Sketching Reality: Realistic Interpretation of Architectural Designs

XUEJIN CHEN
University of Science and Technology of China, and Microsoft Research Asia
SING BING KANG
Microsoft Research
YING-QING XU
Microsoft Research Asia
JULIE DORSEY
Yale University
and
HEUNG-YEUNG SHUM
Microsoft Research Asia

In this paper, we introduce "sketching reality," the process of converting a free-hand sketch to a realistic-looking model. We apply this concept to architectural designs. As the sketch is being drawn, our system periodically interprets its 2.5D geometry by identifying new junctions, edges, and faces, and then analyzing the extracted topology. The user can add detailed geometry and textures through sketches as well. This is possible using databases that match partial sketches to models of detailed geometry and textures. The final product is a realistic texture-mapped 2.5D model of the building. We show a variety of buildings that have been created using this system.

Categories and Subject Descriptors: J.6.0 [**Computer-Aided Engineering**]: Computer-aided design (CAD); I.2.10 [**Artificial Intelligence**]: Vision and Scene Understanding

General Terms: Algorithms, Experimentation, Measurement

Additional Key Words and Phrases: Sketching, realistic imagery, shape

1. INTRODUCTION

Achieving realism is one of the major goals of computer graphics, and many approaches, ranging from physics-based modeling of light and geometry to image-based rendering have been proposed. Realistic imagery has revolutionized the design process, allowing for the presentation and exploration of unbuilt scenes.

Unfortunately, creating new content for realistic rendering remains tedious and time consuming. The problem is exacerbated when the content is being *designed*. 3D modeling systems are cumbersome to use and therefore ill-suited

*This work was done when Xuejin Chen was visiting at Microsoft Research Asia.

Authors' addresses: X. Chen, Microsoft Research Asia, 5/F, Beijing Sigma Center, No.49, Zhichun Road, Hai Dian District,Beijing China 100080; email: xjchen@ustc.edu; S. B. Kang, Microsoft Research, One Microsoft Way Redmond, WA 98052-6399, USA; email: sbkang@microsoft.com; Y.-Q. Xu, H.-Y. Shum, Microsoft Research Asia, 5/F, Beijing Sigma Center, No.49, Zhichun Road, Hai Dian District,Beijing China 100080; email: {yqxu,hshum}@microsoft.com; J. Dorsey, Yale University, P.O. Box 208267, New Haven, CT 06520-8267; email: julie.dorsey@yale.edu.

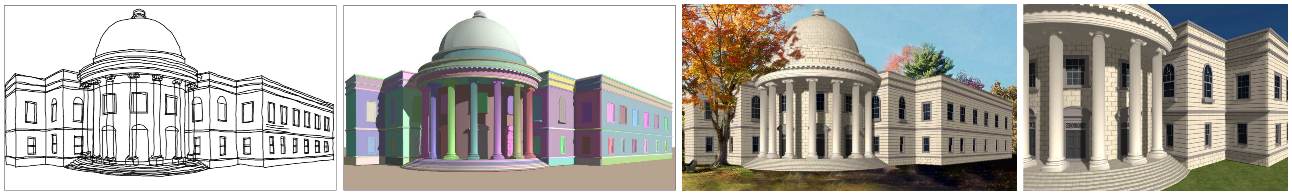


Fig. 1. Sketching reality example (rotunda), from left to right: Sketch of building, extracted 2.5D version, rendered model with a new background and rendered result with different view and lighting.

to the early stage of design (unless the design is already well-formulated). Moreover, the modeling process requires specialized knowledge in object creation and the application of textures to surfaces.

Advances in modeling and rendering notwithstanding, designers therefore continue to favor freehand sketching for conceptual design. Sketching appeals as an artistic medium because of its low overhead in representing, exploring, and communicating geometric ideas. Indeed, such speculative representations are fundamentally different in spirit and purpose from the definitive media that designers use to present designs. What is needed is a seamless way to move from conceptual design drawings to presentation rendering; this is our focus.

In this paper, we introduce a new concept, *sketching reality*, to facilitate the generation of realistic imagery. The process of sketching reality is the reverse of that of non-photorealistic rendering (NPR): Instead of mapping a real image into a stylized version, the goal is to map an NPR drawing into a plausible realistic counterpart. This goal is obviously a very difficult one—how does one interpret a sketch correctly? We apply this concept to architectural design; Figure 1 shows an example of converting a sketch of a rotunda to a 2.5D texture-mapped model. (A 2.5D model has depth computed mostly for geometric elements visible only from the original virtual view, and is thus not a full model.)

2. RELATED WORK

The methodology we describe here builds on previous work in 3D modeling, modeling by sketching, and sketch-based 3D model retrieval. We discuss representative examples of previous work in these areas, focusing on those most relevant to the work described here; comprehensive surveys are beyond the scope of this paper.

Procedural Architecture. Procedural modeling is a way to construct building models using shape grammars. Parish and Muller [Parish and Muller 2001] generated large urban environments but each building consists of simple models and shaders for facade detail. Wonka et al. [2003] generated geometric details on facades of individual buildings with split grammar. As a combination of them, CGA shape grammar is presented to generate massive urban models with geometric details at certain level [Muller et al. 2006]. While procedural methods are capable of generating a wide variety of building types and assemblies thereof, these techniques have found little application in the architectural design process. It is far more intuitive for a designer to draw sketches than to use programming languages with specific grammars.

Hierarchies of Geometric Primitives. A common approach for constructing 3D models is through the use of hierarchies of 3D geometric primitives (e.g., Maya® and AutoCAD®). Users are forced to construct models in a fixed way; more specifically, users must first decompose a model into geometric primitives and associated parameters, and, due to dependencies among parameters, construct those primitives in a particular order.

Sketch Recognition. Approaches for 2D sketch recognition tend to operate on domain-specific inputs, e.g., circuit diagrams, flowcharts, or mechanical drawings. A particularly challenging problem is resolving ambiguities in free-hand sketching. Techniques to address this problem include using predefined grammars within a Bayesian framework [Shilman et al. 2000], dynamically-built Bayes network with context [Alvarado and Davis 2005], and time-sensitive evaluation of strokes [Sezgin et al. 2001].

While most sketching interfaces require the user to draw individual symbols one at a time, Gennari *et al.*'s system [2005] parses hand-drawn diagrams and interpret the symbols. Their system focuses on network-like diagrams consisting of isolated, non-overlapping symbols *without any 3D geometry analysis*. By comparison, our system recovers generic geometric entities from rough sketches and allows strokes to overlap.

Modeling by Sketching. Sketching is a simple way to generate pictures or animations; it is not surprising that there are many systems designed based on this notion. One such example is the commercial SketchUp® system, which makes it fast to design 3D models by using a system of constraints to assist the placement of geometry. These constraints are based on the three orthogonal axes and the positions, symmetries, and proportions of previously drawn shapes.

There are many approaches for automatic interpretation of line drawings to produce 3D solid objects for CAD systems. Varley and Martin [2000], for example, use the traditional junction and line labeling method [Grimstead and Martin 1995] to analyze the topology of the 3D object. The topology gives rise to a system of linear equations to reconstruct the 3D object; note that here drawn hidden lines are not required. In contrast, the method of Lipson and Shpitalni [1996] uses hidden lines in their analysis of face-edge-vertex relationships and searches for optimal edge circuits in a graph using the A* algorithm. The alternative is to use branch-and-bound to identify the faces [Shpitalni and Lipson 1996]. Geometric relationships such as parallelism, perpendicularity, and symmetry are hypothesized and subsequently used to estimate vertex depths. Unfortunately, it is not always possible to produce a globally optimal solution.

Recently, Masry et al. [2005] proposed a sketch-based system for constructing a 3D object as it is being drawn. This system handles straight edges and planar curves. However, it assumes orthographic projection and the existence of three directions associated with three axes (estimated using the angle frequency). Again, the solution may not be unique. Our system, on the other hand, handles the more general (and difficult) case of perspective projection, and allows the user to interactively modify the sketch to produce the desired result.

There are a few sketching systems that handle perspective drawings—usually at a cost. Matsuda et al. [1997], for example, requires the user to draw a rectangular parallelepiped as the "Basic Shape" to first enable the recovery of perspective parameters. Our system requires only three drawn lines for the same purpose. In addition, our system handles (symmetric) curves while theirs does not.

There are systems that use gestures (simple 2D forms) to produce 3D models. Such systems, e.g., SKETCH [Zeleznik et al. 1996] and Sketchboard [Jatupoj 2005], constrain the designer's sketching in exchange for computer understanding. They use recognizable gestures that cause 3D editing operations or the creation of 3D primitives. The approximate models thus generated are typically not appropriate for realistic rendering. SMARTPAPER [Shesh and Chen 2004] uses both gestures and automatic interpretation of drawn polyhedral objects. However, orthographic projection is assumed and hidden lines must be drawn. Furthermore, curves objects and complicated architectural structures and textures are not supported.

One of the more influential sketch-based systems is Teddy [Igarashi et al. 1999], which largely eliminated the vocabulary size and gesture recognition problems by supporting a single type of primitives. Teddy allows the user to very easily create 3D smooth models by just drawing curves and allowing operations such as extrusion, cut, erosion, and bending to modify the model. An extension of Teddy, SmoothSketch [Karpenko and Hughes 2006] is able to handle non-closed curve caused by cusps or T-junctions. While they focus on free-form objects, our system interprets complicated architecture scenes from perspective sketches. A "suggestive" interface for sketching 3D objects was proposed by Igarashi and Hughes [2001]. After each stroke, the system produces options from which the user chooses. While flexible, the frequent draw-and-select action is rather disruptive to the flow of the design.

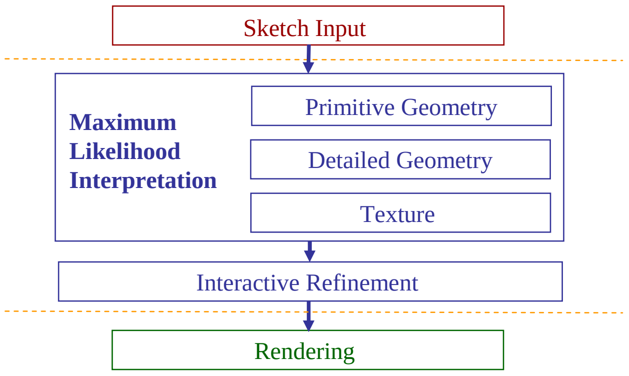


Fig. 2. Overview of our sketching system.

