A Projective Drawing System

Osama Tolba Julie Dorsey Leonard McMillan

Laboratory for Computer Science
Massachusetts Institute of Technology, Cambridge, MA 02139
E-mail:{tolba,dorsey,mcmillan}@graphics.lcs.mit.edu

Abstract

We present a novel drawing system for composing and rendering perspective scenes. Our approach uses a projective 2D representation for primitives rather than a conventional 3D description. This allows drawings to be composed with the same ease as traditional illustrations, while providing many of the advantages of a 3D model. We describe a range of user-interface tools and interaction techniques that give our system its 3D-like capabilities. We provide vanishing point guides and perspective grids to aid in drawing freehand strokes and composing perspective scenes. Our system also has tools for intuitive navigation of a virtual camera, as well as methods for manipulating drawn primitives so that they appear to undergo 3D translations and rotations. We also support automatic shading of primitives using either realistic or non-photorealistic styles. Our system supports drawing and shading of extrusion surfaces with automatic hidden surface removal and highlighted silhouettes. Casting shadows from an infinite light source is also possible with minimal user intervention.

CR Categories: I.3.3 [Computer Graphics]: Graphics Utilities—Graphics Editors; I.3.6 [Computer Graphics]: Methodologies and Techniques—Interaction Techniques

Keywords: Image-based Modeling and Rendering, Misc. 2D graphics, Non-Euclidean Spaces, Non-Photorealistic Rendering

1 Introduction

The advent of computer graphics has greatly influenced many aspects of architectural drafting. Construction drawings, in the form of plans, sections, elevations, and details, are seldom drawn by hand today. In addition, hand-crafted physical models, traditionally used for client presentations, have been largely replaced with three-dimensional computer models and walk-throughs. Perspective drawing, which was once an important technique for exploring and presenting designs, is virtually obsolete due to the speed and flexibility of today's CAD systems. However, 3D modeling systems are generally cumbersome to use, and ill-suited for the early stages of design where freehand sketching is often more appealing.

Traditional perspective drawings are difficult to construct [3]. Only a skilled illustrator can make a drawing with correct proportions. Furthermore, many construction lines are required to achieve this proportionality, making the process laborious. In addition, the views they depict are static, which reduces their 3D impression. Proper shadow construction and shading are also time-consuming. Finally, like all drawings on paper, they are difficult to edit or reuse.

Our goal is to provide interactive techniques to support perspective drawing. This problem has been largely neglected in 2D computer graphics. Almost all current 2D graphics systems use drawing primitives that are represented with *Euclidean* 2D points. The process of constructing a perspective drawing with these systems is nearly as tedious as with traditional media.

We have developed a perspective drawing system that overcomes many of the limitations of traditional perspective drawing and current 2D computer graphics systems. Our system retains the ease-of-use of a 2D drawing systems, but its projective representation provides additional 3D-like functionality. We intend this tool for applications that often do not require actual 3D modeling, such as conceptual design, technical illustration, graphic design, and architectural rendering. In many cases, these applications strive to generate a single perspective view, or a set of views sharing a common viewpoint.

We use *projective* 2D points to compose various renderings of a scene and provide capabilities that are generally thought to require 3D models. For example, our system supports perspective drawing guides, dynamic 3D-like viewing and object manipulation, scene illumination and shading, and automatic shadow construction. In addition, shape modeling operations, such as the simulation of 3D extrusion, can also be performed using projective 2D points.

Our 2D representation does not allow us to simulate walk-throughs. While we are able to simulate single-object motion, it is not possible to correctly simulate the motion of a group of objects (or the viewer). We use transformations of the image of a planar object that are independent of its actual distance from the viewer. In order to transform images of multiple objects, however, we need relative depth information, which our system does not support.

1.1 Related Work

In [12] we introduced a projective 2D point representation and demonstrated that perspective scenes, drawn with freehand strokes, can be properly re-projected into new viewing directions with a variable field of view, thereby giving the drawing an immersive 3D-like effect. In this work, we extend the projective 2D point representation to provide sophisticated tools for constructing perspective geometric shapes. The shapes can be shaded and made to cast shadows. We also provide tools to simulate the apparent 3D motion of these shapes within a scene.

Projective representations underly all panoramic image-based rendering (IBR) systems. For example, “QuickTime VR” represents environments with cylindrical panoramas and synthesizes novel perspective views by providing an interface for panning, tilt-

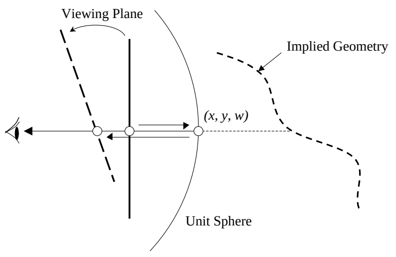


Figure 1: Two-dimensional drawing points are stored on the surface of the unit sphere centered about the viewer. They are displayed by projecting them onto a user-specified viewing plane.

ing, and zooming [1]. IBR systems typically facilitate *navigation* and visualization of a static scene. In our approach, we provide controls for *composing* and *editing* illustrations.

An active area of research is the development of sketching interfaces for 3D modeling [2, 6, 13]. These approaches acknowledge the difficulty of using standard interfaces to build 3D models. Their main premise is that 3D shape can be inferred from freehand strokes that follow a certain syntax, thereby allowing models to be generated very quickly. In our work, we do not infer 3D geometry, rather we make 2D drawings that are *3D-like*.

Non-photorealistic rendering (NPR) techniques apply a "hand-drawn" look to photographs and 3D renderings by simulating many conventional artistic methods. For example, when mimicking pen-and-ink styles, NPR uses hatching or stippling (a collection of short strokes) as a means to convey tonal variation [8, 10]. Another NPR technique is the use of silhouettes to emphasize shape [9]. Our work adopts silhouetting and selected pen-and-ink styles for rendering shaded perspective drawings automatically, although the actual rendering style is not the focus of our work.

2 Projective 2D points

In traditional drawing programs, primitives are specified via a collection of 2D points. Generally, these points are described by two coordinates, which can be imagined to lie on a plane. The coordinates specify the position of a point relative to a specified origin and two perpendicular basis vectors. In mathematical parlance, such points are considered Euclidean 2D points.

This Euclidean representation of points is practically universal to all 2D drawing systems. There are, however, alternative representations of 2D points, which are not only more powerful than Euclidean points, but also contain them as a subset. In particular, the set of projective 2D points can be represented using three coordinates in conjunction with the following rules: the origin is excluded, and all points of the form (a, b, c) and $\lambda(a, b, c)$, where λ is non-zero, are equivalent. The subset of projective points for which a value of λ can be chosen, such that $\lambda(a, b, c) = (\lambda a, \lambda b, 1)$, is the Euclidean subset.

There are several possible mental models for projective 2D points, which are comparable to the plane of the Euclidean points. We adopt a model in which all projective points lie on a unit sphere. Thus, the preferred representation of the point (a, b, c) is the one with λ chosen such that $a^2 + b^2 + c^2 = 1$. We will further restrict all values of λ to be strictly positive. This additional restriction results in a special set of projective points called the *oriented* projective set [11].

