Drawing Interwoven Surfaces

Keith Wiley and Lance Williams

We have created a system called *Druid* which uses a novel approach to representing interwoven surfaces in a $2\frac{1}{2}$D scene. Our system does not rely on a layered representation, and therefore does not suffer from the limitations imposed by layering. *Druid* maintains depth information only as a local property of segments of surface boundaries without requiring global depth information for each surface. The effect is that it is very easy to create drawings of interwoven surfaces with *Druid* while this is very difficult to achieve with other drawing programs at the present time.

I. INTRODUCTION

Classic drawing programs generally rely on technology dating back to the early 1980s. Many features in contemporary drawing programs have their origin in Sutherland's seminal PhD thesis on computer-assisted drawing (Sutherland [1]). Consequently, most drawing programs use a directed acyclic graph to represent surface depths in a $2\frac{1}{2}$D scene. We believe this unnecessarily restricts the space of possible drawings. Specifically, it precludes drawings of surfaces which interweave. An example of a drawing that is difficult to create with such systems is a pair of interlocking rings. More complex patterns include Celtic knots. Our system, *Druid*, allows a user to create and edit such drawings quickly and easily.

II. $2\frac{1}{2}$D DRAWING

Drawing programs allow a user to create and edit a set of objects that are represented either by basic shapes (ellipses and polygons) or by complex curves (splines). Unlike painting programs which store information as a bitmap, drawing programs store objects as abstract representations which can be translated into images depicting the objects, i.e., the representations can be *lrendered* into final images. Consequently, objects in drawing programs can easily be edited after they have been created.

Most drawing programs at the present time represent the depths of objects in a series of layers, one layer per object. The ordering of the layers comprises a directed acyclic graph on the object depths. Consequently, no two objects can both occlude different regions of each other at the same time.

Layering is a fairly simple system to implement, which explains its ubiquity. However, complicated brittle spoofs are required to achieve rendered images that depict interwoven surfaces. A spoof is a visual illusion that creates the appearance of interweaving where, in actuality, the underlying representation has no such concept.

III. THE LABELING SCHEME

Williams [2] devised the labeling scheme used in *Druid*. *Druid* represents two-dimensional surfaces by their closed one-dimensional boundaries. The interiors of surfaces are not relevant to *Druid's* function until the final rendering stage. *Druid* treats the one-dimensional boundaries as a *labeled knot-diagram*. In a *labeled knot-diagram*, boundaries have a In a *sign of occlusion* corresponding to a direction of travel along the boundary such that the bounded surface lies to the right of the boundary. Additionally, boundary segments are assigned a nonnegative *depth index*. *Druid* imposes a set of depth constraints on the ways in which two one-dimensional boundaries can cross each other. Fig. 1 shows these constraints. For a drawing to be considered legal all of its intersections must honor the labeling scheme.

In Fig. 1 and Fig. 2 the arrow along a boundary denotes its sign of occlusion. In Fig. 1 there is no arrow shown for the lower boundary because it does not affect the labeling of that intersection.

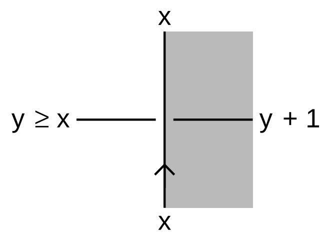

FIG. 1: The labeling scheme: The opposing top segments must have the same depth while the opposing bottom segments must differ in depth by exactly one. Additionally, the unoccluded segment of the bottom segment must be no shallower than the top segments.

As a user creates and edits boundaries for various surfaces in a scene, boundary intersections will be created and destroyed. *Druid* detects these changes on the fly and imposes the labeling scheme on all intersections in the drawing such that the drawing remains legal at all times.

IV. BOUNDARY GROUPING

It is helpful to group boundaries that are components of a single surface. This prevents the ambiguity that would otherwise result as to whether a hole lies beneath, above, or part of the same surface as a surrounding solid boundary. We find groups by searching for cuts between pairs of boundaries. A cut is analogous to a scissor-cut through a surface. The discovery of a legal cut between two boundaries effectively joins the two boundaries into a single boundary (Fig. 2). Represented as a single boundary resulting from a cut, the two original boundaries will then constrain the drawing to labelings in which they bound the same surface.