Sketch-based 3D Model Retrieval. Techniques that retrieve 3D models from sketches are also relevant to our work. Such techniques provide intuitive shortcuts to objects or classes of objects, which are useful for designing. Funkhouser et al.'s system [2003] is a hybrid search engine for 3D models that is based on indexing of text, 2D shape, and 3D model. To index 2D shape, their system compares the sketch drawn by the user with the *boundary contours* of the 3D model from 13 different orthographic view directions and computes the best match using an image matching method [Huttenlocher et al. 1993; Barrow et al. 1977]. Note that the view-dependent internal curves are not used for 2D indexing. Spherical harmonics (quantized into a 2D rotation-invariant shape description) is used to index 3D.

Two methods for 2D drawing retrieval are proposed in [Pu and Ramani 2006]. The first uses a 2.5D spherical harmonic representation to describe the 2D drawing, while the second uses the histogram of distance between two randomly sampled points on the 2D drawing. Hou and Ramani [2006] proposed a classifier combination framework to search 3D CAD components from sketches. However, their system requires sketches of boundary contours of objects drawn from three orthographic views.

In our system, to retrieve a detailed structure, the user draws its *appearance* rather than its boundary contour. We use shape information that includes texture features, relative location information, as well as architectural rules.

3. OVERVIEW

Three parts compose our system: sketch input, automatic sketch interpretation, and interactive refinement (Figure 2). All user interaction is through a digital pen on a tablet. The user starts by drawing three lines to implicitly specify the current camera viewing parameters $\mathbf{c}_0$. The first line corresponds to the horizon while the other two specify the lines corresponding to the two other projected perpendicular directions. Note that $\mathbf{c}_0$ also contain information about the three vanishing points associated with the three mutually perpendicular directions. The user also specifies (through a menu) the type of architecture to be drawn: classical, Eastern, or modern.

As the user sketches, each drawn line is immediately vectorized. Each line drawn by user is interpreted as a dot, circle, straight line segment, or cubic Bezier curve based on least-squared fit error.

A sketch consists of three types of objects, namely, primitive geometries, detailed geometries, and textures. A *primitive geometry* can be a 2.5D or 3D basic geometric element, e.g., rectangle, cube, cylinder, or half-sphere. A *detailed geometry* refers to a complex substructure that is associated with a particular type of architecture, e.g., windows, columns, and roof ornaments. Finally, textures are used to add realism to surfaces, and they can be specified through sketching as well.

At any time, the user can ask the system to interpret what has been sketched so far. The system uses a maximum likelihood formulation to interpret sketches. Depending on the object type, different features and likelihood functions are used in the interpretation.

The user can optionally refine the interpreted result. Our system is designed to allow the user to easily edit the local geometries and textures. The final result can be obtained by rendering the texture-mapped model at a desired lighting condition and (optionally) new viewpoint and background.

4. SKETCH INTERPRETATION WITH MAXIMUM LIKELIHOOD

We assume that the user inputs the sketch by line-drawing, and that each line L_i consists of N_i contiguous 2D points $\{p_{ij} | j \in 1, ..., N_i\}$. Given a sketch with N_S lines, denoted $\mathbf{S} = \{L_i | i \in 1, ..., N_S\}$, a set of architecture-specific priors $\mathbf{A}$, and the camera viewing parameters $\mathbf{c}_0$, we would like to infer the set of objects $\mathbf{M}$, which can be any combination of primitive geometries, detailed geometries, and textures.

We use *maximum a posteriori* (MAP) inference to estimate $\mathbf{M}$ given $\mathbf{S}$, $\mathbf{A}$, and $\mathbf{c}_0$:

$$P(\mathbf{M}|\mathbf{S}, \mathbf{A}, \mathbf{c}_0) \propto P(\mathbf{S}|\mathbf{M}, \mathbf{A}, \mathbf{c}_0) P(\mathbf{M}|\mathbf{A}, \mathbf{c}_0),$$

with $P(\mathbf{S}|\mathbf{M}, \mathbf{A}, \mathbf{c}_0)$ being the *likelihood* and $P(\mathbf{M}|\mathbf{A}, \mathbf{c}_0)$ the *prior*. We assume that $P(\mathbf{M}|\mathbf{A}, \mathbf{c}_0)$ is uniform, which converts the MAP problem to that of maximum likelihood (ML). We would like to maximize the likelihood

$$P(\mathbf{S}|\mathbf{M}, \mathbf{A}, \mathbf{c}_0). \tag{1}$$

In contrast with domain-specific sketches, such as those of circuit diagrams, architectural drawings consist of elements that vary dramatically in scale (from large-scale shapes in the form of primitives to detailed structures). We use three databases that are customized based on type: primitive geometries, detailed geometries, and textures. For each type, we index each entity with its 2D drawing representation, as indicated in Figure 3. Each detailed geometry is associated with specific architectural principles that define its probability of occurrence in the database.

Once the user completes drawing part of the sketch, he then specifies the type. Given this information, our system extracts the appropriate features based on type and retrieves the best match in the database. We define a different likelihood function $P(\mathbf{S}|\mathbf{M}, \mathbf{A}, \mathbf{c}_0)$ on different features extracted from the sketches for each type. Details of the features and likelihood functions used for each type are described in following sections.

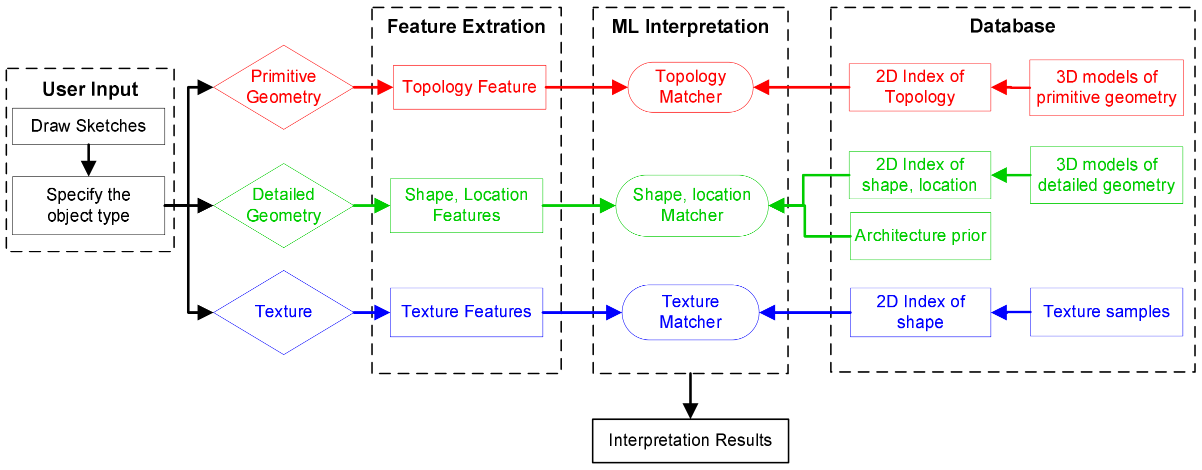


Fig. 3. Use of maximum likelihood interpretation on the three types of objects (primitive geometry, detailed geometry, texture).

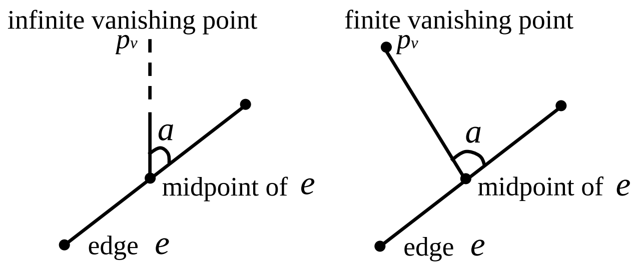


Fig. 4. Distance function $d(e, p_v)$. For an infinite vanishing point (left), the distance is the angle between the edge and the direction of the vanishing point. Otherwise (right), the distance is the angle between the edge and the vector joining the middle point and the vanishing point p_v.

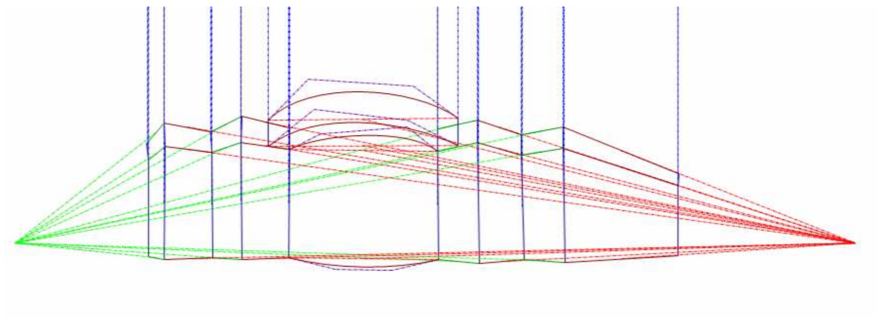


Fig. 5. Edge direction estimation. The edges are color-coded based on perpendicular direction. The dash line connect the midpoint of each edge and its corresponding vanishing point. The classification of direction for a curve is based on the line connecting its two endpoints. Each curve has a set of control points and control lines (purple dashed lines), which are used to compute the junctions and reconstruct the geometry.

5. INTERPRETATION OF PRIMITIVE GEOMETRIES

The overall shape of a building can generally be characterized by combining primitive geometries such as cubes, prisms, pyramids (including curved pyramids), and spheres. The interpretation of geometry primitives from a sketch depends heavily on the projected topology $\mathbf{T}$. By topology, we mean the layout of junctions $\mathbf{J}$, edges $\mathbf{E}$, and faces $\mathbf{F}$, i.e., $\mathbf{T} = \{\mathbf{J}, \mathbf{E}, \mathbf{F}\}$. If we define $\mathbf{T}(\mathbf{S})$ as the topology of the sketch and $\mathbf{T}(\mathbf{M})$ the union of the topologies of the primitive geometries $\mathbf{M}$, the likelihood function (1) reduces to

$$P_{geo}(\mathbf{T}(\mathbf{S})|\mathbf{T}(\mathbf{M}), \mathbf{c}_0). \tag{2}$$

In order to maximize this likelihood function, we have to recover $\mathbf{T}(\mathbf{S})$. This involves edge analysis with vanishing points, junction identification, and face labelling.

5.1 Recovery of Topology

In order to recover the topology of the sketch $\mathbf{T}(\mathbf{S})$, we first progressively build a graph $G(\mathbf{V}, \mathbf{E})$ as the user sketches. $G(\mathbf{V}, \mathbf{E})$ is a null set initially, and as the user adds a line, its vertices and edge are added to $G(\mathbf{V}, \mathbf{E})$. Note that a line can be straight or curved: $\mathbf{E} = \{e_i, | i \in 1, ..., N_E\}$, e_i being the ith edge, and N_E the number of lines. ($N_E \leq N_S$ because we ignore sketched points here.) Each edge has the information on end points (two vertices), type (straight or curved), and numbers that indicate how well the edge fits the three vanishing points (to be described shortly). If the edge is curved, it has the additional information on the Bezier control points. The vertices and edges are analyzed to find the junctions $\mathbf{J}$ in the sketch.