One advantage of projective 2D points is the ease with which they can be manipulated. Unlike Euclidean points, translations of projective points can be described by matrix products, thus allowing them to be composed with other matrix products, such as scaling and rotation. Projective points also permit re-projection to be described as a simple matrix product. Another advantage is that points at infinity are treated as regular points. For example, in a Euclidean system the intersection of two parallel lines must be treated as a special case, while using projective points it is treated as a regular vanishing point. These properties of projective point representations give unique capabilities to our two-dimensional drawing system.

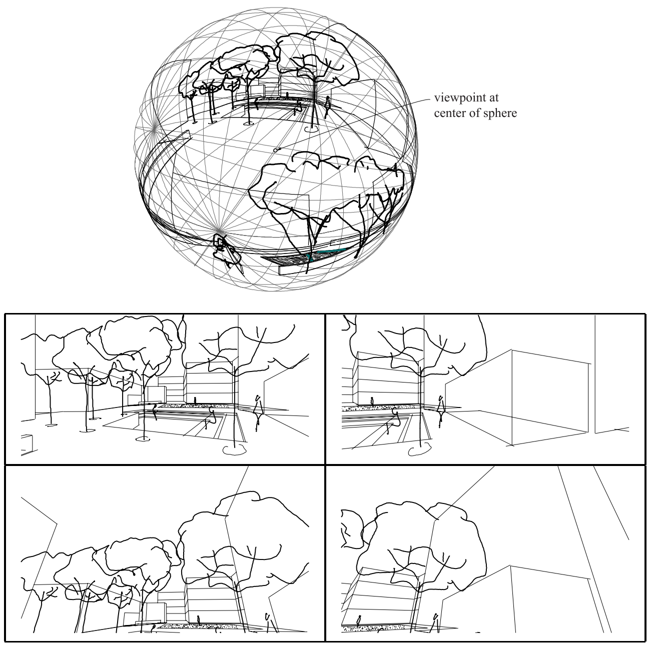


Figure 2: A drawing of an outdoor plaza design shown as points on the unit sphere centered about the viewer, and an array of views generated with our system from the same drawing. The bottom row views look in the same directions as the top row but tilt up.

Each stroke (or shape) in our system is stored as a list of such projective points obtained by back-projecting drawn image points to lie on the surface of a unit sphere centered about a viewpoint, while assuming that the drawing window subtends some solid angle viewing port. The stroke also supports auxiliary attributes such as pen color and thickness. A drawing is a collection of strokes and shape primitives. Our projective representation allows us to generate novel re-projections of the drawing (see Figure 1). These re-projections can be interpreted as rotating and zooming a camera about a single point in a three-dimensional space. However, re-projections of projective 2D points do not exhibit parallax changes resulting from changes in viewing position.

We also use projective points to represent directions such as vanishing points, motion trajectories, infinite light sources, and surface normals. We refer the reader to [7] for a review of projective 2D points and fundamental computational tools.

3 The Perspective Drawing System

We start this section with descriptions of our system's perspective viewing, guides, and primitives. Then we provide a detailed explanation of the aforementioned 3D-like object manipulation.

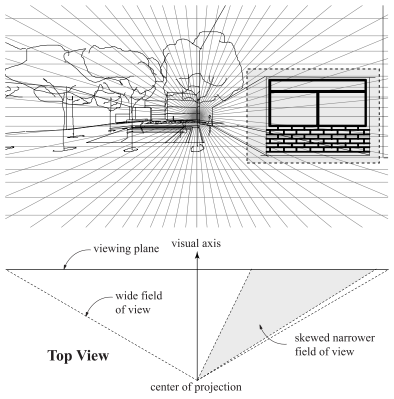


Figure 3: The drawing system supports skewed perspective frustums that enable the user to draw on a frontal plane, such as in the windows and bricks (shown with darker lines) drawn while facing the wall containing them. Without a skewed frustum this wall would either lie outside the picture or appear small within a wide field of view.

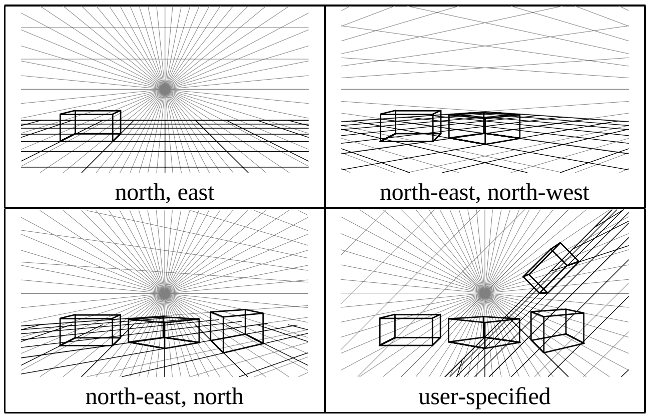


Figure 4: We provide flexible vanishing points and perspective grids as visual guides. Users may select from built-in direction or specify arbitrary ones. The "rectangle" tool respects the current vanishing points as well as the normal to the plane they define.

3.1 Perspective Viewing

In [12] we described how the re-projection of points (see Figure 2) allows for a virtual camera interface that provides instant panning, tilting, and zooming, similar to the QuickTime VR interface [1]. In addition to rotation and zooming, we provide controls for moving the image center, thereby allowing the user to work on parts of the drawing that would otherwise lie outside the field of view, or be too small if the field of view were made very wide. For example, when a user orients the view such that a chosen plane is viewed frontally (i.e. parallel to the viewing plane), the plane may be located outside the field of view. The user may then use the "image center" tool to bring the plane into the picture in order to draw "on it" proportionately. Useful examples include drawing bricks and windows on a facade (see Figure 3). In computer graphics terminology, this operation yields a skewed perspective frustum.

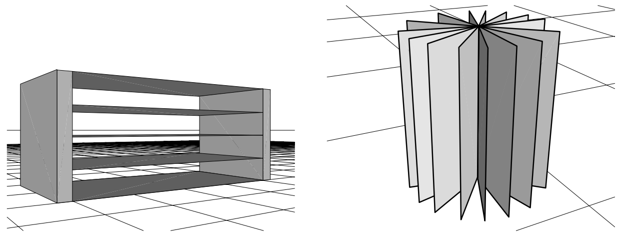


Figure 5: Examples showing the emulation of 3D translation (left) and rotation (right).

3.2 Perspective Guides

The use of a projective representation provides two new types of visual guides beyond the rulers and regular grids used by traditional 2D drawing systems: vanishing point guides and perspective grids.

Vanishing points are traditionally used as directional guides as well as a means for geometric construction. We maintain them in our system for use as guides when drawing lines and rectangles. We also use vanishing points throughout this paper to compute various directions and object points.

Our drawing system supports all of the conventional perspective view categories, such as "single-point," "two-point," and "three-point" perspective, since the viewing direction can be changed dynamically, thereby transforming single-point perspective into two- or three-point perspective, and vice-versa. In fact, vanishing points can be specified in arbitrary directions, which need not even be orthogonal (see Figure 4).