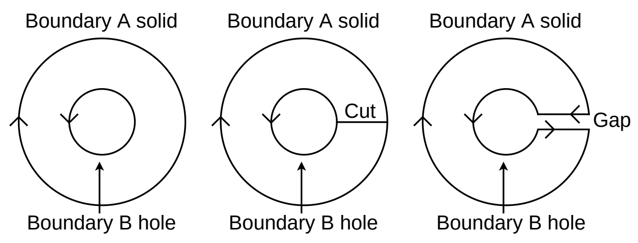

FIG. 2: A cut converts two boundaries into one boundary

V. FINDING A LEGAL LABELING

Given an unlabeled knot-diagram, there is a space consisting of all combinations of possible crossing-states, signs of occlusion, and depth indices. Whenever a change occurs to the topology of the knot-diagram *Druid* must find a new legal labeling as quickly as possible. Furthermore, it is preferable that *Druid* find a new legal labeling that preserves as much state as possible from the prior labeling that existed before the change occurred. We call this the *minimum difference solution*. This is desirable because the user generally wants components of the drawing that are not invalidated by the change to remain unaltered until the user explicitly designates such a change.

Druid's ability to automatically find a new legal solution after a user-interaction alleviates much of the burden on the user. For example, when a user wishes to flip the depth ordering of two overlapping surface regions, the user simply clicks with the mouse on any single intersection associated with the overlapping region in question. It is possible that many other intersections will have to be flipped as well in order to create a legal labeling that corresponds to the desired change, but *Druid* discovers the necessary changes on its own.

Druid's primary challenge is to find the minimum difference labeling in an acceptable time period. For a decent user-experience we would like to achieve turn-around times under a second if possible. This can be difficult because the search spaces for moderately sized drawings can be extremely large.

VI. INTERSECTION PROJECTION

A topology change occurs whenever intersections are created, destroyed, or change their relative positions on a boundary. Topology changes can occur due to a user interaction such as a move, reshape, or deletion of a boundary component. *Druid* needs to find the minimum difference labeling when such a change occurs. Therefore it is crucial to preserve the crossing-states of all existing intersections as the drawing changes. This precludes the simple solution of recomputing intersections every time a topological change occurs.

The better solution, the one adopted here, is to project an intersection from its old position to its new position as a function of the interaction between two boundaries that are being deformed. *Druid* represents boundaries as piecewise linear approximations of B-spline curves. The projection of an intersection is accomplished by determining the traversal of an intersection from one straight line segment to another around a given boundary. The final location of the intersection is determined by finding the pair of straight line boundary segments that intersect after the change has occurred.

VII. FINAL RENDERING OF A DRAWING

To produce an opaque rendering of a drawing *Druid* must determine which surface is at depth zero for each region of the knot-diagram. The region can then be defined as a polygon with vertices at the intersections that bound the region. The depth zero surface's color can be used to fill the region. To produce a transparent rendering, the process is simply extended to find all the surfaces which cover a region and their depth ordering for the region. A transparent color model can then be applied to calculate a single final color for the region. Fig. 3 show some examples of what *Druid* enables a user to create.

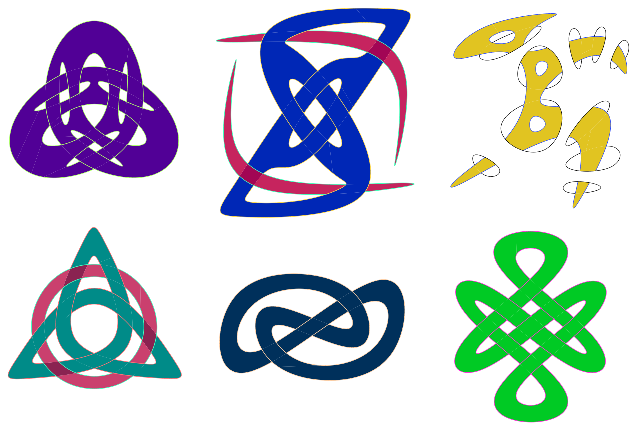

FIG. 3: Some final renderings created with *Druid*

VIII. CONCLUSIONS

We have devised a completely new approach for drawing programs. We do not store global depths for surfaces in a scene, but rather assign local depths to segments of one-dimensional boundaries for each two-dimensional surface and maintain a set of constraints on the ways in which these depths relate to each other at the intersections in the drawing. We have also created an intuitive interface for interacting with such a system that makes creating and editing interweaving drawings fast, easy, and fun.

IX. REFERENCES

[1] Ivan E. Sutherland. Sketchpad: A man-machine graphical communication system. Technical report, Univ. of Cambridge, Cambridge CB3 0FD, UK, September 2003. This technical report is a modern republication of Sutherland's 1963 doctoral dissertation.

[2] Lance R. Williams. *Perceptual Completion of Occluded Surfaces*. PhD thesis, Univ. of Massachusetts at Amherst, Amherst, MA, 1994.