

Harold: A World Made of Drawings

Jonathan M. Cohen and John F. Hughes and Robert C. Zeleznik*

Department of Computer Science
Box 1910 Brown University, Providence, RI 02912

Abstract

The problem of interactively creating 3D scenes from 2D input is a compelling one, and recent progress has been exciting. We present our system, *Harold*, which combines ideas from existing techniques and introduces new concepts to make an interactive system for creating 3D worlds. The interface paradigm in *Harold* is *drawing*: all objects are created simply by drawing them with a 2D input device. Most of the 3D objects in *Harold* are collections of planar strokes that are reoriented in a view-dependent way as the camera moves through the world. Virtual worlds created in *Harold* are rendered with a stroke-based system so that a world will maintain a hand-drawn appearance as the user navigates through it. *Harold* is not suitable for representing certain classes of 3D objects, especially geometrically regular or extremely asymmetric objects. However, *Harold* supports a large enough class of objects that a user can rapidly create expressive and visually rich 3D worlds.

CR Categories and Subject Descriptors: I.3.2 [Graphics Systems]: Stand-alone Systems; I.3.4 [Graphics]: Paint Systems, Utilities, Picture Description Languages; I.3.5 [Computational Geometry and Object Modeling]: Curve, Surface, and Solid Object Representations; I.3.6 [Computer Graphics]: Methodology and Techniques

Additional Key Words: Stroke-based rendering, gestural interfaces, billboards, scene description

1 Introduction

In the children's book *Harold and the Purple Crayon* [11], a small boy, Harold, creates a world by drawing it with his purple crayon. He explores and expands this world by walking into and through it, drawing all the time. In essence, this is the ultimate virtual environment, allowing users both to experience and to create a virtual world around them. An ideal VR system would be capable of this much expressiveness and interactivity, but this problem is, at present, intractable. We therefore state a more restricted form of this problem: Given a scene drawn in 2D from a single point of view, we would like to reconstruct the scene interactively from novel viewpoints. In other words, we would like to be able to draw a scene, move around it, and have everything just "look right."

Previous research has approached this problem in two primary

*{jmc,jfh,bcz}@cs.brown.edu

ways, which we categorize as *geometric* and *image-based*. In the geometric approach, the system attempts to create a geometric description of the 3D scene from the user's 2D input. This is similar to many problems in computer vision, and is essentially the inverse of traditional computer graphics – given a rendering (or often a drawing), the system tries to recreate a geometric description of the scene. New renderings can then be obtained from arbitrary viewpoints. The Sketch [19] and Teddy [10] systems have demonstrated the feasibility of this approach for interactively creating 3D objects from 2D gestures.

A limitation of Sketch and Teddy, however, is that the inferred geometry is often incorrect, and these errors become more and more apparent as the viewpoint changes significantly from that from which the object was initially created. This reflects a fundamental drawback of purely geometric approaches – not all 2D drawings can actually be generated from 3D models. Rademacher [15] discusses how this problem arises when animators attempt to create static 3D models of cartoon characters, and proposes using dynamic view-dependent geometry to address it.

Image-based approaches avoid creating a geometric description of the scene, but instead redisplay the original input image, modified to reflect new camera parameters. The system described in [18] lets the user draw on the inside of a sphere, thus allowing an immersive experience as long as the camera's position remains fixed. In *Tour Into the Picture* [6], the user begins with a 2D image, which may be either a photograph or hand-drawn, and then specifies geometric constraints such as the vanishing point and horizon line. The user can then view the scene from novel camera locations within certain constraints (the user cannot, for example, turn around and look behind herself).

Our approach attempts to find a middle ground. Our system, *Harold*, like Sketch and Teddy, creates a 3D model of the environment. However, our world is populated by drawings, not 3D objects, and thus is similar to image-based methods, particularly [18]. The primary geometric primitive in our system is a *billboard*; these are commonly used in interactive systems to render complex yet unimportant objects with low overhead. A billboard is typically a plane with an image texture-mapped onto it that rotates about some point or axis to face the viewer as much as possible. Our billboards contain collections of planar strokes rather than textures. When the user draws a stroke over a billboard, we simply project the stroke onto the billboard and store it; then, to display the billboard, we re-render each stroke, rotated appropriately (see Figure 1). Thus, we avoid the problem of reconstructing what the backside of a tree looks like – the tree simply has no back. A consequence of this choice is that *Harold*, unlike Sketch and Teddy, works with a very small set of inferences about the user's strokes: for the most part, strokes are simply projected onto a surface and nothing more.

