An Interface for Sketching 3D Curves

Jonathan M. Cohen, Lee Markosian, Robert C. Zeleznik, and John F. Hughes* Ronen Barzel †

Brown University‡ PIXAR

Abstract

The ability to specify nonplanar 3D curves is of fundamental importance in 3D modeling and animation systems. Effective techniques for specifying such curves using 2D input devices are desirable, but existing methods typically require the user to edit the curve from several viewpoints. We present a novel method for specifying 3D curves with 2D input from a single viewpoint. The user first draws the curve as it appears from the current viewpoint, and then draws its shadow on the floor plane. The system correlates the curve with its shadow to compute the curve's 3D shape. This method is more "natural" than existing methods in that it leverages skills that many artists and designers have developed from work with pencil and paper.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques

Keywords: Curve Manipulation, 3D Modeling, Interactive Shadows.

1 INTRODUCTION

Specifying 3D curves is one of the most important tasks that a 3D user interface must support. Curves are used in modeling and CAD systems to specify surface patches [2, 1], as skeletal shapes for implicit surfaces [5], and to define controls for object deformations [18]. Animation systems and VR applications use curves to specify motion and camera paths [7, 14, 12].

Many authors have recognized the importance of specifying curves directly [9, 10, 21, 4, 20, 13]. Although sketched curves are imprecise by nature, sketching allows a user to quickly create a curve that is close to the desired result, even if she has little experience with the underlying curve representation. A novice user can quickly create approximate curves because little overhead is required to learn the interface. A trained artist can apply her existing drawing skills to produce accurate curves because the interface more closely matches the one she is used to – namely pencil and paper.

The technique we present is an extension of the idea used in [11, 20] that a point in 3D can be determined from its image-space projection together with that of its "shadow." (The "shadow" is just the vertical projection of the point onto some horizontal surface.) We apply this idea to a connected set of 3D points to define a curve. With this approach, the user sketches a curve directly into a scene in two strokes: first drawing the curve as it appears from the current viewpoint, and then sketching its approximate "shadow." The effect is to redefine the 3D shape of the curve while leaving its appearance unchanged. The user can then refine portions of the curve by over-sketching either its projected image or that of its shadow. Although this technique is less precise than existing ones, it lets the user quickly sketch a reasonably correct shape that may be further refined with more conventional methods.

2 PREVIOUS WORK

Some techniques for editing curves are indirect in that they require the user to modify parameters, e.g. spline control points or knot values, that in turn affect the curve's shape. Other techniques allow direct manipulation of the curve itself, such as the overdrawing paradigm described by Baudel [4] and direct manipulation of spline curves [9, 10, 21]. The technique we present falls into this latter category.

Although much work has been done in sketching 2D curves [4, 13, 3, 17], few systems have addressed the issue of sketching curves in 3D. One notable exception is the 3-Draw system [16], which uses Polhemus trackers attached to a stylus to allow a designer to sketch in 3D with arm motions.

Commercial 3D modeling systems such as Maya and 3DStudio [1, 2] give the user a variety of techniques for creating and editing curves. Most of these, however, are indirect (e.g. the user edits spline control points or intersects two surfaces). In Maya a user can sketch a curve directly onto a user-defined plane, or more generally onto a surface. Of these techniques, only the latter constitutes a direct method for specifying nonplanar curves. But the user can draw on a surface only where nothing occludes it. To draw all the way around a sphere, for example, the user must draw from multiple camera positions. Thus, there are restrictions on the types of curves that can be sketched from a single view.

The interface we present complements existing 3D modeling systems by providing additional flexibility for directly sketching nonplanar curves.

3 OVERVIEW OF THE SYSTEM

We support four basic operations for sketching curves: drawing a new curve in some plane, "overdrawing" a section of an existing curve, redefining a curve's entire shadow, and overdrawing a section of a curve's shadow. Figures 1, 2, 3, and 4

*[jmc,lem,bcz,jfh]@cs.brown.edu

†ronen@pixar.com

‡Brown University Site of the NSF Science and Technology Center for Computer Graphics and Scientific Visualization, Providence, RI 02912

1999 Symposium on Interactive 3D Graphics Atlanta GAUSA

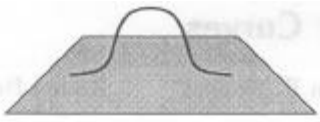

Figure 1: A single stroke creates the initial curve.