Vanishing points are directions represented by points on the unit sphere. They can be visualized as poles of a sphere centered about the viewer. When projected onto the viewing plane, the longitudinal lines of the sphere appear as straight lines converging at the vanishing point, providing the desired visual effect.

We also provide perspective grids in our system. The system automatically adjusts the grids to align with any two of the currently active vanishing points. Grids, like vanishing points, can lie in general positions. This provides the interface necessary for drawing views containing parallel lines, rectangles, boxes, etc. (see Figure 4).

3.3 Perspective Shape Primitives

In addition to basic freehand drawing and straight lines, we support higher level shape primitives such as "perspective rectangles" (images of 3D rectangles), which the user specifies with two corner points. General-shape closed polygons are also supported. Such primitives can have solid color for depicting non-transparent objects. When these primitives overlap, the order in which they are drawn is used to convey occlusion. As with current 2D drawing programs, the user can adjust this stacking order at any time.

3.4 Perspective Shape Manipulation

The drawing system supports manipulations of shape primitives that appear as 3D rigid-body translations and 3D rotations. These manipulations provide flexibility and, together with copy/paste operations, they facilitate the creation of scenes containing symmetric or repeating elements (see Figure 5).

In order to carry out these operations we use projective 2D points and surface normals, which the system automatically infers from user input. For example, the surface normal for a 3D rectangle viewed in perspective is simply the cross product of its two vanishing points (see Figure 6). In addition to their use in shape manipu-

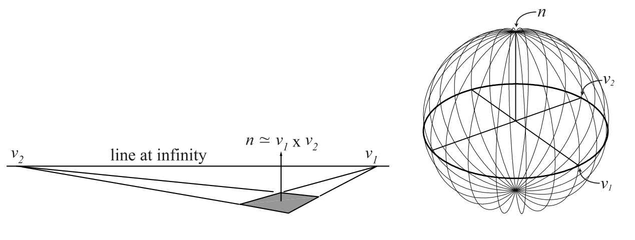


Figure 6: Surface normals are inferred by computing the vector cross product of any two vanishing points associated with the 3D plane. The line joining the two vanishing points is the image of the "line at infinity" of that plane, along which *all* of its vanishing points must lie. It can be envisioned in the spherical model as a great circle perpendicular to the normal direction.

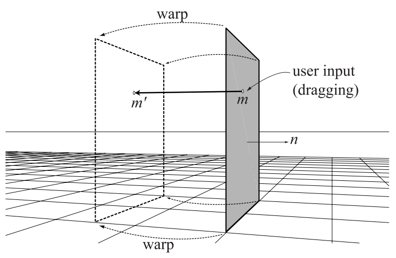


Figure 7: Apparent translation of the plane as it is carried out in our system: Image points are transformed (warped) using a homography that is inferred from two input points (m, m') and the surface normal. The motion trajectory is selected by the user from a list of active vanishing points.

lation, surface normals are also crucial for performing other operations, such as modeling of aggregate shapes, shading, and shadow projection.

In projective geometry, each point has a dual—a line whose coefficients equal the point's coordinates. The dual to a point representing the normal to a 3D plane plays a special role. Suppose the vector (a, b, c) represents the normal to a 3D plane. Its dual is the projective 2D line $ax + by + cw = 0$. We can envision this line as an equatorial great circle whose pole is (a, b, c). Clearly, points on this circle represent *all* directions parallel to the 3D plane, representing ideal (rather than Euclidean) 3D points at infinity. Thus, the line dual to the plane's normal is the image of the *line at infinity* associated with the plane, of which we make frequent use. For example, we arrive at the direction of a 3D line that lies in a given 3D plane by computing the "vanishing point" at the intersection of the line's 2D image with the image of the plane's line at infinity.

3.4.1 Apparent translation

We provide an intuitive user interface for simulating 3D translations of planar objects. The user selects a vanishing point as the motion trajectory (direction of translation) then uses a pointing device to "drag" the object along this trajectory. Note that this operation requires no knowledge of distance or depth of the object, as may initially be imagined. It is possible to carry out this transformation on the image of a plane knowing only its surface normal.

We use mappings of the projective plane, represented by the unit sphere, to accomplish this transformation. Such a mapping is often referred to as a *collineation* or *homography* H. It can be thought of as a warping function applied to any object's points through multiplying them by a 3×3 matrix as follows:

$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} = \lambda \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ w \end{pmatrix},$$

$$\text{or} \qquad \mathbf{m}' \simeq H\mathbf{m}, \tag{1}$$

where $\mathbf{a} \simeq \mathbf{b}$ denotes $\mathbf{a} = \lambda\mathbf{b}$, and λ is an arbitrary scale factor.

As mentioned in [7], the image of a translating 3D plane is transformed by the following homography:

$$H \simeq I + \frac{\delta}{d}\mathbf{t}\mathbf{n}^T,$$

where I is the 3×3 identity matrix, $\mathbf{t}$ is the motion trajectory, δ is the translation distance, and $\mathbf{n}$ is the surface normal of the moving plane, whose initial equation in 3D space is $\mathbf{n} \cdot \mathbf{P} = d$.

Since in a 2D projective setting we have no knowledge of the distance d from the viewpoint to the surface or the actual displacement of the plane δ, we deduce a new quantity $\alpha = \delta/d$. This yields a single-parameter family of homographies compatible with the translation of a 3D plane:

$$T(\alpha) \simeq I + \alpha\mathbf{t}\mathbf{n}^T. \tag{2}$$

All the quantities on the right-hand side of Equation 2 are known except for the scalar parameter α, which can be inferred from a single pair of points $(\mathbf{m}, \mathbf{m}')$ given the location of a point on the surface before and after translation. Such a pair can be specified using the pointing device, and must be constrained to lie on the selected trajectory, hence the single degree of freedom (see Figure 7). We determine α as follows:

$$\alpha = \pm\frac{\|\mathbf{m}' - \lambda\mathbf{m}\|}{\lambda(\mathbf{n}\cdot\mathbf{m})}, \qquad sign(\alpha) = sign(\mathbf{t}\cdot(\mathbf{m}' - \lambda\mathbf{m})). \tag{3}$$

The value of λ is given in the following derivation. From Equations 1 and 2 we get:

$$\mathbf{m}' = \lambda(I + \alpha\mathbf{t}\mathbf{n}^T)\mathbf{m} = \lambda\mathbf{m} + \lambda\alpha\mathbf{t}(\mathbf{n}\cdot\mathbf{m}), \tag{4}$$

where λ is a scale factor that we must determine before we can solve for α. First, we eliminate α by taking the cross product of Equation 4 with $\mathbf{t}$:

$$\mathbf{m}' \times \mathbf{t} = \lambda(\mathbf{m} \times \mathbf{t}).$$

This is an equation of the form: $\mathbf{a} = \lambda\mathbf{b}$ (vector $\mathbf{a}$ is λ-times vector $\mathbf{b}$), the solution for which is:

$$\lambda = \pm\frac{\|\mathbf{m}' \times \mathbf{t}\|}{\|\mathbf{m} \times \mathbf{t}\|}, \qquad sign(\lambda) = sign((\mathbf{m}' \times \mathbf{t})\cdot(\mathbf{m} \times \mathbf{t})).$$

In our application, λ is always positive. We then rewrite Equation 4 as: $\mathbf{m}' - \lambda\mathbf{m} = \alpha\lambda(\mathbf{n}\cdot\mathbf{m})\mathbf{t}$, and solve for α as shown in Equation 3.

