Generating 3D Building Models from Architectural Drawings: A Survey

Xuetao Yin, Peter Wonka, and Anshuman Razdan ■ *Arizona State University*

Automatically generating 3D building models from 2D architectural drawings has many useful applications in the architecture engineering and construction community. This survey of model generation from paper and CAD-based architectural drawings covers the common pipeline and compares various algorithms for each step of the process.

Using 3D building models is extremely helpful throughout the architecture engineering and construction (AEC) lifecycle. Such models let designers and architects virtually walk through a project to get a more intuitive perspective on their work. They can also check a design's validity by running computer simulations of energy, lighting, acoustics, fire, and other characteristics and thereby modify or adjust designs as needed before construction begins. 3D building models also have far-reaching applications beyond AEC, such as real estate, virtual city tours, and video gaming. However, manually creating a polygonal 3D model of a set of floor plans is nontrivial and requires skill and time.

Researchers and CAD developers have been trying to automate and accelerate conversion of 2D drawings into 3D models, but doing so is difficult for several reasons. Foremost among these is the input form, which greatly determines how complicated it will be to extrude a model from architectural drawings. Some systems use digital copies of computer-drawn architectural drawings; others scan paper floor plans as input. However, because paper plans still dominate the architectural workflow, any system that claims to be an end-to-end solution must process raster images.

Although existing solutions share a common pipeline, they often choose different algorithms for various process steps. In this survey, we review the research on automatic generation of 3D building models from both paper- and CAD-based architectural drawings. Besides comparing the systems' robustness and efficiency, we suggest improvements and offer a brief review of industry products.

Architectural Floor Plans

Architectural drawings are essential to designing, narrating, and executing a construction project. Most drawings take the form of floor plans, which portray an orthographic top-down projection of each building level using standardized symbolic representations of the structure's architectural elements. Other kinds of drawings—such as longitudinal-section drawings, elevation drawings, and reflective ceiling plans—work with floor plans to form a complete building specification.

Floor plans have various levels of detail. The most punctilious and intricate floor plans are detailed workplans or *construction structure drawings* (CSDs). CSDs are used exclusively by design engineers and construction managers and often show internal steel bars, the concrete structure for columns, beams and walls, and pipe and ductwork layouts. Recently, Tong Lu and his colleagues designed a system that constructs a detailed building model from computer-drawn CSDs.[1] However, to our knowledge, no research has aimed at interpreting raster images of CSDs.

The most widely distributed form of floor plans lacks detailed construction information. Still, they manage to cover the building's complete layout, which is sufficient to build a model for most applications. Whether these less-detailed floor plans are hand drawn or computer produced, many systems

 Published by the IEEE Computer Society

Figure 1. Different ways to draw a wall with a window and a door. The variable graphic symbols pose challenges for automatically converting 2D drawings into 3D models.

(a) **(b)**

Figure 2. The (a) input and (b) output of a system that converts 2D architectural floor plans to 3D computer graphics models. Systems that accept floor plans as input must rely on image-processing and pattern-recognition techniques to distinguish between graphical symbols, wall lines, dimensions, and so on.

accept them as legitimate input. However, such floor plans use varying graphic symbols, which is a major drawback.

Figure 1 shows examples of common styles for walls, windows, and doors. Instead of being constrained to a particular standard, a drawing's purpose (and the designer's artistic motivation) determines what components will be shown and how they'll look. This creates a major challenge in analyzing and interpreting an image floor plan, and makes a certain amount of human intervention unavoidable.

General System Overview

Figure 2 shows an example input and the desired output from an automated 3D building model system. We categorize existing systems according to the kind of input they use. CAD documents, such as Data Exchange Format (DXF) and AutoCAD Drawing (DWG) files, preserve drawing information as 2D geometric primitives, grouping the architectural components together by type and giving them unique labels. This layered structure of CAD files makes recognition trivial.