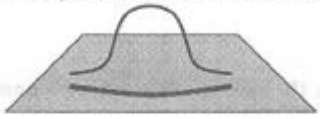

Figure 2: A second stroke defines the curve's shadow and hence its 3D shape.

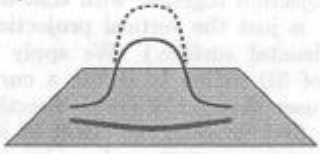

Figure 3: The dashed line indicates an overdraw stroke.

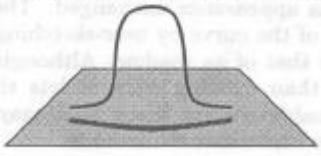

Figure 4: The system blends the overdraw with the original curve to get the final result.

illustrate the steps involved in creating and editing a curve. To distinguish between operations that edit the shadow and operations that edit the curve, the user selects either shadow mode or curve mode via a menu or keyboard shortcut.

When the user draws a stroke in curve mode or shadow mode, the system determines whether the stroke is an "overdraw" by checking whether it starts and ends near and nearly parallel to an existing curve or shadow. If so, we merge it into the existing curve using a method similar to that described in [4]. In curve mode, if the stroke is not an overdraw, the system interprets it as a new curve that is projected onto a plane; the plane is determined by a set of heuristics described below.

To define a shadow, the user (in shadow mode) draws a stroke beneath the curve to be modified. If the stroke appears to be an overdraw, the system blends it into an existing shadow. If there is no existing shadow with which the new shadow can be merged, we test whether its endpoints he approximately below some curve's endpoints. If so, we take this to mean than the the curve's shadow was entirely redrawn. (If not, the stroke is rejected.) Finally, we reproject the curve back into the scene to match its new shadow.

3.1 Drawing Curves and Shadows

We represent 3D curves as parameterized polylines, i.e., as piecewise linear curves defined by a mapping from $[0, l] \rightarrow \mathbf{R}^3$. Before they are used to define curves or shadows, input strokes are smoothed in the following way. First we filter the stroke to remove all points whose screen-space distance is less than some threshold (e.g. 25 pixels) from the previous point. We fit a Catmull-Rom spline [8] to the remaining points and sample the spline every few pixels to generate a smooth-looking polyline.[1]

When a curve is first drawn, we project the 2D stroke onto a plane in world space to create a 3D planar curve. We choose the plane as follows. If either endpoint appears to he on an existing object (or curve), we take this as intentional and place the endpoint in 3D so that it lies on the existing object. We then choose a plane that contains the endpoint (or points). Since one or two points do not uniquely determine a plane, we choose, among all planes containing them, the one that is most nearly screen-parallel.

If neither endpoint appears to he on an existing object in the scene, we determine which plane to use from the angle of the camera. If the camera is looking down, we use the floor plane, and if the camera is at an oblique angle, we use the plane perpendicular to the floor plane that is most nearly screen-parallel.

A shadow is a 3D curve obtained by projecting another 3D curve along a fixed vector, which we call the *projection* vector, onto some surface. In this discussion, we always use the world Y axis as the projection vector, and we always project onto the floor plane. These choices are arbitrary – we could just as easily use the world X vector and let the user draw shadows on a wall. Also, note that in all of our examples, the shadow is a planar curve. This assumption is not necessary for any of the algorithms described below. Thus, we could project shadows onto rolling terrain, for instance.

The key feature of this system is the ability to edit a curve via its shadow. As noted in [11, 20, 19], a point's location is determined uniquely by its appearance from an oblique camera position and by its shadow. We extend this idea to curves: the shape of a 3D curve is determined by its image-space projection and its shadow.[2] Thus, to modify a curve's shape in our system, the user redraws its shadow. This redefines the curve's shape while leaving its appearance from the current camera position unchanged.

It can be difficult to draw a valid shadow for a given curve. To facilitate this, we draw vertical guidelines at both ends of the curve. These lines provide feedback that helps the user align the shadow with the curve. Also, the matching algorithm does not require that the curve and shadow be exactly aligned, only that they be "close," as explained in the next section.

3.2 Correlating Curves with Shadows

Once a curve's shadow or image-space projection has been redefined, we project the curve back into the scene using the following method.