As noted by Zeleznik et al. [19] and Igarashi et al. [10], it is important that views of the scene be rendered in a non-photorealistic style, in order better to convey the imprecise and hand-drawn nature of the underlying geometric description. We take this notion even further and allow the user to draw objects using a variety of stroke styles. With the exception of distance cueing, we render the

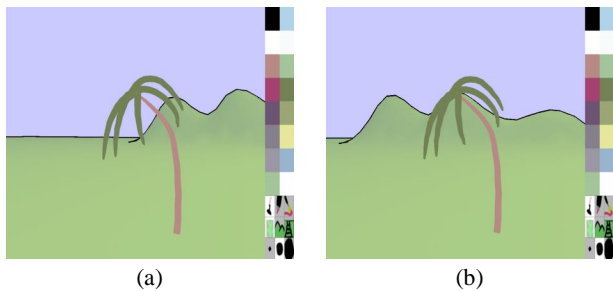


Figure 1 Creating a billboard. (a) The trunk of the tree was drawn as a stroke starting at the base. The leaves were then added to the billboard. (b) That same billboard, seen from a different view; the billboard on which the stroke was drawn rotates so as always to face the viewer as much as possible.

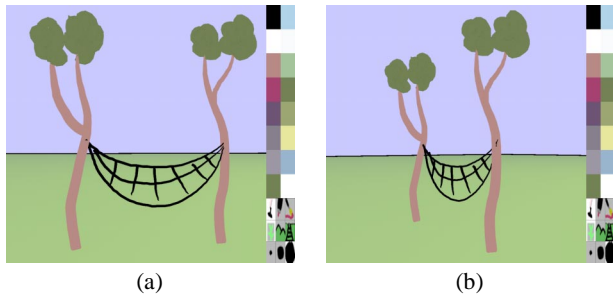


Figure 2 The hammock, which is a *bridge* anchored to the two tree billboards, was created from the viewpoint in (a); (b) shows the objects from a new viewpoint.

strokes exactly as they were drawn. Objects in Harold maintain the distinct stylistic appearance and subtleties imparted by the user, and our worlds thus maintain their intended style and character as the viewpoint changes.

The primary drawback of our approach is that the relationships between objects change as the viewpoint changes. For example, imagine a fenced-in area containing farm animals. If the fences were all billboards, they would rotate through one another as the viewpoint changes, unwittingly freeing the enclosed livestock. We thus need some way of specifying fixed relationships among objects in our world. For this purpose, we use a *bridge billboard*, which is a collection of planar strokes that is anchored to points on two billboards (see Figure 2). Thus, we can string a fence between separate fence posts, or a hammock between two trees. Harold also has a primitive terrain-sketching facility, with which the user can sketch a height-field terrain by drawing the silhouettes of hills or other features.

The entire Harold system is an amalgamation of simple components – there are no algorithmic subtleties, nor any complex constraint-maintenance mechanisms. Our contribution is thus an approach to 3D scene reconstruction that combines features of other approaches as well as novel ideas, and integrates them with a particular set of interface choices to create an interactive system.

2 Related work

Our system has a similar aesthetic to several 2D paint programs. We like the easy-to-use and appealing visual interface of Kid Pix [5], which is a paint program designed for children. In Kid Pix, all operations are easy to find and have immediately observable consequences. Because this type of interface places a low cognitive load

on the user, it encourages experimentation and exploration. Our system is similar to the AltaMira and Quantel Paintbox systems ([16] and [14]), although these systems are not fully 3D, in that they employ a similar notion of creating scenes by layering images on top of one another.

Interactively constructing 3D scenes from 2D input is an active research area. The Sketch and Teddy systems ([19] and [10]), as well as our previous work in sketching 3D curves [2], allow the user to create 3D objects via simple suggestive gestures. In Teddy, for example, the user draws the silhouette of an object, and the system creates a plausible 3D shape with that silhouette. Previous work by Tolba et al. [18] is very similar in spirit and technique to Harold. In their system, user’s strokes are projected onto a sphere centered at the camera. We have in fact incorporated this technique into Harold to let the user to draw objects in the sky. Tour Into the Picture [6] allows the user to “enter” a 2D image by warping the image to simulate new viewing parameters.

We use the stroke-based rendering system described in [13], which is related to the skeletal strokes technique described in [8, 7]. The ground in Harold is rendered using the shader described by Gooch et al. [4], which provides shape information while preserving a brightly colored hand-drawn appearance.

3 The user’s view of the system

We now describe how the system appears to the user. The initial view of the world shows a “ground,” which is a large planar region of the xz -plane, and a “sky,” which is the inside of an enormous sphere.

The interaction metaphor in Harold is *drawing*. With the exception of clicking on toolbar icons to change color, stroke style, stroke width, and drawing mode, all operations are invoked either by clicking on an object or by drawing a stroke. There are three buttons: the drawing button, the camera button, and the eraser button.¹ Figure 5 summarizes the gestures, key-mappings, modes, and their meanings.

The user places her cursor at some point of the screen; this point corresponds to a point of the world, either on the ground or in the sky. (Henceforth, we merely say that she places her cursor “on a point in the world,” glossing over the correspondence induced by tracing a ray from the viewpoint through the point on the film plane and into the world.) She now begins to draw by dragging the cursor with the drawing button depressed. The style of the resulting stroke depends on the current color, the current stroke width, and which of three rendering styles is selected from the menu.

