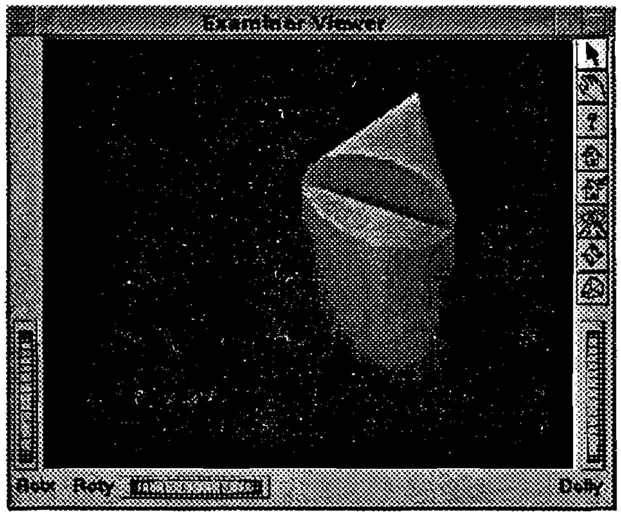
Computing the motion-contour in this manner solves the problem of discretely changing blurs while changing direction. It, however, applies only to objects that grossly look spherical. If a chosen sphere approximates an object badly, for example, if an object is concave or articulated, then the generated motion blur seems unrelated to the originating object.



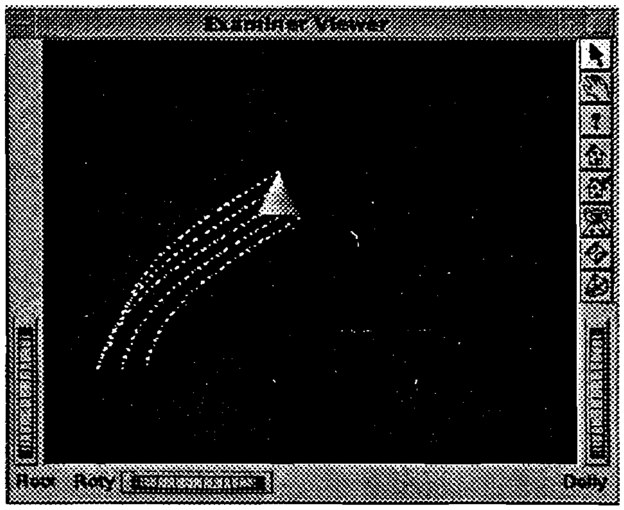
Color Figure 1: A cone stretches along its motion vector.



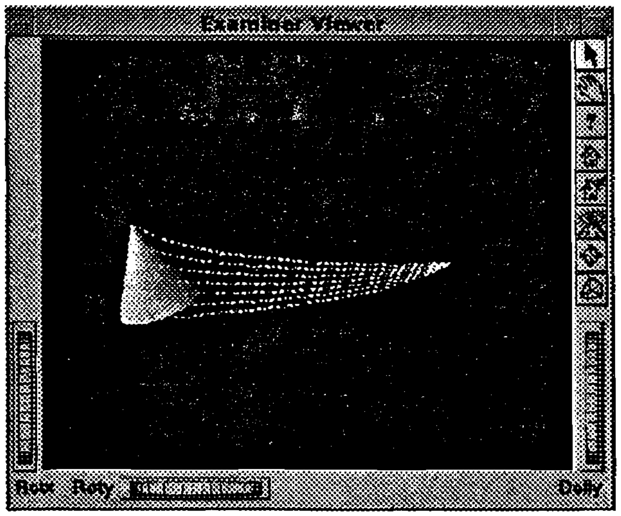
Color Figure 4: Our cubic splines generate a gap between an object and its motion blur.



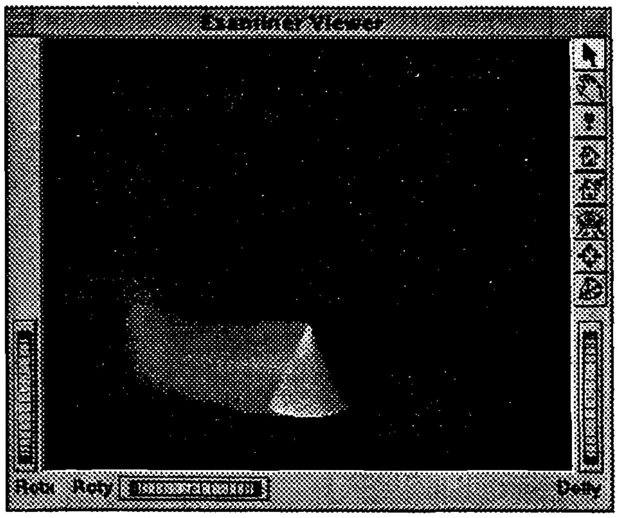
Color Figure 7: To avoid intersecting an object with its blur (shown here), we project the blur onto a plane passing through the object-vertex furthest away from the camera.



Color Figure 8: Stippled speed-lines generated from a set of splines attached to a triangle-contour.



Color Figure 9: Stippled speed-lines generated from a tapered set of splines attached to a line-contour.



Color Figure 10: A comet-tail generated from a tapered set of splines.