Note that $\mathbf{n}\cdot\mathbf{m} = 0$ in the denominator of Equation 3 means that if the plane passes through the origin ($d = 0$), or is viewed "edge-on," the image of the planar shape is reduced to a line. In this case we cannot use a homography to simulate 3D motion.

3.4.2 Apparent rotation

As with apparent translation, the apparent rotation of a 2D perspective primitive about a fixed point, or pivot, can be simulated using

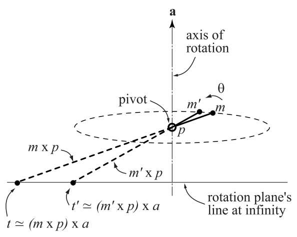


Figure 8: The rotation angle θ is inferred from a pair of input points (m, m') indicating the position of a point before and after rotation. The point rotates in a plane perpendicular to the rotation axis. By extending the lines mp and $m'p$ to the rotation plane's line at infinity we get the directions t and t', which completely specify the rotation angle.

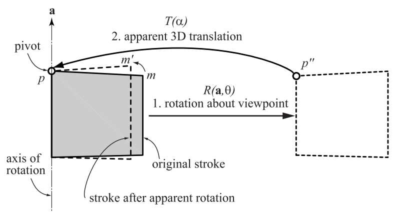

a
T(α)
2. apparent 3D translation
pivot
p
m'
m
p''
R(a,θ)
1. rotation about viewpoint
axis of rotation
original stroke
stroke after apparent rotation

Figure 9: Apparent rotation of the plane is carried out in two steps: In the first step, the plane is rotated about the viewpoint. Then, using an apparent 3D translation, it is moved back to the pivot point.

homographies. For example, a perspective rectangle can appear to revolve about an arbitrary axis passing through a user-selected pivot.

An intuitive user interface allows the user to specify the rotation parameters. First, the user selects the axis from a list of active directions (vanishing points) and uses the pointing device to specify the pivot. Then the rotation angle is specified by dragging the pointing device about the pivot. For proper visual feedback, we make the pointing device appear to orbit in a 3D plane perpendicular to the rotation axis and passing through the 3D pivot. Therefore, we infer the rotation angle from the change in the *direction* of the imaginary 3D line joining the pivot, whose image is $\mathbf{p}$, and the pointing device, represented by $\mathbf{m}$ and $\mathbf{m}'$ (see Figure 8):

$$\theta = \pm sin^{-1}(\|\mathbf{t} \times \mathbf{t}'\|), \quad sign(\theta) = sign((\mathbf{t} \times \mathbf{t}') \cdot \mathbf{a}),$$

where $\mathbf{t}$ and $\mathbf{t}'$ represent the above-mentioned direction before and after the rotation, given by:

$$\mathbf{t} \simeq (\mathbf{m} \times \mathbf{p}) \times \mathbf{a}, \quad \mathbf{t}' \simeq (\mathbf{m}' \times \mathbf{p}) \times \mathbf{a}.$$

Once we have established the rotation axis, pivot and angle, we rotate the object in two conceptual steps (see Figure 9):

1. In the first step, we rotate the object about the viewpoint (at the origin of the world) using the rotation axis and angle desired for the local rotation. All object points, including the pivot itself, move to an intermediate position:

$$\mathbf{m}'' = R(\mathbf{a}, \theta)\mathbf{m}.$$

2. Next, we use apparent 3D translation (Equation 2), where $\mathbf{t} \simeq \mathbf{p} - \mathbf{p}''$, to "move the object back" to the original pivot:

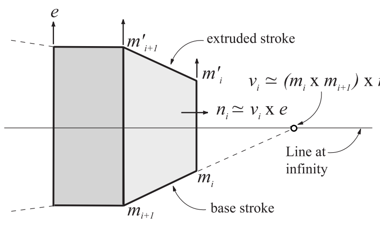


Figure 10: An extrusion shape is created by making a copy of the base stroke and transforming it via a pseudo-3D translation along the extrusion direction. The normal of each facet is computed by intersecting the line joining m_i and m_{i+1} with the base stroke's line at infinity in order to determine a vanishing point v_i, and then computing the normal as the cross product of this vanishing point with the extrusion trajectory.

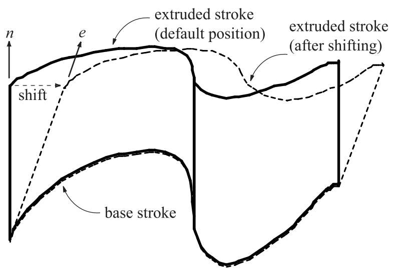

n
e
extruded stroke (default position)
extruded stroke (after shifting)
shift
base stroke

Figure 11: By default, the extrusion trajectory is perpendicular to the base stroke. However, skewness can be introduced by shifting the extruded stroke in any chosen direction.

$$\mathbf{m}' \simeq T(\alpha : \mathbf{p}'' \rightarrow \mathbf{p})\mathbf{m}''.$$

Thus, the desired apparent rotation homography is a composition of a 3D rotation matrix and a pseudo-3D translation homography:

$$\mathbf{m}' \simeq T(\alpha)R(\mathbf{a}, \theta)\mathbf{m}.$$

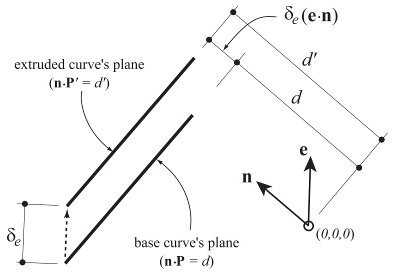
4 Aggregate Shapes

In this section we show how complex aggregate shapes can be modeled under this framework using shaded 2D polygons. We have implemented extrusion shapes as an example of such aggregate shapes. The principles described here are potentially applicable to other shapes as well, such as surfaces of revolution.

4.1 Extrusion

The user draws a freehand "base stroke" and selects the extrusion trajectory from the list of active vanishing points, then *drags* the pointing device to specify the magnitude of the extrusion. The system responds by making a copy of the base stroke, which we call the "extruded stroke," and applies apparent 3D translation to this copy using a variant of Equation 2:

$$T(\alpha_e) \simeq I + \alpha_e \mathbf{e}\mathbf{n}^T,$$



$\delta_e(\mathbf{e}\cdot\mathbf{n})$
extruded curve's plane
($\mathbf{n}\cdot\mathbf{P}' = d'$)
d'
d
$\mathbf{e}$
$\mathbf{n}$
δ_e
base curve's plane
($\mathbf{n}\cdot\mathbf{P} = d$)
(0,0,0)

Figure 12: Geometry of the 3D planes used in extrusion.

where α_e is inferred from the dragging action and $\mathbf{e}$ is the selected extrusion trajectory. Segments of the new extruded stroke are connected to corresponding ones in the base stroke, thereby forming the facets of the shape.