STROKES ON THE SKY. Suppose that her starting point was in the sky. The strokes are then interpreted as “drawing on the sky” and the result is a stroke on the sky, visible whenever she looks in that direction. As mentioned above, this is an implementation of [18].

STROKES ON THE GROUND. Strokes that begin on the ground are interpreted in one of three ways, depending on the drawing mode currently selected on the toolbar: drawing on the ground, drawing billboards, and drawing terrain.

Drawing on the ground. In drawing-on-the-ground mode, a stroke that starts on the ground is treated as laying down marks on the ground itself. This is useful for creating things like train tracks, sidewalks, etc. If the user’s stroke crosses over a silhouette of the terrain, the projection of the stroke onto the ground terrain will be discontinuous. We make the projected stroke continuous by bridg-

¹Our implementation uses left button for drawing, right button for camera controls, and shift-left button for erasing.

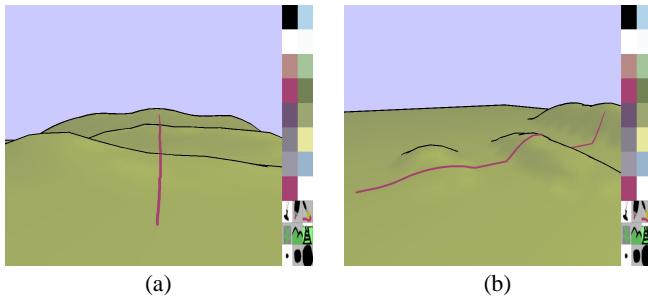


Figure 3 Ground strokes. (a) A ground stroke crossing the silhouette of the hill on the left. (b) The same stroke seen from a different view, showing how the system has filled in the “gap” with a segment.

ing any gaps with curves that look from above like straight line segments (see Figure 3). Since our terrain is a heightfield, for every x and z coordinate, there is a unique y coordinate. We can thus fill in all gaps with a line segment on the xz -plane, and then compute the height at each point along the line segment to create a 3D stroke that lies on the ground.

Drawing billboards. In billboard-drawing mode, a stroke that starts on the ground creates a new *billboard*, anchored at the starting point of the stroke. The billboard’s plane is perpendicular to the ground, and as perpendicular as possible to the eye-to-stroke-start-point vector. Thus the billboard plane’s normal vector, when the billboard is created, is

$$n = \mathbf{y} \times (\mathbf{y} \times (\mathbf{eye} - \mathbf{base})),$$

where **eye** is the camera location, **base** is the location of the base of the billboard and **y** is the vertical unit vector in the world.

After each stroke is added to a billboard, a bounding rectangle for the strokes on that billboard is created, slightly enlarged, and displayed in a semi-transparent light gray. Any subsequent stroke that starts in this highlighted area is added to this billboard, regardless of the current drawing mode. This is important in creating billboards having several strokes that do not actually touch.

The interpretation of strokes has one small subtlety: the final meaning of a stroke is determined at the end of the stroke, when the mouse is released. If the mouse-up occurs *outside* of the highlighted area, and over a *different* billboard, then the stroke is interpreted as defining a *bridge billboard* between these two billboards, or simply a “bridge.”

Drawing bridge billboards. A bridge is created, as just described, by a stroke that starts on one billboard (we’ll call the start point S) and ends on a stroke in another billboard (we’ll call the end point E). Both S and E are taken to be points *on their respective billboards*, not in the world; when the billboards turn to face the camera, the world locations of S and E change as well. When a bridge is created, the system determines the plane that contains the current world-space positions of E , S , and the \mathbf{y} vector. At creation time, this bridge billboard is highlighted, and subsequent strokes drawn on it are recorded just like those on any other billboard.

When the camera is moved, the world-space locations of E and S may change. When this happens, the bridge billboard is scaled in its horizontal direction and sheared along the vertical direction to maintain the correspondence of points of the billboard with the points E and S (see Section 4.3). This helps to maintain the apparent “connections” between strokes in the bridge billboard and strokes in the billboards at either end.

Drawing terrain. In terrain-drawing mode, a stroke that starts and

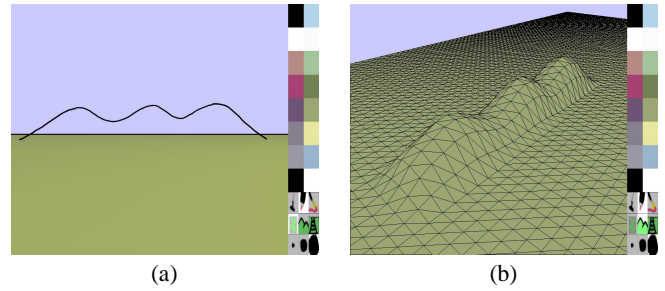


Figure 4 Editing terrain. (a) The user draws a stroke starting on the ground that indicates the shape of a hill; (b) The hill is created by warping the ground to try to match the stroke.

ends on the ground creates a bump in an attempt to make the stroke a silhouette of the newly deformed ground (see Figure 4). All objects in the world are then lifted so that they remain coincident with the ground.