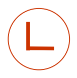
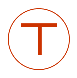
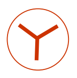
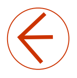
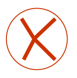
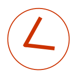
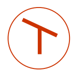
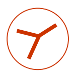
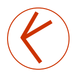
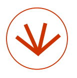
types	**L**	**T**	**Y**	**E**	**X**
samples					
					

Fig. 6. Five types of junctions used in our system

Edge Analysis. We assume that our building consists of three groups of edges that are mutually perpendicular in 3D space. From the user-supplied perspective parameters $\mathbf{c}_0$, we obtain three vanishing points p_{v0}, p_{v1}, and p_{v2}. Each vanishing point is associated with an orthogonal direction. The edges in $G(\mathbf{V},\mathbf{E})$ are grouped based on proximity to these vanishing points measured by the distance function

$$g(e,p_v) = \begin{cases} 1 - \frac{d(e,p_v)}{T_a} & , \text{ if } d(e,p_v) < T_a \\ 0 & , \quad \text{otherwise} \end{cases} , \tag{3}$$

where $d(e,p_v)$ is the distance function between edge e and the hypothesized vanishing point p_v (shown in Figure 4), and T_a is a sensitivity parameter [Rother 2000]. (For a curved edge, we use its end points to approximate its direction.) We set $T_a = 0.35$ radian in our system. Figure 5 shows the result of edge direction estimation, while each color representing the most likely direction.

Junction Types. A *junction* is an intersection point between two or more edges. It is classified based the angle of intersection and the number of intersecting edges. We use five junction types in our work (shown in Figure 6), which are adequate for describing most regular buildings. The L junction is just a corner of a face. The T junction occurs when a face is occluded by another, or when the vertex is on the edge in geometry. The Y junction represents a common vertex where three visible faces meet, while the E junction occurs when two visible faces meet along a common edge. More generally, the X junction is the result of several faces meeting in a common vertex. The extracted junction labels for a part of the hall example are shown in Figure 7.

Assignment of Ordered Vertices to Faces. Once the junction types and edge directions have been analyzed, the next step is to recover the topology of the sketch by associating an ordered list of vertices for each face. Shpitalni and Lipson [1996] assume that the loops with shortest path in the graph G(V,E) are actual faces of the 3D objects for wireframe drawing. They assume hidden lines are drawn by the user to make their analysis possible.

In our system, hidden lines are not necessary. As such, it is possible that a partially occluded face in the sketch does not correspond with the loop with shortest path in the graph. To remove inconsistent faces, we use the following property: Since all the edges of a face are planar, all the vanishing points associated with the groups of parallel edges of a face must be collinear under perspective projection. We use the vanishing point collinearity property to cull impractical faces and determine the existence of occluded T junctions.

The vertex assignment step propagates from a single arbitrary vertex to faces (that contain that vertex) to other vertices (that are part of these faces), and so on. When we examine a face F having vertex V to determine its bounding edge, the candidate edges are those connected to vertex V. We use a greedy algorithm to choose the edge that has the largest probability to be on the face and continue the propagation. Meanwhile, we get some new candidate faces at this vertex V formed by the other edges connecting to it. The algorithm is subject to the condition that the edges of the same face cannot correspond to three orthogonal directions.

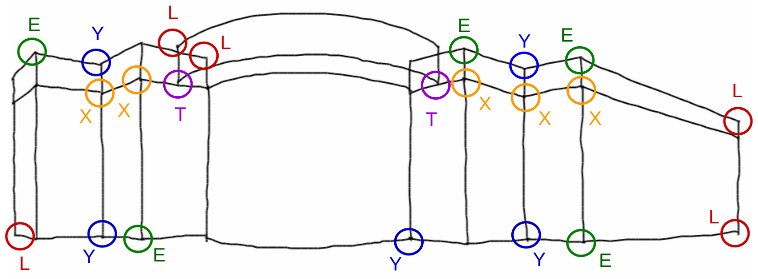


Fig. 7. Junction labels. Only a fraction of the labels are shown for clarity.

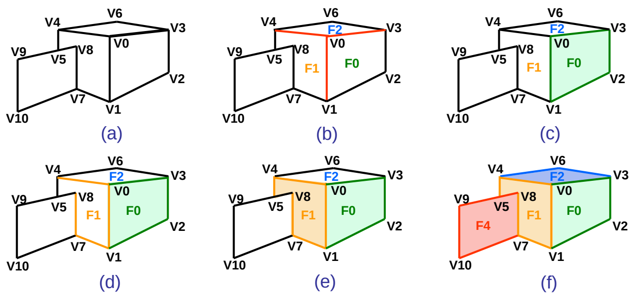


Fig. 8. Example of vertex assignment to faces.

Suppose the current face has k ordered edges $e_0, \ldots, e_{k-1}$, and e_k is the candidate edge. Suppose also the face is bounded by edges associated with two vanishing points p_{va} and p_{vb}. We define the likelihood of F to be a valid face as

$$P(F \text{ valid}|e_0, \ldots, e_k, p_{va}, p_{vb}) = P(p_{va})P(p_{vb}) \sqrt[k+1]{\prod_{i=0}^{k} P(e_i, p_{va}, p_{vb})}.$$

The term $P(p_v)$ describes the likelihood that the vanishing point p_v is consistent with F, i.e., $P(p_v) = \max_i g(e_i, p_v)$, with $g()$ defined in (3). The term $P(e_i, p_{va}, p_{vb})$ is a measure of the likelihood that the edge e_i corresponds to a vanishing point of F, and is defined as $P(e_i, p_{va}, p_{vb}) = \text{larger}(g(e_i, p_{va}), g(e_i, p_{vb}))$, with larger() returning the larger of the two input parameters.

Given p_{v0}, p_{v1}, and p_{v2}, we choose the pair of vanishing points that maximizes $P(F \text{ valid}|e_0, \ldots, e_k, p_{va}, p_{vb})$ as the vanishing points of face F. If this maximum value is small, the face is considered invalid; in this case, propagation is terminated at the current vertex. Otherwise, the face is a valid face and each edge of this face e_i is corresponding to the vanishing point p_{va} if $g(e_i, p_{va}) > g(e_i, p_{vb})$ or p_{vb} otherwise.

We use Figure 8 to illustrate the process of vertex-to-face assignment:

(1) We first start from an arbitrary vertex, in this case, a Y junction $V0$ (Figure 8(a)).

(2) Next, we hypothesize faces according to the junction type and the edges connecting to it. Since the starting vertex correspond to a Y junction, we have three new *candidate* faces (with vertices initially assigned): $F0 : V1 - V0 - V3$, $F1 : V4 - V0 - V1$, and $F2 : V4 - V0 - V3$ (Figure 8(b)).

(3) We traverse along the edges of each candidate face to get a *closed face*.
   - $F0$: Starting from $V3$, there are two candidate edges: $V3 - V2$ and $V3 - V6$. We choose $V3 - V2$ because it closes the loop for $F0$ *and the edges are consistent with the face. The closed face* is thus $V1 - V0 - V3 - V2 -$

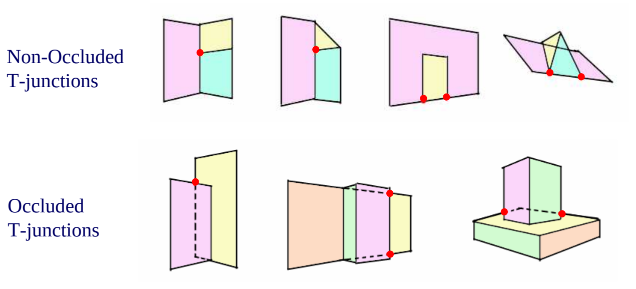


Fig. 9. Different types of T junctions in the sketch. The red vertices are T-Junctions. For the occluded faces caused by T junctions, we complete them according to symmetry or parallelism constraint, shown as dashed lines.

$V1$ (Figure 8(c)).

- $F1$: Starting from $V1$, we proceed toward $V7$ and $V8$. However, the propagation ends because $V8-V9$ is not consistent with the face (Figure 8(d)). As a result, we instead propagate from the other end $V4$, then to $V5$. Since $V4-V5$ is an occluded edge ($V5$ being a T junction), the propagation again terminates. We end up with an incomplete face $V5-V4-V0-V1-V7-V8$ (Figure 8(e)).

- $F2$: By straightforward traversal, we get the vertices of closed face $F2$: $V4-V0-V3-V6-V4$.

(4) We then proceed to another candidate face created at the other vertices, and repeat steps 2-4 until all the vertices and edges are traversed (Figure 8(f)).

During the propagation of the faces, the special T junctions sometimes cause the termination of the propagation and result in incomplete (i.e., partially occluded) faces. We complete the partially occluded faces using symmetry or parallelism constraints (Figure 9). We extend the edge at the T junction to the vertex that produces a symmetric shape with respect to the visible part.

After accounting for partially occluded faces, we look for the best interpretation of the sketches in the database according to the likelihood function defined in (2). We then reconstruct the 2.5D geometry associated with the geometry primitives.

5.2 Geometry Reconstruction

The 2.5D geometry can be recovered from a single view given the camera parameters and symmetry constraints. This is accomplished by constructing a system of linear equations that satisfy the projection, edge-vanishing point correspondence, and structural constraints. Each constraint is weighted based on its perceived reliability. This system of equations is solved using the conventional weighted least squares technique, and produces the 3D locations of all vertices (including those of curved objects).

Note that only the end points (vertices) of curved objects are computed using this single-step procedure. Since curved objects are cubic Bezier curves, we will need to compute their 3D control point locations. This is accomplished using an iterative technique that relies on the projection and symmetry constraints.

5.2.1 *Reconstruction of Planar Objects.* As mentioned earlier, there are three groups of the linear equations or constraints in our system, namely projection, edge-vanishing point correspondence, and structural constraints.

Projection Equations. The camera parameters $\mathbf{c}_0$ are used to produce two linear constraints for each vertex. Given vertex $\mathbf{v}(x,y,z)$ with its projection (u,v), we have

$$\begin{cases} u = \frac{x}{z} * f \\ \\ v = \frac{y}{z} * f \end{cases} \Rightarrow \begin{cases} x * f - u * z = 0 \\ \\ y * f - v * z = 0 \end{cases}.$$

f is the focal length, i.e., the distance from the eye to sketch plane. Note that this form of linearization is theoretically non-optimal, since we are essentially weighting each pair of constraints by the (unknown) depth z. The linearization is done more for convenience and speed, since it allows us to extract the 3D coordinates in a single step. Moreover, the optimality of solution is not critical, since we are dealing with sketches (which provide only approximate shape in the first place). We need only to extract a plausible shape. We "normalize" the implicit weights using the weight $w_{proj} = 1/f$.