We assume either a perspective or orthogonal projection, with the restriction that the camera's "look vector" is not close to parallel with the projection vector. In a perspective projection, the vanishing point for vertical lines must be off the screen. This allows us to define a left-to-right ordering of 3D points (see figure 5). To test if a point A is left or right of another point B, we project A into the image. Then we take the line parallel to the projection vector running through A's world location and project this line into the image. This line (call it I) partitions the image into two sections, one to the left and one to the right. If the image-space projection of B is to the left of I, we say B is *image-space left* of A and similarly for *image-space right*. If B lies on 1, we say A and

[1] This smoothing step, while independent of the overall technique, is important since noise in the input device propagates to the final 3D curves.

[2] In certain cases described below, the curve's 3D shape is not determined uniquely.

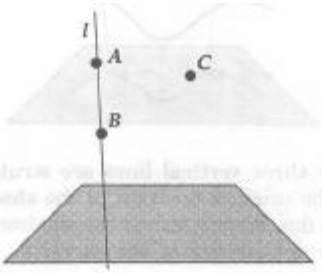

Figure 5: A and B are image-space aligned through the line l, and C is image-space right of both of them.

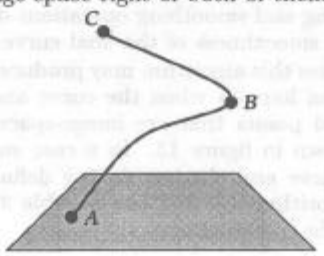

Figure 6: A, B, and C are critical points of this curve.

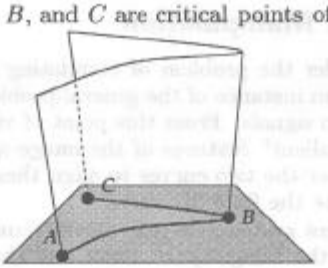

Figure 7: The shadow defines a ruled surface with a silhouette above the interior critical point of the shadow, B.

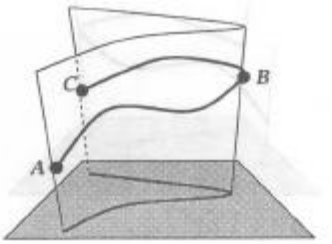

Figure 8: The curve must turn around at B to stay on the surface.

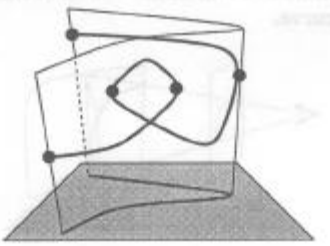

Figure 9: The curve may have more critical points than the shadow and still be valid.

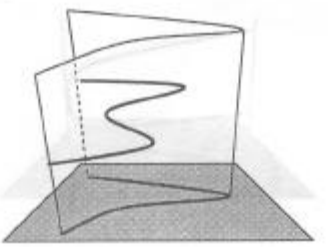

Figure 10: There is no way to project this curve onto the surface to get a continuous 3D curve.

B are *image-space aligned*. If B is within a distance d of l, we say B is *aligned to within d of A* [3].

We define a point on a curve to be a *critical point* if some neighborhood of the point lies entirely to the right or left of the point in image space. Note that by this definition, the first and last points of a curve are critical points (see figure 7). A *span* is the section of a curve between two critical points.

The key observation we use to match a curve with its shadow is that the shadow defines a ruled surface formed by extruding the shadow along the projection vector. An interior critical point in the shadow corresponds to a silhouette of the surface, as shown in figure 7.

To redefine a curve's 3D shape from its shadow, we must project the curve onto this possibly many-layered surface. To do this, we must determine onto which layer of the shadow surface we should project each point of the curve. In general, a critical point on the shadow must correspond with a critical point on the curve. This is because the curve must turn around where the shadow surface turns around in order to stay on the shadow surface, as shown in figure 8. Note that there may be critical points in the curve that do not correspond to critical points of the shadow, as in figure 9.

If there is no way to project the curve onto the surface so that the resulting 3D curve is continuous, the shadow is invalid and we reject it. Figure 10 shows an invalid shadow.

To create the correspondence between layers of the shadow surface and spans of the curve, we need to match critical points of the shadow with critical points of the curve. First, we find the critical points of the shadow and curve. Let $c_0, c_1, \ldots, c_n$ be the list of critical points of the curve, sorted by parameter value, and $s_0, s_1, \ldots, s_m$ be the sorted list of the shadow's critical points.