OTHER GESTURES. A few other gestures can be made with the drawing button. A single click anywhere in the scene un-highlights the highlighted billboard. A single click on a billboard stroke highlights the billboard that contains that stroke. Colors can be drag-and-dropped from the color palette onto the sky, the ground, or any stroke, to change their colors accordingly.

ERASER GESTURES. Clicking on a stroke with the eraser button removes it. Scribbling on a billboard with the eraser button depressed removes the entire billboard.

CAMERA CONTROL. We wanted a driving-style interface for controlling the camera location in Harold. We rejected World-In-Hand controls because we wanted to give the user a sense of the size of the world around her. We wanted a technique that would blend aesthetically with the stroke drawing nature of Harold. Igarashi’s technique [9] was appropriate because of its path-drawing interface; however, we adapted this technique so that the user could explicitly specify a point in the scene at which the camera would look, and we set the camera always to be two units above the ground (so that one unit is approximately equal to one meter).

To move the camera, the user draws a stroke on the ground with the camera button depressed. This path is displayed as a red line, and the gaps in the stroke that the user can’t see are filled in as described above. The user then clicks on a point on the world. The camera moves along the drawn path at a constant speed (5 meters per second) and ends up looking at the point where the user clicked. A click on the sky while the camera is traveling along a path cancels the current path.

To swivel and tilt the camera, the user clicks on the sky with the camera button and “drags” the sky side-to-side to swivel, or up and down to tilt. We constrain the camera only to tilt 30 degrees up or down from the horizontal plane.

We also provide Doom-style camera controls [17] via the numeric keypad, so the user can optionally fly through the world. This particular choice of controls breaks the drawing metaphor, but is a simple way to give the user more freedom of movement in the virtual world.

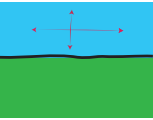


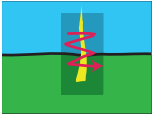




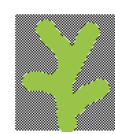


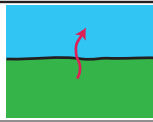
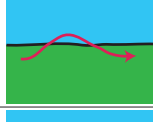
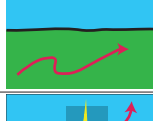



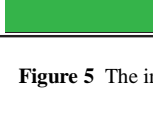
	Button/Gesture	Meaning	Button/Gesture	Meaning
Camera Button		Side to side: rotate Up and down: tilt		Specify camera path for motion
		Delete stroke		Delete billboard
Widgets (with Drawing Button)		Select stroke color		Drag color onto stroke (or other object)
		Select stroke style (ink brush, marker, or watercolor)		Select stroke width (small, medium, large)
Gestures with Drawing Button				
	Mode choice			
	Gesture	Billboard mode	Terrain mode	Ground mode
		Create billboard	Ignored	Draw on ground: ignore parts of stroke on sky
		Create billboard	Create hill	Draw on ground: ignore parts of stroke on sky
		Create billboard	Create hill	Draw on ground
		Extend billboard with new stroke	Extend billboard with new stroke	Extend billboard with new stroke
		Create bridge	Create bridge	Create bridge
		De-select billboard	De-select billboard	De-select billboard
		Draw on the sky	Draw on the sky	Draw on the sky

Figure 5 The interface components: red arcs are strokes, red dots are clicks.

4 Implementation details

4.1 Storing collections of strokes

All strokes are stored with an associated color, style, and width. The width can be in either pixels or world-space units. Strokes on billboard and bridges are stored in world-space width, while strokes in the sky and on the ground are stored in pixel width. If the width is in world-space units, we first compute the corresponding width in pixels that an object would have at that distance from the camera and then render the stroke. Thus strokes change their widths to indicate distance from the camera. We clamp all pixel widths so strokes are at least three pixels wide.

Since our toolbar indicates stroke widths by circles of different pixel radii, one issue that arises when storing strokes with world-space width is what does it mean to select a particular stroke width. We decided that selecting a stroke width in pixels should indicate the width at which a particular stroke would be rendered if it were at the same distance from the camera as the filmplane. Thus, when the user has selected a particular stroke width, strokes drawn with that width may have different pixel widths depending on their distances from the camera.

The collection of strokes associated with a billboard or a bridge is represented as a list of polylines in the coordinate space of the billboard, which is a copy of \mathfrak{R}^2 . The coordinates are then mapped into the world via the transformations described in the following two sections.