Edge-Vanishing Point Correspondence. In Section 5.1, we described how each edge is assigned a direction (and hence a vanishing point). This assignment can be cast as a constraint:

$$\mathbf{v}_1 - \mathbf{v}_2 = \Omega(p_1, p_2, p_v) \frac{v_{1z}|p_1 - p_2|}{f|p_v - p_2|}(x, y, f)^{\mathrm{T}},$$

where $\mathbf{v}_1$ and $\mathbf{v}_2$ are the 3D endpoints of the edge, and p_1 and p_2 are the respective corresponding 2D points. p_v is the vanishing point associated with the edge direction. v_{1z} is the z-component of $\mathbf{v}_1$, and $\Omega(p_1, p_2, p_v)$ returns 1 if p_1 is between p_2 and p_v, or -1 otherwise. The weight for this constraint is the distance measure of the edge to the assigned vanishing point, i.e., $w_{vp} = g(e, p_v)$.

Structural Constraints. We also impose the structural constraints of symmetry, parallelism, and perpendicularity on the vertices as

$$\begin{array}{rl} equality: & \mathbf{v}_a - \mathbf{v}_b = \mathbf{v}_c - \mathbf{v}_d \\ parallelism: & \mathbf{v}_a - \mathbf{v}_b \parallel \mathbf{v} \\ perpendicularity: & (\mathbf{v}_a - \mathbf{v}_b) \cdot \mathbf{v} = 0 \end{array}.$$

Note that these structural constraints are applied through geometric primitives (each having its own predefined constraints on the vertices and edges). Given the recovered topology (Section 5.1), we find the most likely geometric primitive in our database. We then apply the appropriate structural constraints to the vertices and edges to recover their 3D values. In order to satisfy the constraints, we assign a very large weight on them; in our implementation, this weight is $w_{con} = 1000$.

Since the vertex coordinates are relative, we set one vertex as the reference. We randomly select a vertex $\mathbf{v}_{ref}$: $x_{ref} = u_{ref}$, $y_{ref} = v_{ref}$, and $z_{ref} = f$. Given all the visible vertices $\mathbf{v}_0, ..., \mathbf{v}_{n-1}$, with $\mathbf{v}_0 \equiv \mathbf{v}_{ref}$, we have $3n$ unknowns, that is, $\mathbf{x} = (x_0, y_0, z_0, ..., x_{n-1}, y_{n-1}, z_{n-1})^T$.

Using the projection, edge-vanishing point correspondence, and structural constraints described above, we arrive at a system of weighted linear equations $\mathscr{A}\mathbf{x} = \mathscr{B}$, which we can easily solve to produce the 3D vertex coordinates. An example of reconstructed geometry is shown in Figure 10.

5.2.2 *Reconstruction of Curved Objects.* While planar surfaces are prevalent in architecture, curved surfaces are not unusual. Shape reconstruction of curved surfaces from a single view requires assumptions or constraints, such as orthographic projection [Prasad et al. 2006] or reflecting symmetry [Hong et al. 2004]. We assume symmetry for curved surfaces to enable a unique solution to be found. In addition, we require interactive rates of reconstruction, which most previous techniques are unable to satisfy.

We accomplish our goal of shape recovery at interactive rates by approximating the geometry of curved edges with a small number of cubic Bezier curves. There are symmetric curves with rotation or reflection symmetry that have same

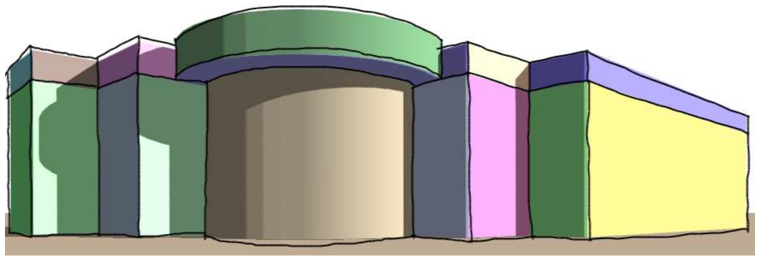


Fig. 10. Reconstructed geometry model from free-hand drawn sketch of rotunda.

```
(1) Regularly sample $N$ points along $\gamma^k(s), k = 0, ..., K-1$:
    For $i = 0 : N-1$
    - Sample 2D points $p_i^k$ on the curves $\gamma^k$, $k = 0, ..., K-1$; corresponding 3D points are $\mathbf{v}_i^k, k = 0, ..., K-1$.
    - Build a system of linear equations using projection, edge-vanishing point correspondence, and symmetric equations considering $\mathbf{v}_i^k, k = 0, ..., K-1$ are all related by rigid transformations to the 3D end points.
    - Solve to recover $\mathbf{v}_i^k, k = 0, ..., K-1$.
(2) Fit cubic Bezier curve to reconstructed points for each curve (outputs are control points):
    For $k = 0 : K-1$
    - CubicBezierFitting($\mathbf{v}_0^k, \mathbf{v}_1^k, ..., \mathbf{v}_{N-1}^k$)$\Rightarrow$ ($\mathbf{v}'^k_{ctr_0}, \mathbf{v}'^k_{ctr_1}, \mathbf{v}'^k_{ctr_2}, \mathbf{v}'^k_{ctr_3}$). $\mathbf{v}'^k_{ctr_0}$ and $\mathbf{v}'^k_{ctr_3}$ are the two endpoints of the 3D curve $\Gamma^k$.
(3) Optimize parameterized curves defined by common set of control points specified by $\mathbf{t}$: $\mathbf{v}^k_{ctr_1}(\mathbf{t}), \mathbf{v}^k_{ctr_2}(\mathbf{t})$, $k = 0, ..., K-1$.
    The optimal parameters are:
    $\mathbf{t}^* = \arg\min \sum_{k=0}^{K-1} \sum_{i=1}^{2} \|\mathbf{v}^k_{ctr_i}(\mathbf{t}) - \mathbf{v}'^k_{ctr_i}\|^2$ subject to $\mathscr{A}_t \mathbf{t} \geq \mathscr{B}_t$.
```

Fig. 11. Algorithm for reconstructing K symmetric curves. In our implementation, $N = 50$.

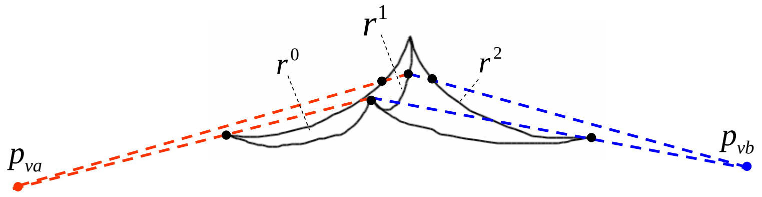


Fig. 12. Sample a group of symmetric points on symmetric curves according to vanishing points.

parameters in each type of primitive geometry. This symmetric constraint, together with the projection constraint, allows a unique solution to be extracted.

Let the 2D curved lines which are the projection of the 3D symmetric curves be denoted $\gamma^0(s)$, $\gamma^1(s)$,...,$\gamma^{K-1}(s)$, $s \in [0,1]$. We wish to reconstruct their 3D curve counterparts $\Gamma^0(\mathbf{t})$, $\Gamma^1(\mathbf{t})$,...,$\Gamma^{K-1}(\mathbf{t})$. $\mathbf{t}$ is the vector of parameters of the cubic Bezier curves that are common across these 3D curves (modulo rigid transformation). The process of estimating these 3D curves is decomposed into three steps for speed considerations: (1) sampling points on the curve and estimating their 3D positions, (2) fitting cubic Bezier curves on the reconstructed 3D points, and (3) optimizing a common set of parameters for all these curves. These steps are detailed in Figure 11.

We use Figure 12 to illustrate Step 1 on three curves γ^0, γ^1, and γ^2. We know that all 3D lines that pass through corresponding points on two 3D symmetric curves must be parallel. The projections of these lines must then intersect at a vanishing point; the red dashed lines are two such lines, with the vanishing point being p_{va}. First, we regularly sample one of the 2D curves (say γ^0): $p_i^0 = \gamma^0(\frac{i}{N-1}), i = 0, ..., N-1$. Given the vanishing point p_{va}, we can find the

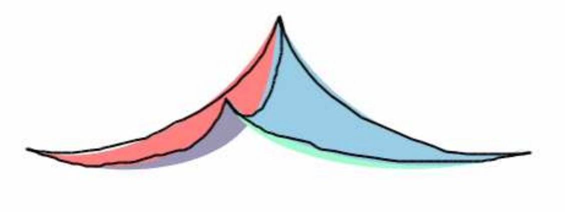


Fig. 13. Result of reconstructing the symmetrically curved surfaces. The sketch is the set of solid dark lines while the recovered curved surfaces are color-coded.

corresponding points on γ^1, and subsequently, those on γ^2, this time using vanishing point p_{vb}. The two blue dashed lines show how two corresponding points in γ^2 are computed given two points in γ^1 and vanishing point p_{vb}.

The least-squares minimization in Step 3 is subject to a collection of inequality constraints on the control point parameterization. These constraints are specified as $\mathscr{A}_t\mathbf{t} \geq \mathscr{B}_t$; the inequality is necessary to enforce the *ordering* of the control points. We use the Gradient Projection Method to solve the non-linear optimization problem with linear constraints in Step 3. The reconstructed result for the two symmetric surfaces is shown in Figure 13.

6. INTERPRETATION OF DETAILED GEOMETRIES

While the overall shape of the building provides a good picture of the design, the details are typically required to make the design look compelling. Our system allows the user to add detailed geometries such as windows, columns, and entablatures. At first glance, this seems impossible to implement. For example, the complicated geometry of the capital of a column seems difficult to reconstruct from a coarse sketch.

Interestingly, however, architects do not usually draw fine delicate strokes to indicate detailed structures, but rather apply a series of well-established architectural conventions for drawing. We consulted an architect and relevant literature to construct priors in the form of databases that reflect the various rules for different architectural styles. For example, the layout of classical columns and entablature structure is well-defined. One rule specifies that the entablature must be resting on top of the columns. Another requires the entablature to consist of three parts called the cornice, frieze, and architrave, each with specific shapes. In the sketch of the rotunda shown in Figure 1(a), the strokes for the entablatures and steps are very similar in that they both have repeated curve lines. They are differentiated by relative location on the building.