To begin, we verify that c_0 corresponds with s_0. (At this point, we may have to reverse the parameterization of the curve to make these points match.) We iterate through the critical points of the shadow in order and attempt to match each point s_i with some critical point on the curve. We do this as follows.

First, let c_j be the next unmatched critical point on the curve. If c_j is aligned to within 25 pixels of s_i, then c_j is matched with s_i, and we go on to s_{i+1}. Otherwise, we test if c_j is between s_{i-1} and s_i (in terms of the image-space left-to-right ordering). If not, then there is no valid match because there is no shadow underneath some span of the curve adjacent to c_j. Also, if c_j is the final endpoint of the curve and is unmatched, then there is no valid match. Finally, we increment j and repeat for the next critical point on the curve.

The system can match a shadow with a curve even when the shadow does not align exactly with the curve, as follows. After creating a correspondence between critical points, we

[3] Note that in a perspective projection, *aligned to within d* is not a symmetric relation.

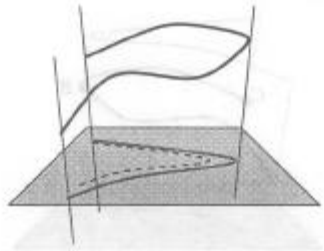

Figure 11: The dashed line indicates the original shadow. The solid line indicates the shadow after it has been adjusted to match the curve.

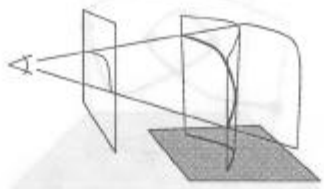

Figure 12: The curve defines a surface containing all rays from the viewpoint through each point on the curve. We intersect this with the shadow surface to get the final 3D curve.

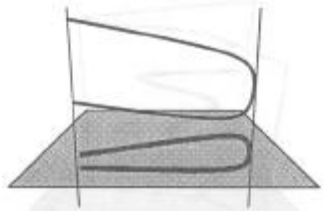

Figure 13: The shadow and curve do not determine a unique 3D curve. This is because all four endpoints are image-space aligned.

deform the shadow so that all matching critical points are precisely image-space aligned. This is done by rotating and scaling each span of the shadow to align it with the corresponding span of the curve, as shown in figure 11. This step allows the user to sketch an approximate shadow, leaving it to the system to ensure that curve and shadow are precisely aligned.

We now have a valid aligned shadow and a correspondence between each span of the shadow and some span of the curve. Just as the shadow defines a particular surface, we can think of the curve as defining a unique surface containing all rays extending from the camera through the image-space projection of the curve. We intersect each portion of the curve surface with the corresponding layer of the shadow surface to produce a section of the 3D curve, as shown in figure 12. Because we use a piecewise linear representation for our curves, we intersect these two surfaces by breaking them up into planar segments and intersecting the corresponding segments. We splice all such sections together to get the final 3D curve.

Near a critical point on the shadow, the tangent plane to the shadow surface is oriented nearly edge-on to the camera. This has the effect of magnifying noise in the 2D input: that is, small variations in the input stroke result in large variations in depth for the 3D curve. To alleviate this problem, we remove points that are nearly aligned to critical points of the shadow, replacing them with a smooth spline that joins

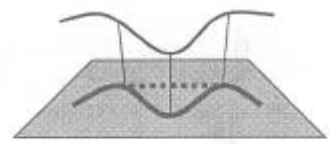

Figure 14: The three vertical lines are struts. The dashed line indicates the original position of the shadow. Dragging the center strut downwards moves the shadow along the floor while fixing the appearance of the curve.

neighboring sections of the 3D curve. Finally, we perform the same filtering and smoothing operations described above to improve the smoothness of the final curve.[4]

In certain cases this algorithm may produce an unintended result. This can happen when the curve and shadow have multiple critical points that are image-space aligned. One example is shown in figure 13. In a case such as this, the image-space curve and shadow do not define a unique 3D curve. Our algorithm will find one possible 3D curve, but it might not be the intended one.

3.3 Strut Manipulation

We can consider the problem of correlating a curve with a shadow to be an instance of the general problem of matching features in two signals. From this point of view, we wish to extract the "salient" features of the image-space curve and shadow, register the two curves to align these features, and finally calculate the final 3D curve.