4.2 Billboard transformations

Whenever the camera moves, each billboard must transform itself to face the camera. While a number of possible transformations would achieve this affect, we chose a relatively simple transformation in which each billboard rotates around a single point fixed at the base of the billboard. To minimize the artifacts that occur when rotating a billboard, we determine the point of rotation by finding where the billboard's strokes touch the ground. This rotation point is updated every time a stroke is added to or removed from the billboard.

Since some billboards have more than one point where a stroke touches the ground and others have none, our algorithm for choosing a rotation point searches the collection of stroke polylines on the billboard for all points with a locally lowest y-value. We next find all such minima that are within a small distance of the ground (we use 0.5 meters), and of these determine L and R as the leftmost and rightmost (i.e., the points with the highest and lowest x -value in the coordinate space of the billboard). We choose the midpoint of L and R for the center of rotation. When no strokes have points near the ground, we simply choose as the center of rotation the lowest point over all strokes on the billboard.

Each billboard is represented internally as a copy of \mathfrak{R}^2 together with a basepoint b in \mathfrak{R}^3 ; for any viewpoint v , we build a rigid transformation that maps the origin of \mathfrak{R}^2 to b and maps \mathfrak{R}^2 to a plane through b that contains \mathbf{y} and the vector $(v - b) \times \mathbf{y}$. If we use coordinates $[x, y, 1]^t$ for points of \mathfrak{R}^2 and coordinates $[x, y, z, 1]^t$ for points of \mathfrak{R}^3 , then our transformation is represented by the 4×3 matrix

$$M = \begin{bmatrix} \gamma & 0 & b_x \\ 0 & 1 & b_y \\ \zeta & 0 & b_z \\ 0 & 0 & 1 \end{bmatrix}$$

where

$$\gamma = (v - b)_x / \sqrt{(v - b)_x^2 + (v - b)_z^2}$$

$$\zeta = -(v - b)_z / \sqrt{(v - b)_x^2 + (v - b)_z^2},$$

which maps the origin to b , the unit \mathbf{y} -vector in \mathfrak{R}^2 to the unit \mathbf{y} -vector in \mathfrak{R}^3 , and the unit \mathbf{x} -vector to the horizontal unit vector in \mathfrak{R}^3 that is orthogonal to $v - b$.

4.3 Bridge transformations

A bridge is created with a pair of points S and E , each on a separate billboard. Let \hat{S} and \hat{E} denote the world-space locations of those points at the time the billboard is created. Let q denote the vector $\hat{E} - \hat{S}$ and $m = [q_x, 0, q_z, 0]$ denote its projection to the xz plane. Then we build a map from \mathfrak{R}^2 to \mathfrak{R}^3 defined by the transformation

$$N = \begin{bmatrix} \phi & 0 & \hat{S}_x \\ 0 & 1 & \hat{S}_y \\ \chi & 0 & \hat{S}_z \\ 0 & 0 & 1 \end{bmatrix}$$

where

$$\phi = q_x / \|m\| \text{ and } \chi = q_z / \|m\|,$$

which sends the origin $\mathbf{0}$ to \hat{S} , the unit \mathbf{y} -vector of \mathfrak{R}^2 to the unit \mathbf{y} -vector of \mathfrak{R}^3 , and the unit \mathbf{x} -vector in \mathfrak{R}^2 to the unit vector in the direction m . The preimage of E under this transformation is some point e of \mathfrak{R}^2 . Since the preimage of S is the origin and the preimage of E is e , the preimage of $q = S - E$ is $e - \mathbf{0}$; thus $N(e - \mathbf{0}) = q$.

To store a stroke made on a bridge billboard, we project each vertex onto the bridge's plane to get a point in \mathfrak{R}^3 . This point is then transformed by N^{-1} and recorded in \mathfrak{R}^2 ; when the bridge is to be redisplayed from some other view in which the world-space locations of E and S are now \bar{E} and \bar{S} , we build a new transformation \bar{N} that maps the origin to \bar{S} , the \mathbf{y} -axis of \mathfrak{R}^2 to the \mathbf{y} -axis of \mathfrak{R}^3 , and the point e to the point \bar{E} (or equivalently, maps the vector $e - \mathbf{0}$ to the vector $\bar{q} = \bar{S} - \bar{E}$). Letting $\bar{m} = [\bar{q}_x, 0, \bar{q}_z, 0]^t$ be the projection of the vector \bar{q} to the xz -plane, the matrix for this transformation is

$$\bar{N} = \begin{bmatrix} \bar{\phi} & 0 & \bar{S}_x \\ 0 & 1 & \bar{S}_y \\ \bar{\chi} & 0 & \bar{S}_z \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ \frac{\bar{q}_y}{q_y \| \bar{m} \|} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{\| \bar{m} \|}{\| m \|} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where

$$\bar{\phi} = \bar{q}_x / \| \bar{m} \| \text{ and } \bar{\chi} = \bar{q}_z / \| \bar{m} \|.$$

Reading right to left, the first matrix scales the domain in x so that after being transformed, the vertical line that contains e will map to the vertical line containing \bar{E} . The second matrix is a shear such that the vertical line containing the origin is unchanged and the vertical line containing e is raised so that e maps to \bar{E} . The third matrix is simply a rigid transformation like the one built for N above.