This approach assumes that the base stroke represents a planar curve in 3D space. By default, the user interface initially assigns the base stroke's normal as the extrusion direction. Normals to the facets are inferred from vanishing points (see Figure 10), allowing for shading and shadow projection. Later, the user may shift the extruded stroke in any direction, thereby simulating a skewed extrusion shape (see Figure 11). The extrusion direction is re-computed as the intersection of any two facet sides connecting the base and extruded stroke. Facet normals are also updated using this new extrusion direction.

4.1.1 Apparent Translation

We also provide the ability to move an extruded shape in the scene using the same interface that we described for planar objects (Section 3.4.1). The apparent collective motion of the aggregate shape is performed using two homographies: $T(\alpha)$ and $T(\alpha')$ for the base and extruded strokes respectively. We determine the base stroke's parameter α directly from user input as described in the planar object case, while the parameter α' for the extruded stroke can be determined as follows:

$$\alpha' = \frac{\alpha}{1 + \alpha_e(\mathbf{e}\cdot\mathbf{n})}.$$

Derivation: Suppose that the imaginary 3D curve represented by the base stroke lies in a 3D plane whose equation is $\mathbf{n}\cdot\mathbf{P} = d$, and the extruded curve lies in a parallel plane: $\mathbf{n}\cdot\mathbf{P}' = d'$. From Figure 12 we can deduce that:

$$d' = d + \delta_e(\mathbf{e}\cdot\mathbf{n}), \qquad (5)$$

where δ_e is the extrusion distance. From our definition of α in Section 3.4.1, we have:

$$\alpha_e = \frac{\delta_e}{d}, \quad \alpha = \frac{\delta_t}{d}, \quad \alpha' = \frac{\delta_t}{d'}, \qquad (6)$$

where δ_t is the translation distance. By substituting the value of d' from Equation 5 into 6, we have:

$$\alpha' = \frac{\delta_t}{d + \delta_e(\mathbf{e}\cdot\mathbf{n})}.$$

Since $\delta_e = d\alpha_e$ (Equation 6), we have:

$$\alpha' = \frac{\delta_t}{d + d\alpha_e(\mathbf{e}\cdot\mathbf{n})} = \frac{\delta_t/d}{1 + \alpha_e(\mathbf{e}\cdot\mathbf{n})} = \frac{\alpha}{1 + \alpha_e(\mathbf{e}\cdot\mathbf{n})}.$$

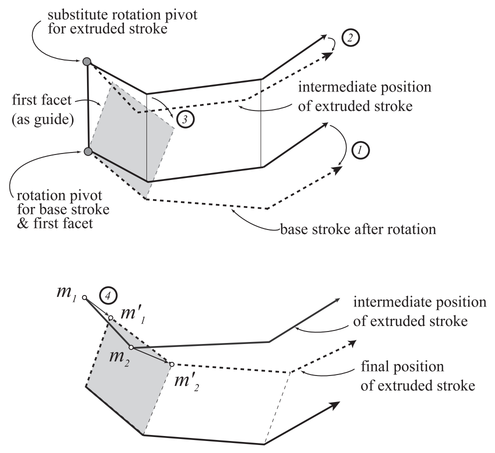


Figure 13: An extrusion shape is rotated in several steps: First, both the base and extruded strokes are rotated about their first points. The first facet is also rotated in order to determine the final position of the extruded stroke (top). Then the extruded stroke is moved to the correct position using the first facet as guide (bottom).

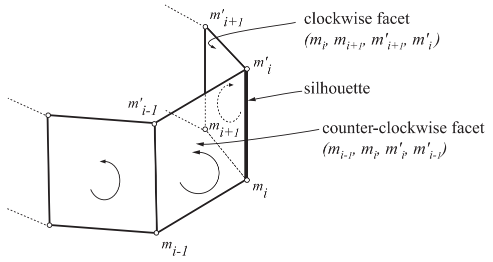


Figure 14: An edge of an extrusion shape is determined to be on the silhouette if its neighboring facets are drawn in opposite directions.

4.1.2 Apparent Rotation

Rotation of an extruded shape about a pivot is also possible. Any point could serve as the origin of the rotation. For simplicity, we choose the pivot to be the first point in the base stroke. We perform the rotation in a series of steps (see Figure 13):

1. Rotate the base stroke about the rotation pivot.
2. Rotate the extruded stroke about *its* first point. This results in an intermediate position for the extruded stroke.
3. Rotate the first facet of the shape about the rotation pivot in order to establish the correct positions for the first and second points in the extruded stroke.
4. Move the extruded stroke from its intermediate position to the correct position determined in step 3. For this operation, we use apparent 3D translation (Equation 2), where $\mathbf{t} \simeq (\mathbf{m_1} \times \mathbf{m_1'}) \times (\mathbf{m_2} \times \mathbf{m_2'})$.

4.2 Silhouettes

Rather than drawing all facets of an extrusion, and in keeping with the hand-drawn look, we have developed techniques to highlight the boundaries and silhouettes of extrusion shapes. Silhouettes of

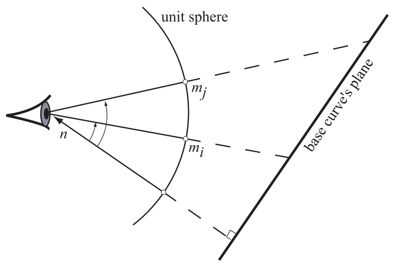


Figure 15: Points on a plane make a greater angle with its normal as they move farther away from the viewpoint. This observation is used to draw an extrusion shape in a back-to-front order.

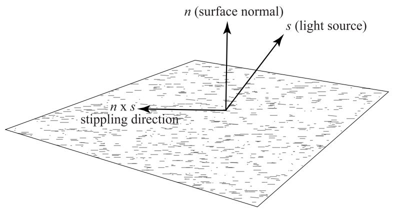


Figure 16: The stippling direction is determined from the surface normal and light direction.

faceted surfaces lie at the edges between two facets, one of which is front-facing while the other is back-facing [9]. A simple 2D method can be used to determine the existence of this condition [5]. If the edges of two neighboring facets are drawn in opposite directions (i.e., clockwise vs. counter-clockwise), the shared edge is on the silhouette (see Figure 14).

4.3 Visibility

Although inter-object visibility cannot be unambiguously resolved for 2D representations, intra-object visibility can be determined in some instances. For example, the facets of an extrusion shape can be drawn in a back-to-front order using the simple observation that points on the base plane make a greater angle with the plane's normal as they move farther away. For example, in Figure 15, we have $\mathbf{m}_j \cdot \mathbf{n} > \mathbf{m}_i \cdot \mathbf{n}$; therefore, a facet based at $\mathbf{m}_j$ is potentially occluded by another based at $\mathbf{m}_i$ (assuming the extrusion shape is not skew). Based on this dot product criteria, we create a sorted list of facet indexes that we use for rendering. Re-sorting of this list is necessary if the object undergoes apparent translation or rotation.