In contrast, when a system takes a raster image of a floor plan as input, there's no obvious distinction between graphical symbols, wall lines, dimensions, scales, textual content, and leading lines (that is, the straight lines that lead to measurement or text). So, to decipher the information needed for extrusion, the system must rely on image-processing and pattern-recognition techniques.

Figure 3 (next page) shows the basic model extrusion steps, and Figure 4 (page 23) shows an ideal solution's two-phased operational pipeline. Most actual systems differ slightly from this model. However, by combining different system ideas, our common framework can help developers structure and compare existing solutions. As we discuss later, systems generally differ in the choice of algorithms or the task execution order.

Besides general characteristics, most systems also share common shortcomings. The biggest is the lack of generality. Pattern recognizers are typically constrained to a small set of predefined symbols. Also, current systems don't exploit information embedded in text strings, which could be a valuable cue to the building's spatial structure and topology. Most systems also neglect the "finishing touches." To offer a better visual appearance, for example, a system could either procedurally generate indoor and façade textures or automatically derive them from photographs. In addition, systems fail to appropriately orient the architectural elements' placement in the 3D model. Finally, several systems use imperfect algorithms, thus requiring substantial user assistance in some steps. Systems

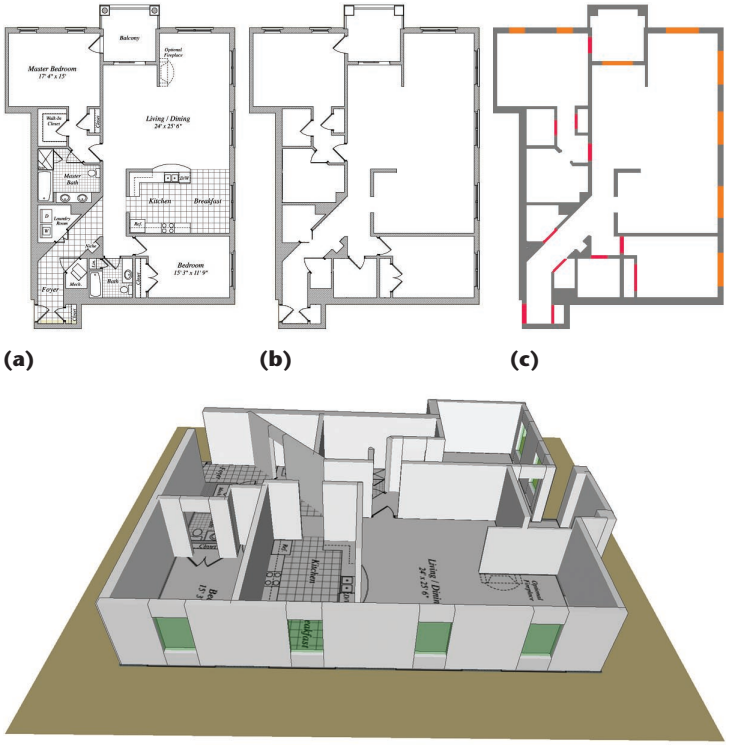

Figure 3. Critical steps in 3D model extrusion. The system takes as input (a) an original floor plan. It then uses algorithms for (b) denoising and text removal and (c) symbol recognition and 2D geometry creation. Finally, it (d) extrudes a 3D model.

need more accurate, efficient, and automated algorithms, especially for the pipeline's first phase.

Converting Floor Plan CAD files

Systems using CAD-based floor plans don't have the overhead or ambiguities related to image processing and pattern recognition; they focus more on 3D model extrusion. University of California, Berkeley researchers Rick Lewis and Carlo Séquin introduced a system that semiautomatically creates detailed 3D polygonal building models using floor plans created in AutoCAD.[2] The system groups architectural symbols into dedicated layers in standard DXF files. Although this simplifies the recognition algorithm's task, the geometry typically suffers from errors and ambiguities, especially at the joint regions. The system deals with geometric flaws by correcting disjoint and overlapping edges. During the extrusion phase, it collects the topology of spaces and portals and thereby guarantees proper polygon orientation. After it has modeled each floor, the system stacks the floors to form the complete model. With embedded topology, designers can use the resulting models for various applications, such as smoke propagation simulation. The system is highly automated but requires user assistance to correct geometry flaws.