4.4 Rendering strokes

As described above, each stroke is defined by a sequence of points (a polyline) and a pixel width that is either stored or computed. This sequence of points is then rendered as a stroke using the method of Northrup and Markosian [13]. The three stroke styles we have implemented provide a fair range of expressiveness while maintaining interactive frame rates. Examples of these stroke types are shown in Figure 6.

For the marker style, we use a stroke with constant width and no mitering at the endpoints. For the ink style, we use a stroke with mitered endpoints and a width that tapers to 0. Since the width is tapered over the length of the stroke, the stroke gets wider near its beginning as its length increases. In our interactive system, this

gives the effect of ink “bleeding” and spreading outward as the user continues to draw a stroke.

For watercolor strokes, the stroke is drawn with increasing transparency along its length. We linearly fade out the transparency to 0.5 along the first 20 vertices of the stroke, and draw all subsequent vertices with 0.5 transparency. This implementation captures only a tiny fraction of true watercolor behavior (as modeled by Curtis et al. [3]), but is efficient enough for rendering in our real-time system.

Strokes are rendered by first building triangle strips, and then passing them to the rendering pipeline using OpenGL [1]. Since strokes are view-dependent, we cannot just cache them in display lists; they must be rebuilt for each frame. However, the user typically maintains a fixed camera position while drawing an object. To get a higher sampling rate for the input device in this case and improve the interactivity of the system, we can take advantage of display lists and cache the triangle strips that have already been built.

4.5 Terrain

Terrain-editing strokes must start and end on the ground. Call the starting and ending points S and E . Just as in the bridge-definition rules in sections 4.3, these two points, together with the y -vector, determine a plane in \mathbb{R}^3 , that we call the *projection plane*. The points of the terrain-editing stroke are projected onto this plane (this projection, which is a curve in \mathbb{R}^3 , is called the *silhouette curve*); the shadow of the resulting curve (as cast by a sun directly overhead) is a path on the ground (we call this the *shadow*). Points near the shadow have their elevation altered by a rule: each point P near the shadow computes its new height (y -value) as a convex combination

$$(1 - w(d)) \cdot P_y + w(d) \cdot h$$

where d is the distance from P to the projection plane, h is the y -value of the silhouette curve over the nearest point on the projection plane to P , and $w(d)$ is a weighting function given by

$$w(d) = \max \left(0, 1 - \left(\frac{d}{5} \right)^2 \right).$$

This gives a parabolic cross-section of width 10 for a curve drawn over level terrain. Other choices for w would yield hills with different shapes that might be more intuitive, but this particular choice gives reasonable results in most cases.

Note that if the silhouette curve bends back on itself (i.e. it defines a silhouette that cannot be modeled using a heightfield), then the variation of height along the shadow will be discontinuous. The resulting terrain then may have unexpected features.

5 Limitations and discussion

Figures 7 and 8 show two scenes created using Harold. Both scenes took under 15 minutes to draw. Figures 9 and 10 show how Harold can be used for conceptual prototyping of outdoor scenes. Figure 11 shows a simple moon scene.

The major problem that arises in trying to reconstruct a 3D scene from 2D input is trying to determine what an object looks like from a new point of view. One can think of billboards as providing a crude solution to this problem – we approximate the appearance of an object from a new viewpoint as its appearance from the initial viewpoint. Thus billboards work well for objects that look approximately the same from all directions, e.g., trees, flowers, telephone poles, and even roughly drawn characters. When this assumption of approximate radial symmetry does not hold, as in the case of a house, say, billboards look odd and visually disturbing.

A different approach may have more success in cases like this. The advantage of a system like Teddy, for example, is that for certain types of objects, Teddy can produce fairly accurate approximations of the objects’ appearances from novel viewpoints. Another strategy is to constrain the camera so that the user views objects only from viewpoints from which the system can plausibly reconstruct an object’s appearance. This is the approach taken in Tolba, Dorsey, and McMillan’s system [18], and one we have incorporated to a certain extent in Harold. This is why in Harold the camera is constrained to stay at a constant height over the ground (with the exception of the optional flying controls); we have made the system’s task slightly easier since we don’t have to reconstruct the appearances of objects from above or below.

Another drawback of billboards is that they do not maintain a fixed relationship with one another as the camera moves. For example, when two large billboards are close together, they may intersect each other as the viewer moves and the billboards rotate. This can produce surprising visual effects. Our algorithm for choosing the center of rotation partially addresses this problem, as does the use of bridges, but there are some cases that neither approach handles particularly well.