5 Shading

Our drawing system provides shading capabilities by inferring surface normals (see above) and allowing the user to insert infinite light sources into the scene. The picture can then be rendered with flat-shaded solid color (using any local lighting model) or with artistic styles such as stippling and hatching.

We have implemented a basic stippling algorithm (used in Figure 24-a) that employs short strokes whose direction is determined by the surface normal and light direction (see Figure 16). The density of the strokes is determined by a Lambertian shading computation, and their position and length are randomized in order to emulate a hand-drawn look (see code in Figure 17).

Another shading style that we support is a simple hatching method (used in Figure 24-b). This method generates a look that is consistent with that of the manual illustrations in [3]. Instead of Lambertian shading, it generates four levels of grey according to the following rules: Shadows are hatched with maximum density, objects facing away from the light are hatched with lighter density, and light stippling is applied to objects that are dimly lit (i.e., the angle between the normal and the light source is greater than 45 degrees).

Due to the computational overhead of artistic shading (about 2 seconds for a complex scene), we adopt the following strategy: Shading strokes are computed in screen coordinates when there is no camera motion, then back-projected and stored on the unit sphere. As the user rotates the camera, the stored strokes are used to render the scene. Although the stored shading becomes somewhat inaccurate during camera motion, this strategy provides adequate feedback during scene navigation and avoids the flickering that would result from re-computing the strokes during camera motion.

```
COMPUTE-STIPPLE
StippleDirection ← $\mathbf{n} \times \mathbf{s}$
Convert-StippleDirection-To-Screen-Coordinates
StippleDensity ← MaxDensity $\times(1 - min(0, \mathbf{n} \cdot \mathbf{s}))$
NumStipples ← StippleDensity × Bounding-Box-Area
for $i$ ← 1 to NumStipples
    do BeginPoint ← Random-Point-Inside-Bounding-Box
        EndPoint ← BeginPoint + (Random-Length × StippleDirection)
        Clip-Stipple-to-Shape; Back-Project-Stipple; Add-to-StippleList
```

Figure 17: Pseudo-code for stippling algorithm.

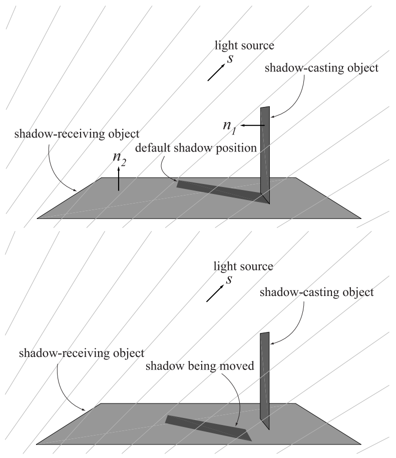


Figure 18: Shadows are projected automatically using the surface normals and light source direction (shown with faded lines). The shadow is initially attached to the object casting the shadow (top). Later the user may *drag* the shadow in order to simulate distance between the shadow-casting and shadow-receiving objects (bottom). The system automatically re-projects the shadow during this dragging operation.

6 Shadows

Shadows play an important role in scene presentation due to their effectiveness in conveying shape and relative position information. Following classical line construction techniques, we have implemented an automatic algorithm that computes the shape of an object's shadow as cast from an infinite (directional) light source like the sun. However, due to the lack of depth information, the shadow

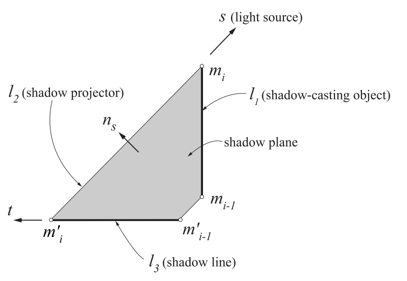


Figure 19: The shadow of a stroke is determined by marching along the stroke and projecting the shadow segments iteratively. m'_i is determined from m'_{i-1} by intersecting l_2 and l_3, where l_2 is the shadow projector and l_3 is the shadow line. l_3 is found by intersecting an imaginary shadow plane with the shadow-receiving object.

```
PROJECT-SHADOW
$\mathbf{n}_1 \leftarrow$ shadow-casting-object-normal
$\mathbf{n}_2 \leftarrow$ shadow-receiving-object-normal
$\mathbf{m}'_1 \leftarrow \mathbf{m}_1$                          ▷ Shadow attached to first point.
$k \leftarrow length[stroke]$                        ▷ Number of points in stroke
for $i \leftarrow 2$ to $k$
    do $\mathbf{l}_1 \leftarrow \mathbf{m}_{i-1} \times \mathbf{m}_i$          ▷ Shadow-casting stroke line.
       $\mathbf{l}_2 \leftarrow \mathbf{s} \times \mathbf{m}_i$                 ▷ Shadow projector.
       $\mathbf{v} \leftarrow \mathbf{l}_1 \times \mathbf{n}_1$                 ▷ Vanishing point.
       $\mathbf{n}_s \leftarrow \mathbf{s} \times \mathbf{v}$                  ▷ Shadow plane's normal.
       $\mathbf{t} \leftarrow \mathbf{n}_s \times \mathbf{n}_2$                 ▷ Intersection of 2 planes.
       $\mathbf{l}_3 \leftarrow \mathbf{m}'_{i-1} \times \mathbf{t}$             ▷ Shadow line.
       $\mathbf{m}'_i \leftarrow \mathbf{l}_2 \times \mathbf{l}_3$               ▷ Shadow point.
```

Figure 20: Pseudo-code for shadow projection.

is initially attached to the object casting the shadow, then the user may *drag* it to the desired position (see Figure 18). This *dragging* operation is achieved with our "apparent 3D translation" method by using the light's direction as the translation trajectory. Later, if the user re-positions the light source, the new position of the shadow is recomputed automatically, without any further user intervention. Note that, by using this shadow construction interface, it is possible to construct a scene with incomplete or even inconsistent shadows. It is the artist's responsibility to maintain the scene's integrity.

The information that is needed to compute the shadow is the surface normals for both the object casting the shadow and the one receiving it. The shadow of a stroke (or polygon) is determined by marching along the stroke and projecting its successive segments onto the shadow-receiving object. The first shadow point is attached to the corresponding point in the stroke. Thereafter, each shadow point is determined by intersecting a *shadow line* with a *shadow projector*—a line joining the light source and the shadow-casting point (see Figure 19). The trajectory of the shadow line is determined by intersecting an imaginary *shadow plane* with the shadow-receiving object. All these operations are performed in two dimensions using vector cross products as shown in the pseudo-code (see Figure 20).

Using similar techniques, shadows can be automatically re-projected as the light source moves (see Figure 21). An imaginary shadow plane is constructed encompassing the old and new shadow projectors—its surface normal inferred from the old and new light directions. The intersection of the shadow plane with the shadow-receiving object gives us the trajectory along which the new shadow point must lie. We intersect this trajectory with the new shadow projector to arrive at the new shadow point (see code in Figure 22).