The likelihood function defined in (1) can thus be customized for detailed geometries as follows:

$$P_{detail}(\mathbf{S}|\mathbf{M},\mathbf{A},\mathbf{c}_0) \rightarrow P(\mathbf{F}_{shape},\mathbf{L},style,|\mathbf{M},\mathbf{A},\mathbf{c}_0) \propto P(\mathbf{F}_{shape}|\mathbf{F}^{\mathbf{M}}_{shape})P(\mathbf{L},style,|\mathbf{M},\mathbf{A}). \tag{4}$$

$\mathbf{F}_{shape}$ and $\mathbf{F}^{\mathbf{M}}_{shape}$ describe the shape features of the strokes drawn by user and the 2D index of the detailed geometry in the database, respectively. The features include the the density of the strokes, the aspect ratio of the stroke area, and the statistic texture features. $\mathbf{L}$ is the vector of stroke locations on the building, such as on the top of roof, along an edge, at bottom of an object, etc. The best interpretation $\mathbf{M}$ for the drawn sketch also relies on the prior $\mathbf{A}$ with particular architecture style $style$.

6.1 Shape Features

Prior 3D model retrieval approaches typically rely on contour matching. In our case, however, the user generally does not draw the boundary contour to represent a detailed geometry (the exceptions being windows and doors). To represent repetitive detailed geometry, such as tiles on a roof, the user needs only to draw a few representative texture-like strokes on the roof surface.

In our system, we use shape similarity to extract the right detailed geometry from a sketch. Our shape similarity measure includes some form of texture similarity. Assume that the user drew N_s strokes $\mathbf{S} = \{s_0, s_1, ..., s_{N_s}\}$ to represent the detailed geometry. Our measure of texture similarity consists of metrics associated with three types of histograms, namely, the shape histogram, orientation histogram, and length histogram.

Shape Histogram. This histogram specifies the normalized frequency of occurrance of types of stroke shapes for the drawn sketch. It is defined as $\mathbf{h}_s = \{\hat{n}_i | i = dot, circle, straight\ line, curve\}$, where $\hat{n}_i = n_i/N_S$, with n_i being the number of strokes associated with the ith shape type. Two shape histograms $\mathbf{h}_{s1}$, $\mathbf{h}_{s2}$ are similar if their Euclidean distance is small:

$$d_s(\mathbf{h}_{s1}, \mathbf{h}_{s2}) = |\mathbf{h}_{s1} - \mathbf{h}_{s2}|.$$

Orientation Histogram. The orientation histogram is widely used in hand gesture recognition [Lee and Chung 1999]. For both straight lines and curves, we just use the line between the two endpoints to calculate the orientation. Their orientation angles are estimated and accumulated in the appropriate histogram bins (the orientation angle $\in [0, \pi)$ is quantized into n_o bins). We then normalize the frequency as was done for the shape histogram: $\mathbf{h}_o = \{\hat{m}_i | i = 0, ..., (n_o - 1)\}$, with $\hat{m}_i = m_i/N_S$ and m_i being the number of the lines whose orientation angle $\in [i\pi/n_o, (i+1)\pi/n_o)$.

Because there is ambiguity in the reference angle, rotationally shifted versions of the histogram are equivalent. Hence, to compute the similarity between two histograms $\mathbf{h}_{o1}$ and $\mathbf{h}_{o2}$, we first compute the Euclidean distances between $\mathbf{h}_{o1}$ and n_o shifted versions of $\mathbf{h}_{o2}$, and take the minimum distance.

$$\begin{aligned} d'_o(\mathbf{h}_{o1}, \mathbf{h}_{o2}) &= |\mathbf{h}_{o1} - \mathbf{h}_{o2}(s')|, \text{where} \\ s' &= \arg\min_{s=0\ldots(n_o-1)} |\mathbf{h}_{o1} - \mathbf{h}_{o2}(s)|, \end{aligned}$$

with $\mathbf{h}_{o2}(s)$ being $\mathbf{h}_{o2}$ rotationally shifted by an angle equivalent to s bins (i.e., $s\pi/n_o$). The normalized shift of the histogram is $\hat{s}_{min} = \min(s', n_o - s')/n_o$.

We penalize the shift of the histogram to distinguish the sketches that has small shifting histogram distance. The modified distance of the orientation histogram is defined as

$$d_o(\mathbf{h}_{o1}, \mathbf{h}_{o2}) = d'_o(\mathbf{h}_{o1}, \mathbf{h}_{o2}) + \lambda \hat{s}_{min}.$$

λ is a factor that penalizes histogram shifting. In our implementation, $n_o = 10$ and $\lambda = 0.05$.

Length Histogram. The length histogram is used to characterize the distribution of stroke sizes. Given the bounding box of the drawn sketch with its width w and height h, we use $L_{nor} = \max(w, h)$ as the normalizing factor for each stroke. The width of the histogram bin is set to L_{nor}/n_l. (Note that the number of bins is $n_l + 1$; the last bin is used to accumulate stroke lengths $\geq L_{nor}$.) We define the length histogram as $\mathbf{h}_l = \{\hat{l}_i | i = 0, \ldots, n_l\}$. $\hat{l}_i = l_i/N_S, i = 0, 1, \ldots, n_l$, where $l_i (i < n_l)$ is the number of the strokes with lengths $\in [i/L_{nor}, (i+1)/L_{nor})$ and l_{n_l} is the number of strokes with lengths $\geq L_{nor}$. In our implementation, $n_l = 10$. Two length histograms $\mathbf{h}_{l1}$ and $\mathbf{h}_{l2}$ are similar if their Euclidean distance is small:

$$d_l(\mathbf{h}_{l1}, \mathbf{h}_{l2}) = |\mathbf{h}_{l1} - \mathbf{h}_{l2}|.$$

We define the texture distance between two sketches T_1 and T_2 to be a weighted average of the distance in the shape, orientation, and length histogram:

$$d_{all}(T_1, T_2) = w_s d_s(\mathbf{h}_{s1}, \mathbf{h}_{s2}) + w_o d_o(\mathbf{h}_{o1}, \mathbf{h}_{o2}) + w_l d_l(\mathbf{h}_{l1}, \mathbf{h}_{l2}). \tag{5}$$

w_s, w_o, and w_l are the weights used to indicate the amount of influence for the different histogram measures. Because the scale of the sketch can vary significantly (which results in widely varying length histogram distances), we assign a smaller weight to the length histogram to reduce its influence. In our implementation for detailed geometry, $w_s = w_o = 10, w_l = 1$. Note that because the orientation histogram and length histogram are quantized, we pre-process them with a simple kernel smoother with filter weights $(0.1, 0.8, 0.1)$ to reduce the quantization effect.

Recall that the bounding box of the sketch is of width w and height h; let the total length of all the strokes be L_{sum}. The density of the strokes $\Delta_s = L_{sum}/(w*h)$ and the aspect ratio $a_s = w/h$. We define the shape similarity between the strokes drawn by user and the index drawing of the detailed geometry as

$$P(\mathbf{F}_{shape}|\mathbf{F}^{\mathbf{M}}_{shape}) = P(\Delta_s)P(a_s)P(T_s|T_m) = \exp\left(-\frac{(\Delta_s-\mu^m_d)^2}{2\sigma^{m2}_d}\right)\exp\left(-\frac{(a_s-\mu^m_a)^2}{2\sigma^{m2}_a}\right)\frac{1}{1+d_{all}(T_s,T_m)}.$$

The parameters μ^m_d,σ^m_d, μ^m_a,σ^m_a are determined according to different detailed geometry entity $\mathbf{M}$. $P(T_s|T_m)$ describes the similarity between the texture features of the sketch T_s and the indexed drawing of the detailed geometry model T_m.

6.2 Location Feature

The user adds detailed geometry (e.g., roof tiles) on top of a primitive geometry (e.g., planar or curved surface) by sketching within the primitive geometry. An example is shown in Figure 14(b); here, the roof tiles (in black) were drawn within a curved surface (in red). The location of the drawn detailed geometry *relative to the primitive geometry* is used as a hint to interpret the sketch of detailed geometry.

Given N_s strokes, the location feature $\mathbf{L} = \{l_0, l_1, ..., l_{N_s}\}$ encodes these relative spatial relationships. We define 11 types of spatial relationships as shown in Figure 14(a). The first four ("AT_VERTEX", "ALONG_EDGE", "ON_EDGE", and "CONNECT_EDGE") describe the placement of a stroke (associated with the detailed geometry) relative to a vertex or edge in the sketched primitive geometry. A stroke is labeled "AT_VERTEX" if the entire stroke is near a vertex (green dashed circle). A stroke is labeled as "ALONG_EDGE" if its two endpoints are near two vertices of an edge e and the stroke is near the edge throughout. A stroke is classified as "ON_EDGE" if the entire length of the stroke is close to a point on an edge. A stroke is labeled "CONNECT_EDGE" if its two endpoints are near two different edges.

The other seven spatial types (with the suffix "_SURFACE") characterize spatial relationships between the stroke and a face of the primitive geometry. (The face was recovered using the technique described in Section 5.) The location of the stroke is classified based on where it is on the plane, as shown in the bottom row of Figure 14(a). The stroke is labeled "ON_SURFACE" if it spans across the 2D region of the face.

Figure 14(b) shows examples of the location relationships between the strokes representing detailed geometries and the interpreted primitive geometries. (Note that only a few representative strokes are labeled for clarity.)

After determining the relationship between each stroke and the reconstructed overall shape of the building, we combine the location feature vector $\mathbf{L}$ with the architectural prior to produce the probability $P(\mathbf{L}, style, |\mathbf{M}, \mathbf{A})$. The prior A defines the occurrence of the detailed geometries based on location in different architecture styles. Combining $P(\mathbf{L}, style, |\mathbf{M}, \mathbf{A})$ with the shape similarity $P(\mathbf{F}_{shape}|\mathbf{F}^{\mathbf{M}}_{shape})$ as formula (4), we finally get the best matched detailed geometry entity with the largest likelihood $P_{detail}(\mathbf{S}|\mathbf{M}, \mathbf{A}, \mathbf{c}_0)$. Example entries in the database of detailed geometries (that associates 2D sketch drawings with 3D models) are shown in Figure 15.

The user can then either choose to use the most likely component or select from the small set of candidates. Once the component has been selected, it is then automatically scaled, rotated, and moved to the target location.

Our system allows the user to easily edit the design. We provide common editing functions such as modifying the geometry, adding, deleting, moving, copying, and pasting components. Furthermore, spatially regular components can also be aligned and made uniform in a single step, which is convenient for refining groups of windows. Figure 16 shows the intermediate steps of creating the rotunda example: from the primitive geometries to detailed geometries.

7. INTERPRETATION OF TEXTURES

Once we have interpreted the geometry of the sketch, the user has the option of assigning textures to surfaces as a finishing touch. In keeping with the spirit of sketch-based input, the user can assign a texture to a surface by directly drawing a likeness of that texture.