Also, the interface in Harold is more obviously modal than systems like Sketch or Teddy. It is not clear whether this is a drawback or not. The notion that modes are “suggestive” rather than rigid may be somewhat awkward to some users as well: one can be in billboard mode and still perform operations that are not related to billboards, such as drawing on the sky.

6 Future work

An obvious direction for future work is to incorporate more techniques from other systems such as Sketch or Teddy, or our 3D curve system. It is not clear how to do this in many cases, and it is clearly a nontrivial task. However, such a system could potentially be a very powerful modeling tool for creating visually rich 3D environments.

Although we have three types of brushes, it is clear that a richer mechanism for creating drawings would be useful. All of the features of ordinary paint programs could conceivably be incorporated, although certain issues arise in the context of our system that do not arise in 2D paint programs. For example, flood-filling a collection of strokes on a billboard is not as simple as flood-filling a region of a 2D image because there may be gaps between strokes that are not visible if the billboard is sufficiently far away. A naive flood fill would then unexpectedly flow outside of the intended area. These are interesting problems to investigate.

Other directions we are exploring include extending the drawing metaphor to allow the user to “draw” animations, such as a river that appears to be flowing or rain that appears to be falling. It is conceivable that somehow we could take this even further and allow the user to “draw” simple behaviors, thereby creating interactive 3D environments.

Finally, there are many possible extensions to the rendering system, such as adding automatic level-of-detail for strokes that are far away, improvements to the overall efficiency of the system, and incorporating other non-photorealistic rendering styles such as those described in [12].

7 Acknowledgements

Thanks to the artists and art students who used Harold and gave us feedback, especially Michael Legrand and Noah Rafor. Thanks also to Lee Markosian and J.D. Northrup for providing the stroke-

based rendering system. Finally, thanks to the entire graphics group for their support. This work is supported in part by the NSF STC for Computer Graphics and Scientific Visualization, Adobe, Advanced Network and Services, Alias/Wavefront, Department of Energy, IBM, Intel, Microsoft, National Tele-Immersion Initiative, Sun Microsystems, and TACO.

References

- [1] “OpenGL Architecture Review Board”. *OpenGL Reference Manual, 2nd Edition*. Addison-Wesley Developer’s Press, 1996.
- [2] Jonathan M. Cohen, Lee Markosian, Robert C. Zeleznik, John F. Hughes, and Ronen Barzel. An interface for sketching 3d curves. In *1999 Symposium on Interactive 3D Graphics*, pages 17–21. ACM SIGGRAPH, April 1999.
- [3] Cassidy J. Curtis, Sean E. Anderson, Joshua E. Seims, Kurt W. Fleischer, and David H. Salesin. Computer-generated watercolor. In *SIGGRAPH 97 Conference Proceedings*, pages 421–430. ACM SIGGRAPH, August 1997.
- [4] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A non-photorealistic lighting model for automatic technical illustration. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, pages 447–452. ACM SIGGRAPH, July 1998.
- [5] Craig Hickman. Kid pix: The early years. <http://pixelpoppin.com/kidpix/KPHistory/>.
- [6] Youichi Horry, Ken ichi Anjyo, and Kiyoshi Arai. Tour into the picture: Using a spidery mesh interface to make animation from a single image. In *SIGGRAPH 97 Conference Proceedings*, pages 225–232. ACM SIGGRAPH, August 1997.
- [7] Siu Chi Hsu and Irene H. Lee. Drawing and animation using skeletal strokes. In *SIGGRAPH 94 Conference Proceedings*, pages 109–118. ACM SIGGRAPH, July 1994.
- [8] Siu Chi Hsu, Irene H. Lee, and N. E. Wisema. Skeletal strokes. In *Proceedings of UIST 93*, pages 197–206. ACM SIGCHI, November 1993.
- [9] Takeo Igarashi, Reiko Kadobayahi, Kenji Mase, and Hiehiko Tanaka. Path drawing for 3d walkthrough. In *Proceedings of UIST 98*. ACM SIGCHI, November 1998.
- [10] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3d freeform design. In *SIGGRAPH 99 Conference Proceedings*, pages 409–416. ACM SIGGRAPH, August 1999.
- [11] Crockett Johnson. *Harold and the Purple Crayon*. Harper-Collins Juvenile Books, 1977.
- [12] Michael A. Kowalski, Lee Markosian, J.D. Northrup, Lubomir Bourdev, Ronen Barzel, Loring S. Holden, and John F. Hughes. Art-based rendering of fur, grass, and trees. In *SIGGRAPH 99 Conference Proceedings*, pages 433–438. ACM SIGGRAPH, August 1999.
- [13] J. D. Northrup and Lee Markosian. Artistic silhouettes: A hybrid approach. In *Non-photorealistic Animation and Rendering*. ACM SIGGRAPH, June 2000.
- [14] Quantel. Quantel paint box. <http://www.quantel.com>.
- [15] Paul Rademacher. View-dependent geometry. In *SIGGRAPH 99 Conference Proceedings*, pages 439–446. ACM SIGGRAPH, August 1999.
- [16] Alvy Ray Smith. Varieties of digital painting. ftp://ftp.alvyray.com/PstScrtpt/8_Paint.ps, 1995.
- [17] Id Software. Doom. Computer game, 1993.
- [18] Osama Tolba, Julie Dorsey, and Leonard McMillan. Sketching with projective 2d strokes. In *Proceedings of UIST 99*. ACM SIGCHI, November 1999.
- [19] 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.