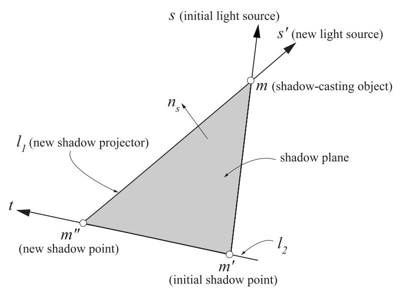


Figure 21: A shadow point is re-projected after some change in the light source direction. The new shadow point is determined by intersecting the lines l_1 and l_2, where l_1 is the new shadow projector, and l_2 is the line along which the movement of shadow point is constrained. l_2 is found by intersecting an imaginary shadow plane with the shadow-receiving object.

```
REPROJECT-SHADOW-POINT
$\mathbf{n} \leftarrow$ shadow-receiving-object-normal
$\mathbf{l}_1 \leftarrow \mathbf{m} \times \mathbf{s}'$              ▷ Shadow projector.
$\mathbf{n}_s \leftarrow \mathbf{s} \times \mathbf{s}'$              ▷ Shadow plane's normal.
$\mathbf{t} \leftarrow \mathbf{n}_s \times \mathbf{n}$               ▷ Intersection of 2 planes.
$\mathbf{l}_2 \leftarrow \mathbf{t} \times \mathbf{m}'$
$\mathbf{m}'' \leftarrow \mathbf{l}_1 \times \mathbf{l}_2$             ▷ New shadow point.
```

Figure 22: Pseudo-code for shadow re-projection.

7 Examples

We have created many drawings with our system, which demonstrate its usefulness and versatility. We used the system in conjunction with other media, such as paper sketches, paintings, and photographs. The following examples also demonstrate a variety of looks the system can generate, including freehand strokes, silhouette rendering, and fully-shaded scenes.

Paper Sketches. This example shows a panoramic sketch created entirely from a series of freehand sketches originally drawn on paper using a digital notepad. The panorama was assembled from sketches pointing at four different directions by estimating the fields of view visually (see Figure 23).

Shadows and Shading. Many of the features of our system were used in the construction of a perspective drawing of the Court of the Myrtles at Alhambra Palace, Spain (see Figure 24). For example, special vanishing points aided in the drawing of the roof tiles. Symmetrical and repeating architectural features, such as the colonnade, where copied and moved using the "apparent translation" operation. Shadows, including those cast by the colonnade and lattice onto the back wall, were projected semi-automatically. The shadow re-projection algorithm was then used to visualize the motion of the sun across the courtyard (see Figure 25).

Extrusion. This example shows the maze garden at Hampton Court Palace, which was generated by extruding the plan drawing of the maze (see Figure 26). Care was taken to maintain a proper depth order amongst the hedges. Since our system relies on a stacking order for conveying occlusion, it is not possible to have one shape wrapping around another. Such a shape must be broken up during modeling into smaller fragments—ones that are either exclusively behind or in front of other objects. This limitation, however, can be mitigated with a "grouping" tool, whereby visibility within a group is resolved on a facet-by-facet basis rather than by objects.

Projective Textures. In [12] we demonstrated the usefulness of integrating traditional drawing media and photographs with computer-based drawing. However, our previous techniques re-

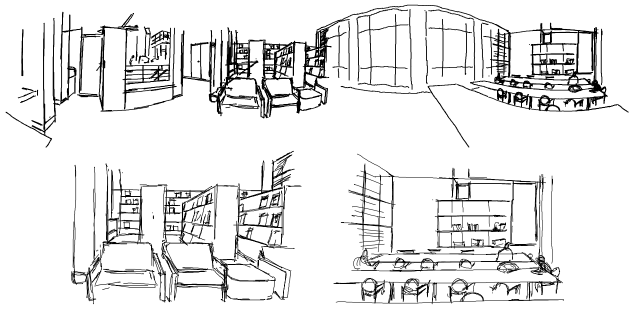


Figure 23: Panorama of library interior shown as an unrolled cylinder (top), and two of the four sketches used to create it (bottom).

quired the use of a digital notepad to draw on paper, and we used cylindrical panoramas exclusively rather than planar photographs. We have added the capability to use scanned drawings and photographs as textures applied to perspective rectangles. Texture mapping in this fashion is achieved with projective 2D mappings and image re-sampling [4]. We use transparency channels with texture to alleviate the limitations of a rectilinear shape. For example, a painting of a tree can be scanned and a binary mask created using off-the-shelf image editing software. The mask defines the tree as opaque and the rest of the rectangle transparent. The texture and mask may then be applied to a perspective rectangle inside our system. The user retains the ability to manipulate the rectangle as before, thereby allowing for precise placement of the tree in the scene. Thus, a row of trees can be created effectively by translating copies of the textured rectangle along the row's axis.

We created a scene using textured perspective rectangles composed against a panoramic backdrop (see Figure 27). Our objective was to visualize a proposed architectural design within its context. An artist used acrylic paint to create a perspective rendering of the proposed office building. We also took a series of concentric photographs of the site from approximately the same position as the painting, and used off-the-shelf software to create a cylindrical panorama, which was further processed with image editing software to imitate the look of the painting. In our system we displayed this panorama and placed a textured rectangle containing the painting, with a mask delineating the building. We added other rectangles to depict trees, people, and foreground objects, such as objects in the real scene that occlude the proposed building. The system's tools for translating and rotating rectangles provided a flexible means for placing them.

8 Discussion and Future Work

We have presented a perspective drawing system that improves upon traditional perspective drawing and greatly expands the utility of current 2D computer graphics systems. The system has the same ease-of-use as 2D systems, but offers many 3D-like qualities. It is intended for situations where the construction of a full fledged 3D model may not be necessary, for example when the viewpoint is constant. Therefore, time is potentially saved and artistic expression is not sacrificed.

The most obvious limitation of our system is the lack of relative depth information between objects. To some extent, this limitation can be overcome by allowing the user to adjust the stacking order as is commonly done in current 2D drawing packages. However, the lack of true depth information manifests itself in various other ways. For example, it prevents us from grouping objects and translating them collectively. Analogously, true 3D walk-throughs are not possible in this system. Furthermore, general lighting operations, such as shading with a local light source, require relative depth information.