In our current system, we take into account only the critical points of the image-space curve and shadow. That is, critical points are the only features we consider to be salient. The system ignores other features, such as bends, because we assume that the user will draw such features in correct alignment (figure 17 shows what happens when this is assumption is incorrect).

It might be useful to provide automatic registration of such features – one possible way to do this would be to adapt the dynamic timewarping algorithm from [6] to register the curve with the shadow.

Though we do not currently support this more general notion of signal matching, we do let the user explicitly align certain features. To facilitate this, we allow the user to draw *struts*, which are lines parallel to the projection vector that connect a fixed point on the shadow with a fixed point on the curve. After placing a strut, the user may drag its bottom up or down. This has the effect of adjusting the shadow while leaving the appearance of the curve unchanged, as shown in figure 14. The shadow is affected just in the span between the two neighboring struts.

4 DISCUSSION

Color Plates 14 - 17 show our system in action.

This system is well suited for applications that require fast specification of approximate 3D curves. Applications that require more precise curves might still benefit from this technique, because of the lack of overhead required and the simplicity of the interface. In one scenario, the user would

[4] **Because we perform these filtering and smoothing steps, the appearance of the curve is not constant – it often changes by a few pixels after each edit.**

quickly sketch an approximate curve, then refine its shape with more conventional techniques.

We mentioned previously that our technique can be extended to allow shadows on walls or nonplanar surfaces. There is also no reason to restrict the user to drawing the curve and shadow from the same point of view. The user might draw the shadow from an overhead camera position (thus specifying the shadow more accurately), then sketch the curve from an oblique viewpoint.

A limitation of this method is that it can be quite hard to judge what the shadow should look like for complex 3D curves, especially from an oblique viewpoint. We have observed that users in our lab, even those with artistic training, have considerable difficulty drawing corkscrews and other spiraling shapes. In such cases, a better solution might be to use a 3D input device such as a Phantom or a 3D tracker.

5 FUTURE WORK

We use context-sensitive commands to indicate oversketching operations and keyboard modifiers to indicate modes and to differentiate between different editing operations. Although this works, it is neither consistent nor supported by user studies. We would like to find a more streamlined user interface, perhaps using marking menus or gestural commands [15]. We would also like to have more users try this system, especially users with artistic training but little experience with computer graphics tools.

Finally, we have started to use this curve-sketching technique within a sketch-based free-form modeling system. We believe this interface is a good starting point from which to build a modeling system that leverages a user's talent with pencil and paper to create more complicated shapes than was possible with the original SKETCH system [20].

6 ACKNOWLEDGMENTS

We thank Andries van Dam and the Graphics Group. This work is supported in part by the NSF Graphics and Visualization Center, Advanced Network and Services, Alias/Wavefront, Autodesk, IBM, Intel, Microsoft, National Tele-Immersion Initiative, Sun Microsystems, and TACO.

References

[1] Alias / Wavefront. *Maya*, 1.0 edition, 1998.

[2] Autodesk. *3D Studio MAX*, 1996.

[3] Michael J. Banks and Elaine Cohen. Realtime spline curves from interactively sketched data. In *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, pages 99–107, 1990.

[4] Thomas Baudel. A mark-based interaction paradigm for free-hand drawing. In *Proceedings of UIST 94*, pages 185–192. ACM SIGGRAPH, 1994.

[5] Jules Bloomenthal and Brian Wyvill. Interactive techniques for implicit modeling. In *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, pages 109–116, 1990.

[6] Armin Bruderlin and Lance Williams. Motion signal processing. In *SIGGRAPH 95 Conference Proceedings*, pages 97–104. ACM SIGGRAPH, August 1995.

[7] M. F. Cohen. Interactive spacetime control for animation. In *SIGGRAPH 92 Conference Proceedings*, pages 293–302. ACM SIGGRAPH, July 1992.

[8] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, third edition, 1993.

[9] Barry M. Fowler and Richard H. Bartels. Constraint-based curve manipulation. *IEEE Computer Graphics and Applications*, pages 43–49, September 1993.

[10] Cindy Grimm and Matthew Ayers. A framework for synchronized editing of multiple curve representations. In *EUROGRAPHICS '98*, pages C–31 – C–40, 1998.

[11] Kenneth P. Herndon, Robert C. Zeleznik, Daniel C. Robbins, D. Brookshire Conner, S. Scott Snibbe, and Andries van Dam. Interactive shadows. In *Proceedings of UIST 92*, pages 1–6. ACM SIGGRAPH, November 1992.