Figure 6 The top strokes are marker style, the middle strokes are ink style, and the bottom strokes are watercolor style.



Figure 9 Here Harold was used to create an initial conceptual sketch of an outdoor scene. The scene was created by an architectural student within an hour of first using Harold.

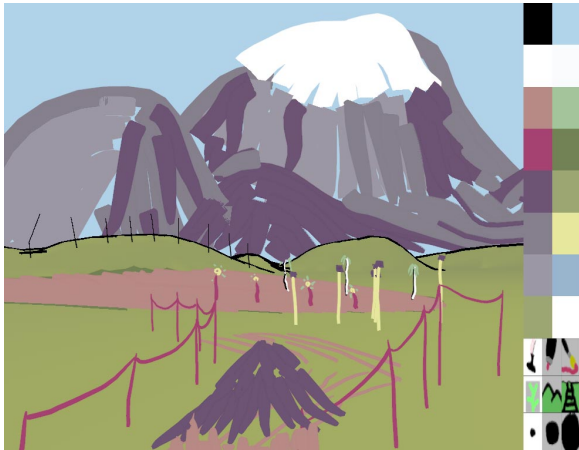


Figure 7 This scene was created in Harold. The mountains are painted on the sky, the fences are bridges strung between billboards, and the hut in the lower center is a billboard.

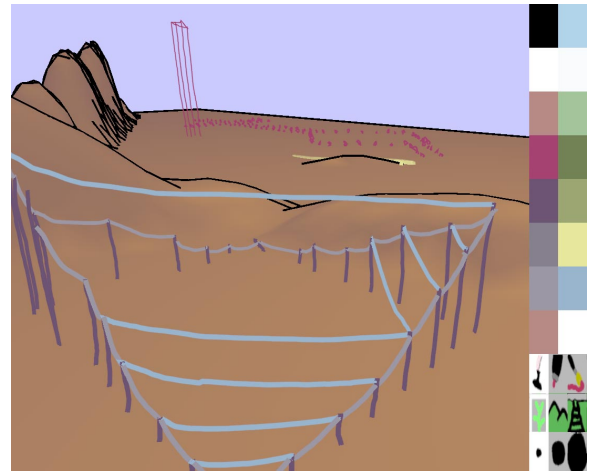


Figure 10 The same scene from a different viewpoint.

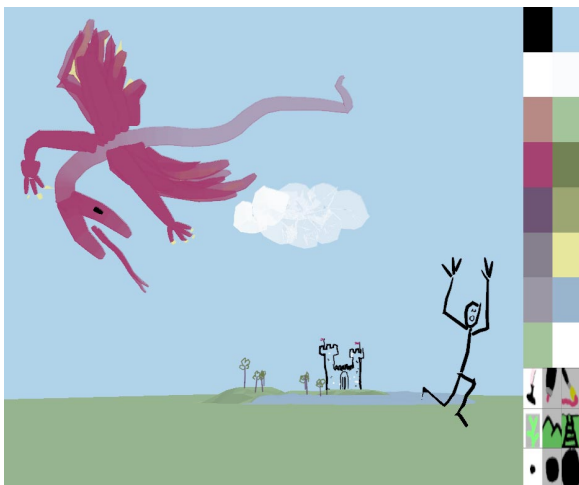


Figure 8 This scene was created by an art student after using Harold for approximately 1 hour.

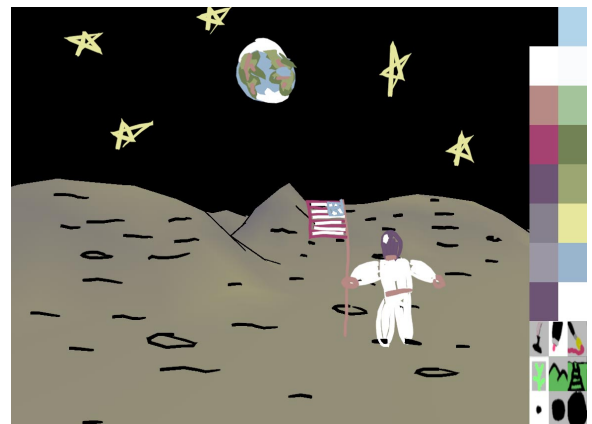


Figure 11 A moon scene created in Harold. The bright colors and different stroke styles enable Harold to capture the liveliness of a child's drawing.