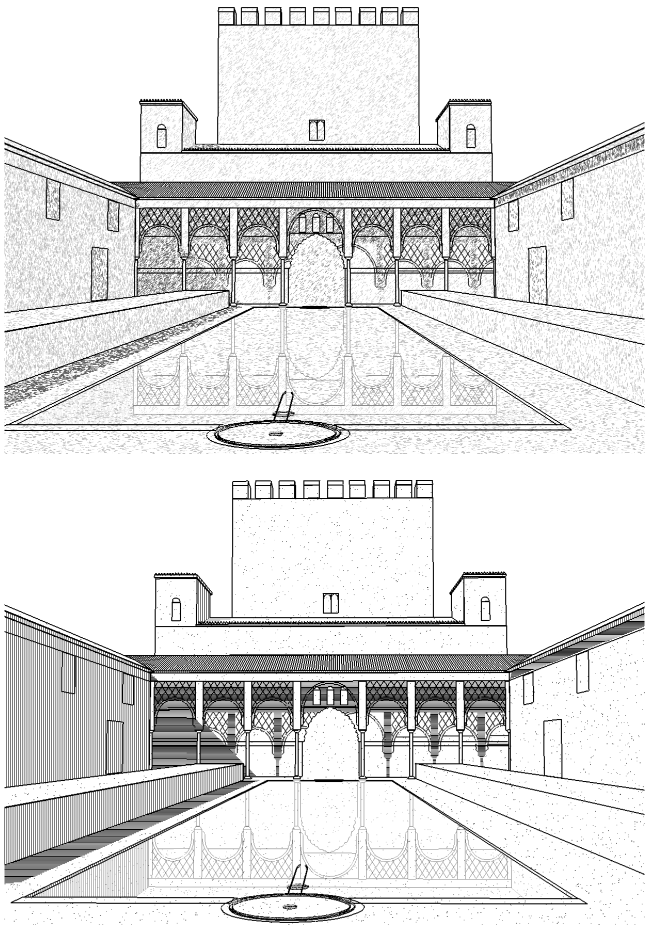


Figure 24: Using our system, we created this artistic rendering of the Court of the Myrtles at Alhambra Palace, Spain using stippling (top) and hatching (bottom).

Some of the examples were created with the help of students, who found the system's freehand and geometric primitives easy to use. However, understanding how object manipulation and shadows work required some experience with perspective drawing. The example of Alhambra was also somewhat cumbersome to construct. It took five to six hours to arrive at convincing proportions and to arrange the primitives in the drawing stack. This kind of time investment, however, is common among professional illustrators.

Our approach and representation have applications in other areas of computer graphics. For example, they are suitable for computer animation, where panoramic backdrops may be constructed using our system, providing flexible panning and zooming, in addition to shading and shadows.

We hope to embellish our system with additional modeling operations, such as the ability to generate other types of aggregate shapes. We could also include other ready-made primitives like "boxes" and "cylinders." In addition, the rendering styles we presented are by no means the only ones applicable to this approach. We could use virtually any rendering style. As more sophisticated automatic stippling or painterly rendering algorithms become available, they can be added to our system.

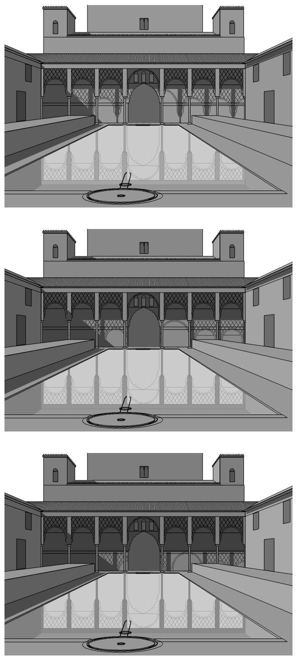


Figure 25: This sequence, showing the motion of the shadow across the back wall, was generated automatically from Figure 24.

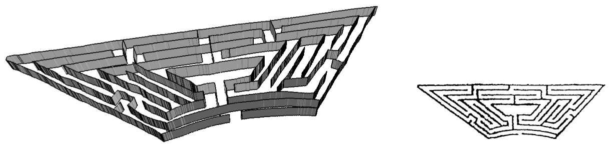


Figure 26: Perspective view of the maze garden at Hampton Court Palace (left) created by extruding its plan drawing (right).

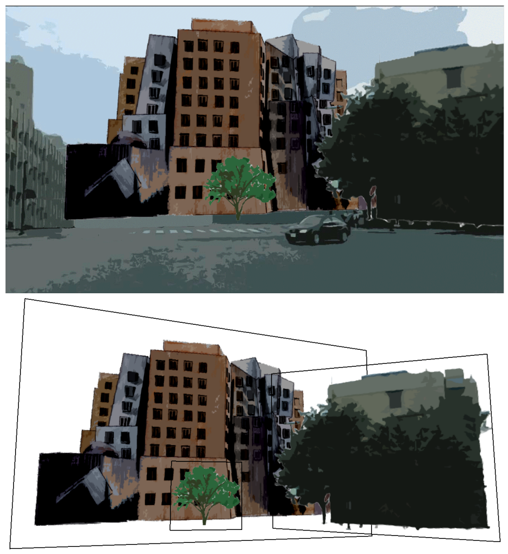


Figure 27: This scene depicts a proposed building within its real context (top). In addition to the panorama of the site, the scene contains three perspective rectangles with projective textures and transparency channels (bottom). Two of these textures include proposed elements, while the third is an existing building (the building on the right) that would occlude the proposed one.

9 Acknowledgments

We would like to thank Aparna Das and Stephen Duck for their contributions to our examples. This work was supported by an NSF CAREER awards (CCR-9624172 and CCR-9875859) and an NSF CISE Research Infrastructure award (EIA-9802220).

References

[1] Shenchang Eric Chen. Quicktime VR - An Image-Based Approach to Virtual Environment Navigation. In *SIGGRAPH 95 Conference Proceedings*, pages 29–38, August 1995.

[2] Jonathan M. Cohen, John F. Hughes, and Robert C. Zeleznik. Harold: A World Made of Drawings. In *NPAR 2000: First International Symposium on Non Photorealistic Animation and Rendering*, pages 83–90, June 2000.

[3] Robert W. Gill. *Creative Perpsective.* Thames and Hudson, London, 1975.

[4] Paul S. Heckbert. *Fundamentals of Texture Mapping and Image Warping.* Master's thesis, UCB/CSD 89/516, CS Division, EECS Dept, UC Berkeley, May 1989.

[5] S.C. Hsu, I.H.H. Lee, and N.E. Wiseman. Skeletal Strokes. In *UIST 93: Proceedings of the ACM SIGGRAPH & SIGCHI Symposium on User Interface Software & Technology*, November 1993.

[6] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A Sketching Interface for 3D Freeform Design. In *SIGGRAPH 99 Conference Proceedings*, pages 409–416, August 1999.

[7] Kenichi Kanatani. Computational Projective Geometry. *CVGIP: Image Understanding*, 54(3): 333–348, 1991.

[8] John Lansdown and Simon Schofield. Expressive Rendering: A Review of Nonphotorealistic Techniques. *IEEE Computer Graphics and Applications*, 15(3): 29–37, May 1995.

[9] Ramesh Raskar and Michael Cohen. Image Precision Silhouette Edges. In *1999 ACM Symposium on Interactive 3D Graphics*, pages 135–140, April 1999.

[10] Michael P. Salisbury, Sean E. Anderson, Ronen Barzel, and David H. Salesin. Interactive Pen-And-Ink Illustration. In *SIGGRAPH 94 Conference Proceedings*, pages 101–108, July 1994.

[11] Jorge Stolfi. *Oriented Projective Geometry: a Framework for Geometric Computations*. Academic Press, Boston, 1991.

[12] Osama Tolba, Julie Dorsey, and Leonard McMillan. Sketching with Projective 2D Strokes. In *UIST 99: Proceedings of the 12th Annual ACM Symposium on User Interface Software & Technology*, CHI Letters, 1(1): 149–157, November 1999.

[13] 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, August 1996.