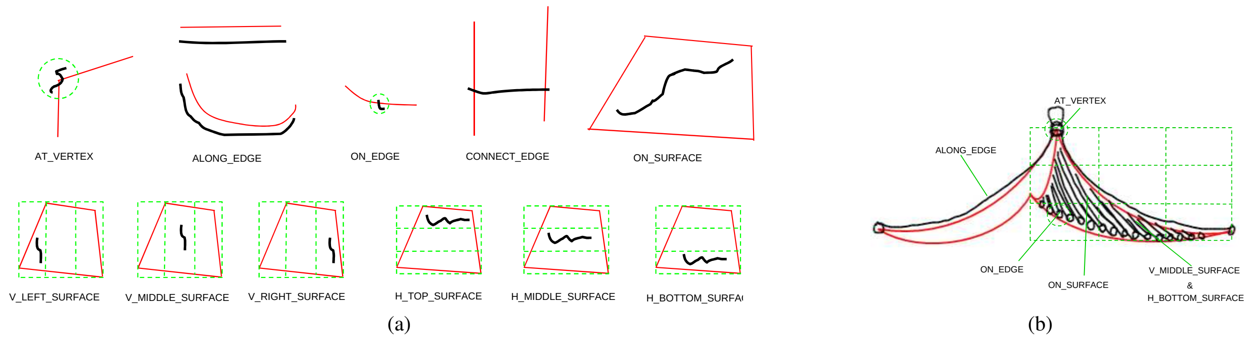


Fig. 14. Use of location as a feature. (a) 11 different types of relationships between a stroke and primitive geometries. The red lines are part of the overall shape of primitive geometries while the black strokes are the current stroke representing part of the detailed geometry. The green dashed lines are used to measure the spatial relationships. (b) An example of the location relationships between the strokes (black) and the primitive geometries (red). See text for details.

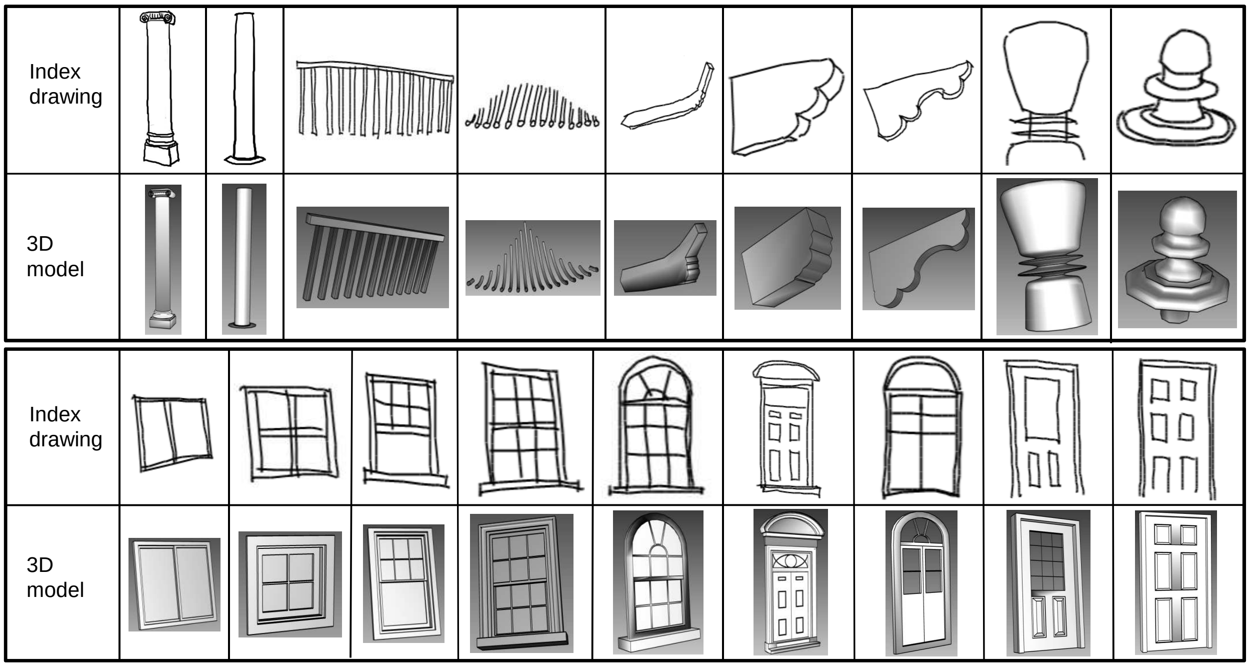


Fig. 15. Examples of 2D index drawings and associated 3D models of detailed geometries in our database.

This feature is made possible using a texture database that associates actual textures to drawn versions. When the user draws a sketch of the texture, our system computes its similarity with those in the database and retrieves the actual texture associated with the best match. Since architects use regular sketches to represent textures, we use the statistic shape features as used in the interpretation of detailed geometry. For texture, the likelihood (1) reduces to the combined histogram distance between the texture strokes drawn by user and the indexing shortcuts in the database:

$$P(\mathbf{h}_s, \mathbf{h}_o, \mathbf{h}_l, style|\mathbf{M}, \mathbf{A})) \propto \frac{P(T_m|style, \mathbf{A})}{1 + d_{all}(T_s, T_m)} \propto \frac{1}{1 + d_{all}(T_s, T_m)}, \tag{6}$$

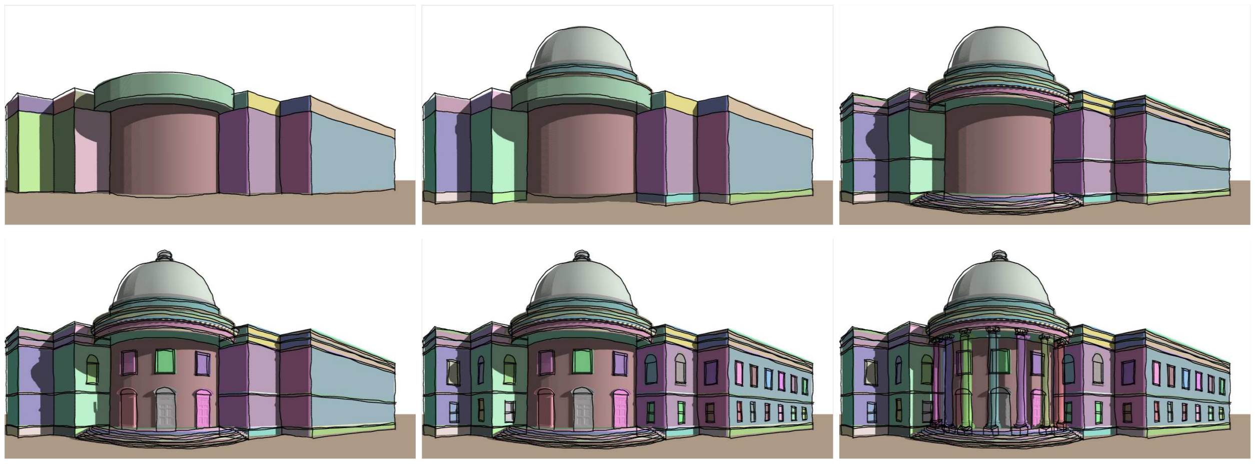


Fig. 16. Progression of sketching and interpreting the rotunda example (left to right, top to bottom).

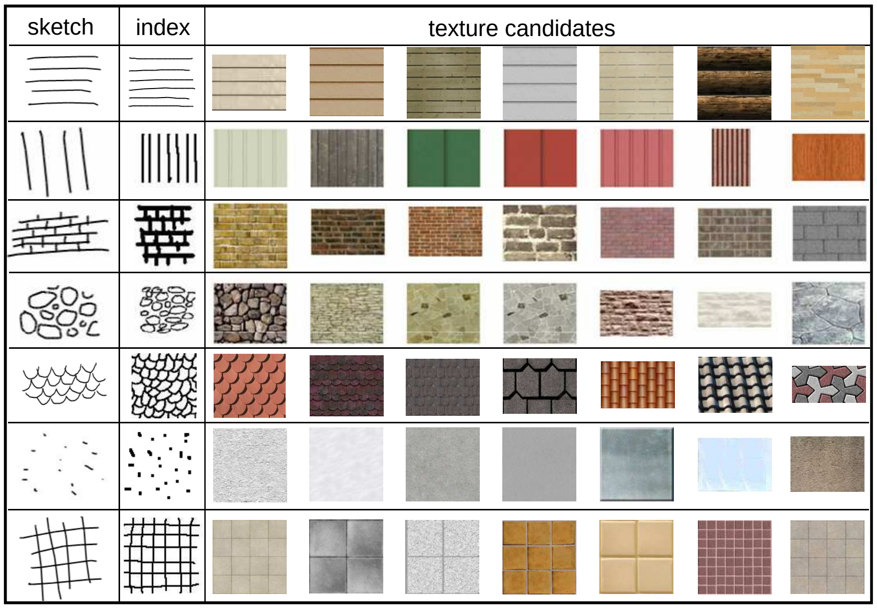

sketch
index
texture candidates

Fig. 17. Examples of indexed sketches of textures to actual textures in the texture database.

with $d_{all}(T_s, T_m)$ defined in equation (5). Generally, there are no significant architectural rules that link texture to style. To simplify our analysis, we set $P(T_m|style, \mathbf{A})$ to be uniform, i.e., we rely entirely on the recognition of the sketched texture. In interpreting regular textures, the shape histogram has largest effect on the decision of texture types while the length histogram has smallest effect. In our implementation, $w_s = 100, w_o = 10, w_l = 1$.

Our system compares the coarse sketches drawn by the user with the 2D index drawings of the textures and recovers the actual texture (or small set of candidates) that maximizes the likelihood (6). Example results are shown in Figure 17. The user can then either use the most likely candidate or select from the small set of texture candidates.

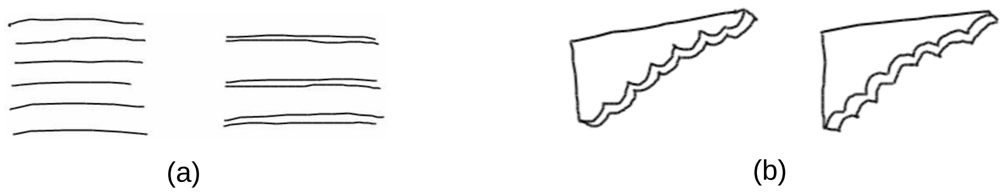


Fig. 18. Example failure cases. (a) Failure for texture, (b) failure for detailed geometry. Each pair has similar stroke type, orientation histogram, and length histogram. However, the spatial distributions of the strokes are different, which results in our failure to tell them apart.

Design	*Time (mins.)*	$N_{strokes}$	$N_{components}$
Rotunda	20	571	116
House	5	160	39
Eastern-style pavilion	7	251	2347
Tall building	10	615	642

Table I. Statistics for the sketches. Notice the large number of geometry components produced for the Eastern-style pavilion, even though the number of user strokes is modest. This is because there are many tiles on the roof, and each tile is considered a component.