[12] T. Igarashi, R. Kadobayashi, K. Mase, and H. Tanaka. Path drawing for 3d walkthrough. In *Proceedings of UIST 98*, pages 173–174. ACM SIGGRAPH, 1998.

[13] T. Igarashi, S. Matsuoka, S. Kawachiya, and H. Tanaka. Pegasus: A drawing system for rapid geometric design. In *CHI'98 Summary (ACM Conference on Human Factors in Computing Systems)*, pages 24–25, 1998.

[14] R. Pausch, T. Burnette, D. Brockway, and M. E. Weiblen. Navigation and locomotion in virtual worlds via flight into hand-held miniatures. In *SIGGRAPH 95 Conference Proceedings*, pages 399–400. ACM SIGGRAPH, 1995.

[15] Dean Rubine. Specifying gestures by example. In *SIGGRAPH 91 Conference Proceedings*, pages 329–337. ACM SIGGRAPH, August 1991.

[16] Emanuel Sachs, Andrew Roberts, and David Stoops. 3-draw: A tool for designing 3d shapes. *IEEE Computer Graphics and Applications*, pages 18–25, November 1991.

[17] P.H. Schneider. An algorithm for automatically fitting digitized curves. In A. Glassner, editor, *Graphics Gems*. Academic Press, 1990.

[18] Karan Singh and Eugene Fiume. Wires: A geometric deformation technique. In *SIGGRAPH 98 Conference Proceedings*, pages 405–414. ACM SIGGRAPH, July 1998.

[19] Robert C. Zeleznik, Andrew S. Forsberg, and Paul S. Strauss. Two pointer input for 3d interaction. In *Computer Graphics (1997 Symposium on Interactive 3D Graphics)*, April 1997.

[20] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. Sketch: An interface for sketching 3d scenes. In *SIGGRAPH 96 Conference Proceedings*, pages 163–170. ACM SIGGRAPH, August 1996.

[21] J. M. Zheng, K.W. Chan, and I. Gibson. A new approach for direct manipulation of free-form curves. In *EUROGRAPHICS '98*, pages C–327 – C–334, 1998.

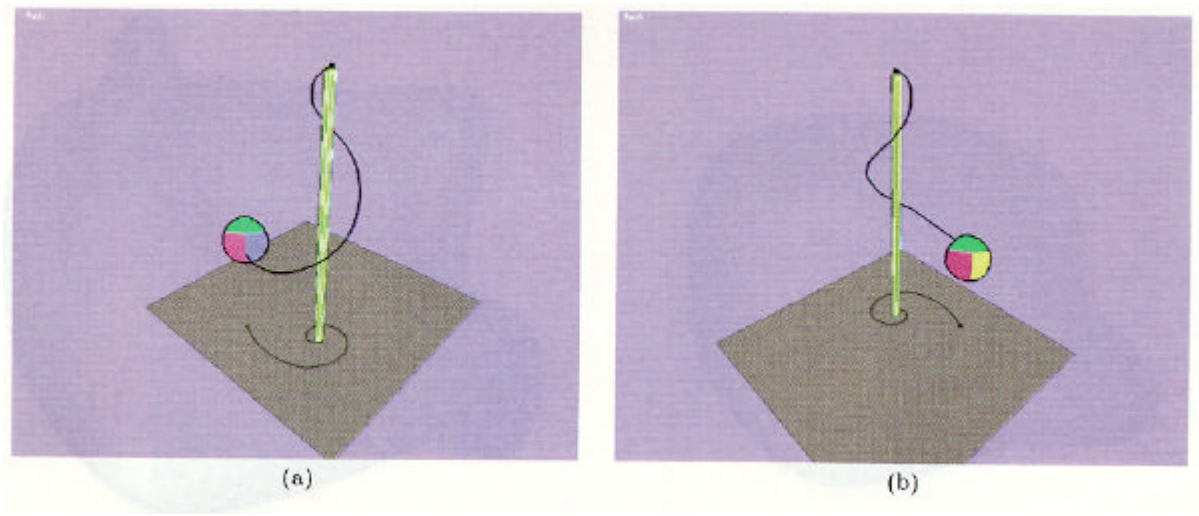

Figure 15: The rope in the tetherball scene was drawn from the viewpoint in [a]. Figure [b] shows the same scene from a different viewpoint.