By using information on the stroke type, orientation, and length, our system can distinguish many different types of uniform textures. However, our system fails to distinguish textures with non-uniformly distributed strokes as shown in Figure 18(a). This is because our system currently does not gather and use local spatial information. Figure 18(b) shows a similar problem, this time for interpreting detailed geometry. Here, the user has to manually select the correct one from the candidates. The use of local spatial information for retrieval is a topic for future work.

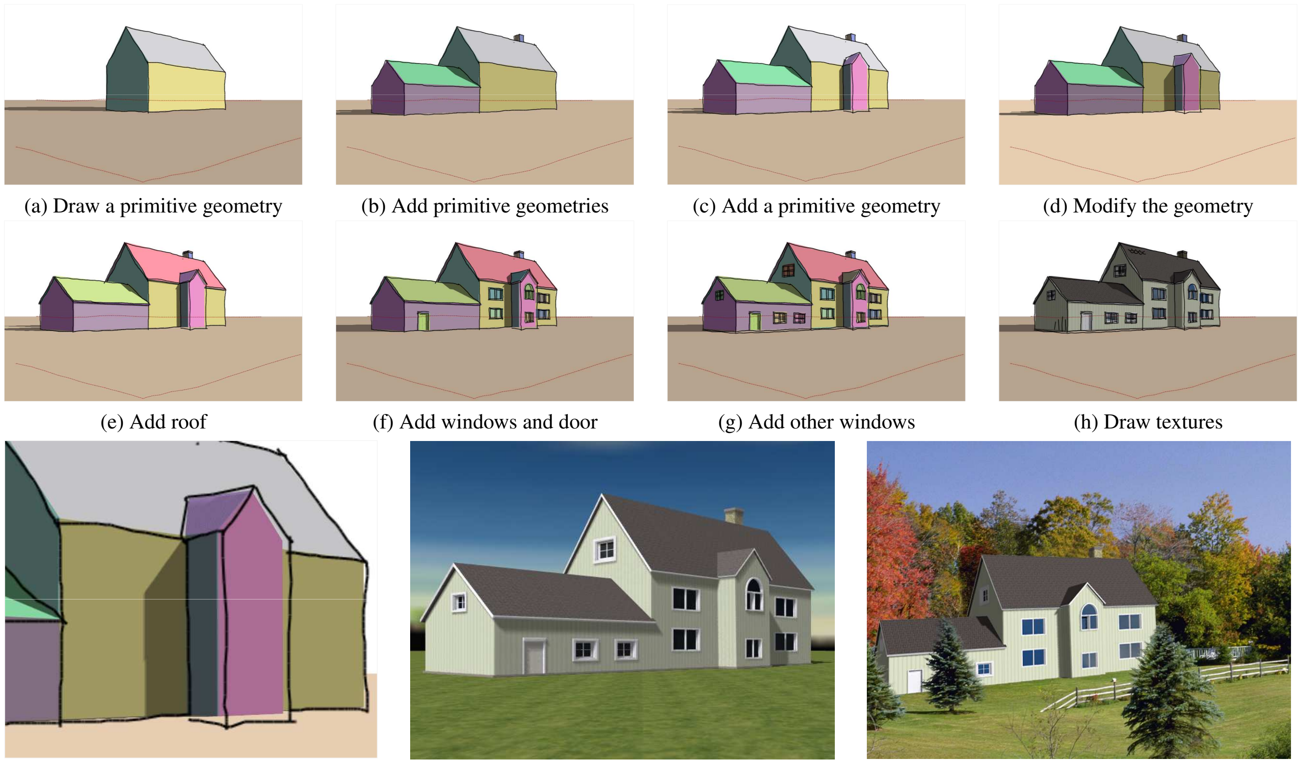
8. RESULTS

All the results were generated on 3.2GHz Pentium 4 PC with 1GB of RAM and a GeForce 6800 XT 128MB graphics card. The sketches were made on a WACOM tablet. Our detailed geometry database consists of 20 categories of detailed geometries, with category consisting of two specific shapes. There are 15 categories of textures in our texture database, with 10 real textures per category on average. (The relatively large number of categories of textures makes it difficult to specify texture just by name, such as “brick” or “stone.”)

Our system allows the user to easily and quickly design by sketching. Our system is geared to those who have a background in architecture. We first asked a licensed architect with 8 years of practical experience in architecture to use our system; he was able to create complicated buildings in minutes, after being trained for an hour. The results for the rotunda are shown in Figure 1. This example shows how a relatively simple sketch in (a) can be easily converted to a much more appealing rendering shown in (c). Three additional examples are shown in Figure 19 (house), Figure 20 (tall building), and Figure 21 (Eastern-style pavilion). In each figure, we show the progression of sketch interpretation and the realistic rendered results. Table I shows how much time it took to produce the examples. Despite the wide range of complexity, all the examples were created through sketching in 20 minutes or less. Table I also lists the number of strokes and geometry components that were generated.

Our system also allows a typical user to create buildings easily by sketching. A modest range from quite coarse to more carefully-drawn sketches can be interpreted in a geometrically-consistent manner. While the architect’s drawings are cleaner than typical ones (Figure 1), we also show two coarser hand-drawn sketches of the overall shape of the rotunda example, one (Figure 22) less-carefully drawn than the other (Figure 23).

As mentioned in Section 5.2, we assign different weights to different types of equations. If the user wishes to emphasize the structure constraint, he may increase the constraint weight w_{con}, while reducing the projection weight w_{proj}. Alternatively, he may increase the projection weight w_{proj} to force the reconstructed geometry to be closer to the drawn coarse sketch. Here we show three different results reconstructed with different projection weights, to illustrate



(a) Draw a primitive geometry (b) Add primitive geometries (c) Add a primitive geometry (d) Modify the geometry

(e) Add roof (f) Add windows and door (g) Add other windows (h) Draw textures

(i) Closeup view (j) Realistic rendering (k) Composed with still image.

Fig. 19. House example. (a-h) Progression of sketch interpretation. The red dash lines represent the amount of perspectiveness. (i) A closeup view. Notice the deviation of the coarse sketch from the geometrically consistent result. (j,k) Final rendering results with different texture, lighting, and background.

the tolerance of our system to input sketches with varying degrees of imperfection. A larger projection weight results in smaller deviation of the geometry from the sketch, while a smaller projection weight leads to a result that better satisfies the structural constraints.

As further demonstration that the user is not required to draw very carefully for our system to work, compare the group of drawn parallel edges and the estimated vanishing point for the sketch in Figure 23. Figure 24(a) shows the estimated parallel lines from its computed vanishing point. Figure 24(a) shows what happens when pairs of edges are independently used to estimate the parallel lines. This bolsters our claim that our system is robust enough to handle some amount of looseness in the sketch.

9. CONCLUDING REMARKS

In this paper, we propose the concept of *sketching reality*, which is the process of mapping a sketch to realistic imagery. We demonstrated this concept by developing a system that takes as input a sketch of an architectural design to produce a realistic 2.5D interpretation. Our system allows the user to sketch primitive and detailed geometries as well as textures (all from a *single* view), and provides a number of editing operations. We are not aware of any system with similar features.

We do not claim that all the features in our system are new; indeed, many of them are inspired by previous work. Rather, we customized all these features to allow them to function together in a single system for the purposes of demonstrating the new concept and illustrating its power.

As a prototype system, it has limitations. Since we use architectural-specific knowledge to guide the sketch interpretation, our system is geared more for users with an architectural background. A casual user would need to be familiar with basic architectural principles to effectively use our system.

A topic for future work relates to curved objects. Currently, symmetry is necessary for our system to reconstruct the curved surface from single view. It would be interesting allow other types of 3D curves or multi-view inputs. Also, our system cannot handle disconnected objects, as this generates relative depth ambiguity. This is an open problem in single-view reconstruction, and unless size priors are given, the user will be required to provide the depth relationship.

While more work is required to ensure that our system is tolerant of large errors and supports a wider range of architectural styles, the set of features presented in this paper is a starting point to illustrate the novel concept of sketching reality. Given the modest set of features, we are already able to produce good results for a variety of different architectural styles. We also plan to expand our system to allow sketch-based designs of other objects, such as cars and furniture.

ACKNOWLEDGMENTS

We would like to thank the reviewers for their valuable feedback and Crystal Digital Technology Co., Ltd. for providing the 3D models of detailed geometries used in our database. We also highly appreciate Lihan Yu's help in designing the buildings used in this paper.

REFERENCES

Alvarado, C. and Davis, R. 2005. Dynamically constructed bayes nets for multi-domain sketch understanding. In *Proc. of Int'l Joint Conf. on Artificial Intelligence*. San Francisco, California, 1407–1412.

Barrow, H. G., Tenenbaum, J., Bolles, R., and Wolf, H. 1977. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 659–663.

Funkhouser, T., Min, P., Kazhdan, M., Chen, J., Halderman, A., Dobkin, D., and Jacobs, D. 2003. A search engine for 3D models. *ACM Trans. Graph. 22,* 1, 83–105.

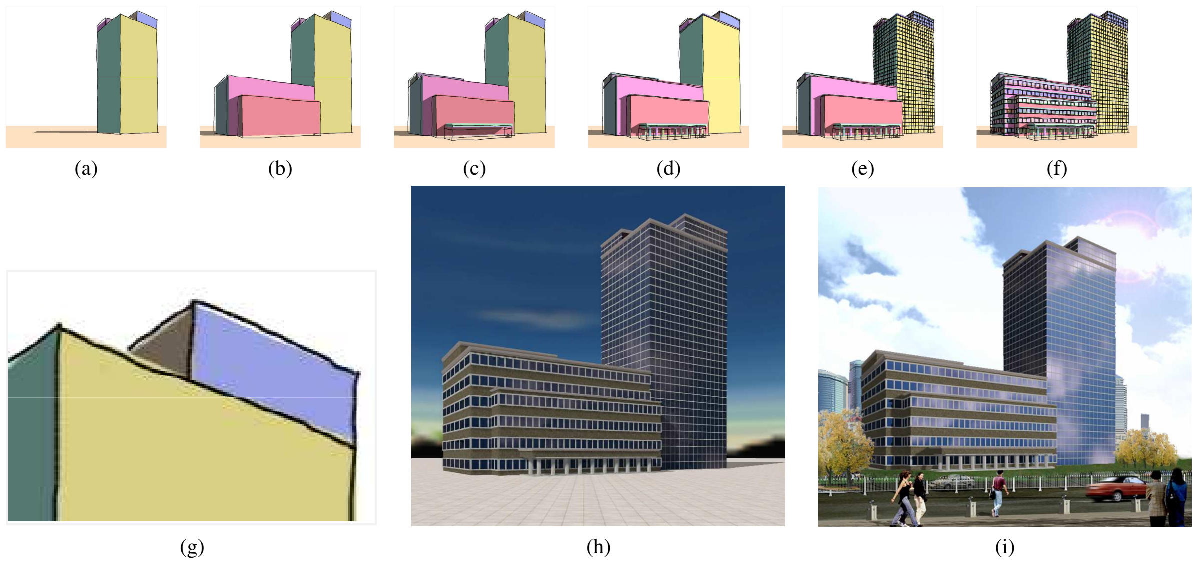


Fig. 20. Tall building. (a-f) Progression of sketch interpretation, (g) a closeup view that shows the deviation of the coarse sketch from the interpreted geometry, (h,i) rendering results with different texture, lighting, and background.

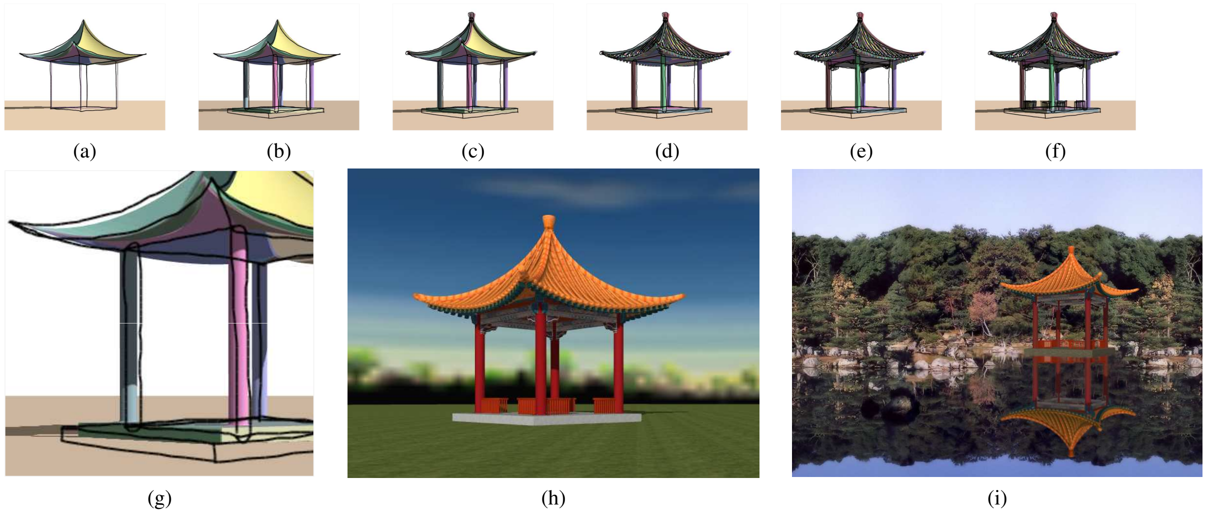


Fig. 21. Pavilion example. (a,b) Overall shape being created, (c-f) detailed geometries being progressively added, (g) a closeup view to show the deviation of the coarse sketch from the interpreted geometry, (h,i) final rendering results with different texture, lighting, and background.

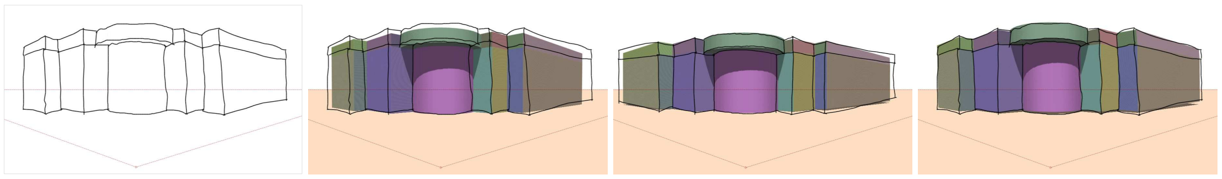


Fig. 22. Input sketch (less carefully drawn). From left to right: Input, result with small projection weight $w_{proj} = 0.1/f$, result with medium projection weight $w_{proj} = 1/f$, result with high projection weight $w_{proj} = 5/f$.

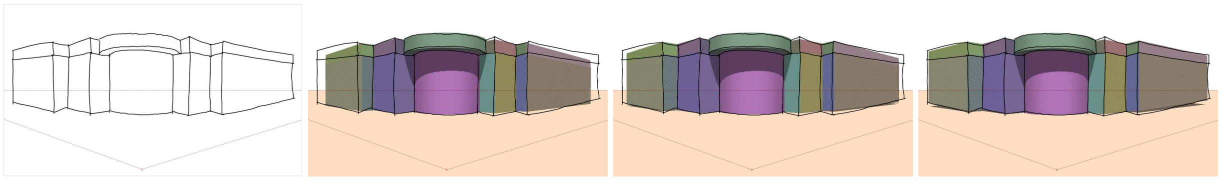


Fig. 23. Input sketch (more carefully drawn). From left to right: Input, result with small projection weight $w_{proj} = 0.1/f$, result with medium projection weight $w_{proj} = 1/f$, result with high projection weight $w_{proj} = 5/f$.

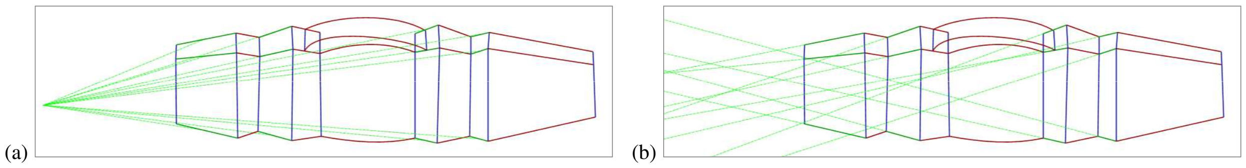


Fig. 24. Comparison between the individually drawn parallel lines and the estimated vanishing point. (a) *Corrected* parallel lines (in green) as a result of estimating the vanishing point. (b) Result of computing the vanishing point independently for each pair of the parallel lines. The lines in (b) do not intersect at a unique location, illustrating that the sketch is indeed only coarsely drawn.

GENNARI, L., KARA, L. B., STAHOVICH, T. F., AND SHIMADA, K. 2005. Combining geometry and domain knowledge to interpret hand-drawn diagrams. *Computers and Graphics*, 547–562.

GRIMSTEAD, I. J. AND MARTIN, R. R. 1995. Creating solid models from single 2D sketches. In *Proc. of Symp. on Solid Modelling and Applications*. 323–337.

HONG, W., MA, Y., AND YU, Y. 2004. Reconstruction of 3D symmetric curves from perspective images without discrete features. In *European Conference on Computer Vision*. Vol. 3. 533–545.

HOU, S. AND RAMANI, K. 2006. Sketch-based 3D engineering part class browsing and retrieval. In *EuroGraphics Workshop on Sketch-Based Interfaces and Modeling*. 131–138.

HUTTENLOCHER, D. P., KLANDERMAN, G. A., AND RUCKLIDGE, W. J. 1993. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence 15,* 9 (September), 850–863.

IGARASHI, T. AND HUGHES, J. F. 2001. A suggestive interface for 3D drawing. In *14th Annual Symposium on User Interface Software and Technology, ACM UIST'01*. 11–14.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3D freeform design. In *Proc. of ACM SIGGRAPH*. 409–416.

JATUPOJ, P. 2005. Sketchboard: The simple 3D modeling from architectural sketch recognition. In *Computer Aided Architectural Design in Asia*.

KARPENKO, O. A. AND HUGHES, J. F. 2006. Smoothsketch: 3D free-form shapes from complex sketches. *Proc. of ACM SIGGRAPH and ACM Trans. on Graphics 25,* 3, 589–598.

LEE, H.-J. AND CHUNG, J.-H. 1999. Hand gesture recognition using orientation histogram. In *Proc. of IEEE Region 10 Conf.* Vol. 2. 1355–1358.

LIPSON, H. AND SHPITALNI, M. 1996. Optimization-based reconstruction of a 3D object from a single freehand line drawing. *Journal of Computer Aided Design 28,* 8, 651–663.

MASRY, M., KANG, D., AND LIPSON, H. 2005. A freehand sketching interface for progressive construction of 3D objects. *Journal of Computer and Graphics, Special Issue on Pen-Based User Interfaces 29,* 4, 563–575.

MATSUDA, K., SUGISHITA, S., XU, Z., KONDO, K., SATO, H., AND SHIMADA, S. 1997. Freehand sketch system for 3D geometric modeling. In *SMA '97: Proceedings of the 1997 International Conference on Shape Modeling and Applications (SMA '97)*. IEEE Computer Society, Washington, DC, USA, 55–63.

MULLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND GOOL, L. V. 2006. Procedural modeling of buildings. *Proc. of ACM SIGGRAPH and ACM Trans. on Graphics 25,* 3, 614–623.

PARISH, Y. I. H. AND MULLER, P. 2001. Procedural modeling of cities. In *Proc. of ACM SIGGRAPH*. 301–308.

PRASAD, M., ZISSERMAN, A., AND FITZGIBBON, A. 2006. Single view reconstruction of curved surfaces. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*. Vol. 2. 1345–1354.

PU, J. AND RAMANI, K. 2006. On visual similarity based 2D drawing retrieval. *Computer-Aided Design 38,* 3, 249–259.

ROTHER, C. 2000. A new approach for vanishing point detection in architecture environment. In *Proc. of British Machine Vision Conf.* 382–391.

SEZGIN, T. M., STAHOVICH, T., AND DAVIS, R. 2001. Sketch based interfaces: Early processing for sketch understanding. In *Proc. of Workshop on Perceptive User Interface*.

SHESH, A. AND CHEN, B. 2004. SMARTPAPER–An interactive and easy-to-use sketching system. In *Proc. of Eurographics*.

SHILMAN, M., PASULA, H., RUSSELL, S., AND NEWTON, R. 2000. Statistical visual language models for ink parsing. In *AAAI Spring Symp. on Sketching Understanding*. 126–132.

SHPITALNI, M. AND LIPSON, H. 1996. Identification of faces in a 2D line drawing projection of a wireframe object. *IEEE Transactions on Pattern Analysis and Machine Intelligence 18,* 10, 1000–1012.

VARLEY, P. A. AND MARTIN, R. R. 2000. A system for constructing boundary representation solid models from a two-dimensional sketch. In *Proc. of Geometric Modeling and Processing*. 13–30.

WONKA, P., WIMMER, M., SILLION, F., AND RIBARSKY, W. 2003. Instant architecture. *Proc. of ACM SIGGRAPH and ACM Trans. on Graphics 22,* 3, 669–677.

ZELEZNIK, R. C., HERNDON, K. P., AND HUGHES, J. F. 1996. SKETCH: An interface for sketching 3D scenes. In *Proc. of ACM SIGGRAPH*. 163–170.

...