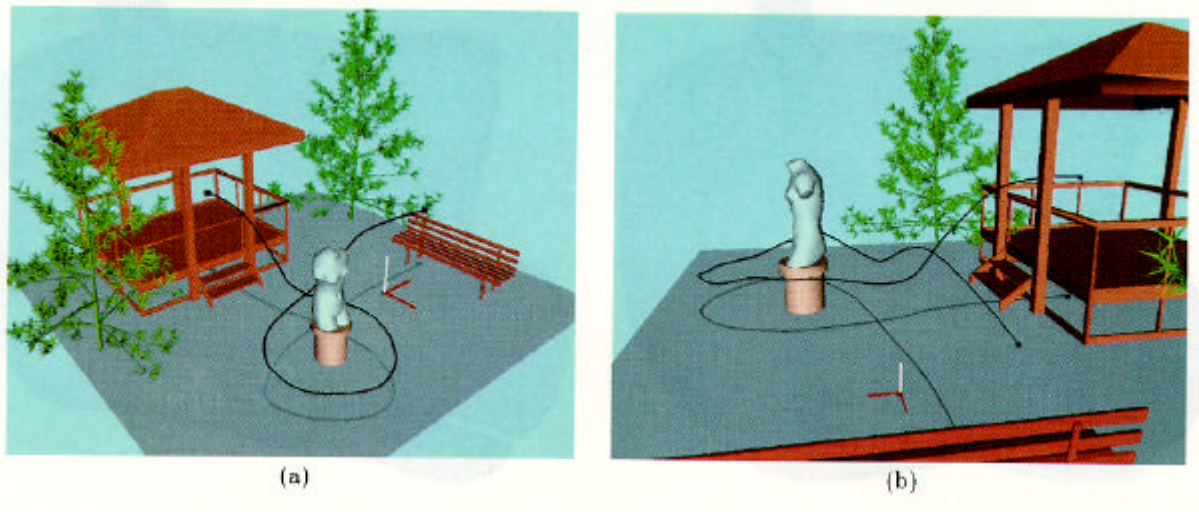

Figure 16: The user has sketched a camera path through this virtual environment. The curve was created from the viewpoint, in [a]. Figure [b] shows the scene from a different viewpoint.

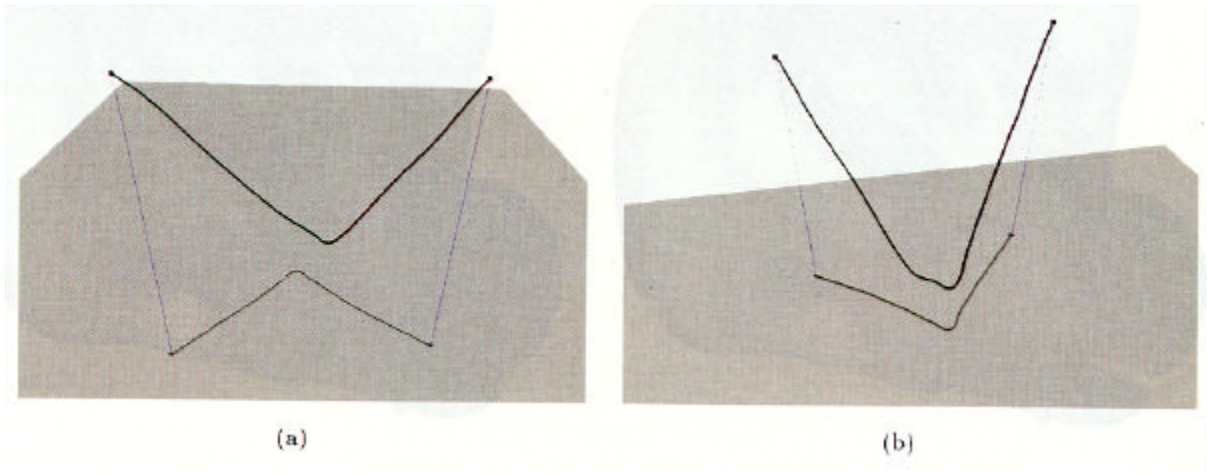

Figure 17: In color plate [a] we have intentionally misaligned the bends in the shadow and the curve. Note in figure [b] how the shape of the 3D curve has two bends while the sketched curve and shadow each have one.