Clifford So and his colleagues at the Hong Kong University of Science and Technology (HKUST) view the model conversion problem in the VR context.[3] Architecture and urban design are a significant market for VR techniques, so automating model generation is extremely beneficial. After observing conventional manual model reconstruction, the authors identified its three major tasks: wall extrusion, object mapping, and ceiling and floor construction. They then incorporated automated approaches to each, including automatic wall polygon extrusion, generating and placing customized templates of random orientation and size, and advancing front triangulation. This greatly reduces processing time.

However, this approach still requires considerable manual effort: users must mark-up wall lines, specify architectural objects, and assign the objects to individual transformation matrices. Consequently, for the system to perform adequately, the input file must contain fully established semantic information and be error free.

With the Building Model Generation (BMG) project (http://city.csail.mit.edu/bmg), Massachusetts Institute of Technology researchers set out to fully automate construction of a realistic MIT campus model. The project's pipeline is similar to that of the UC Berkeley system but attaches an extra process to automatically position and orient building models using a map for guidance.

Lu and his colleagues at China's Nanjing University developed systems to construct models from computer-drawn CSDs[1] and vectorized floor plans.[4] Unlike a computer-produced drawing, a vector image contains geometric primitives without labels to indicate their types, making symbol recognition much more difficult. As in the HKUST project, this system also differentiates the walls from other architectural components. It detects parallel line-segment pairs as walls and removes them from the drawing. It recognizes the remaining primitives as different symbols by finding feature matches with predefined patterns. Each pattern contains a target symbol's graphical primitives and corresponding geometric constraints, as well as integrated information about its context in the drawing (environment). During recognition, the system orders pattern constraints by their priority level and checks them one at a time. It removes corresponding primitives from the drawing immediately after satisfying all of a pattern's constraints. The system pays significant attention to a building model's structural details. The processes are highly automated once the user imports all the patterns. However, the system's robustness is highly sensitive to input quality.

Converting Floor Plan Images

CAD tools are a relatively recent development in architectural history. Many drawings are still done on paper and saved as scanned images. Such images can't be input in the systems we just described. Before model extrusion, scanned images must be converted to properly structured CAD documents or something semantically equivalent. Doing this manually is labor intensive and time consuming, even with a moderate number of plans.

Philippe Dosch and his colleagues at France's Lorraine Laboratory of Research in Information Technology and Its Applications (Loria) proposed a complete solution for analyzing raster images and generating 3D models.[5] The system contains three major steps.

1. During image processing and feature extraction, the system vectorizes input raster images as sets of polylines and arcs. It supports large images through integrated tiling and merging processes.
2. During 2D modeling, the system uses constraint networks[6] to recognize vector elements as architectural symbols and integrates them into a description of the building layout.
3. During 3D modeling, the system separately extrudes a 3D model of each floor and assembles them to form the entire building.

This system addresses almost all pipeline issues. It recognizes one of the largest symbol sets of all the studied systems and demonstrates maturity and robustness in steps such as image processing and model extrusion. The system requires moderate human assistance; some intervention remains unavoidable for steps such as arc detection in vectorization and symbol recognition.

At the Chinese University of Hong Kong (CUHK), Siu-Hang Or and his colleagues developed a system to solve a slightly simplified problem that considers only walls, doors, and windows.[7] The overall execution flow is similar to the Loria system but emphasizes 3D-model extrusion. CUHK's system distinguishes walls as inner structures from building outlines, which it uses to match neighboring floors. (In contrast, the Loria system uses intrusion structures—such as elevator wells—to guide matching.) During vectorization, the system extracts outlines of black pixels in the raster image and matches them with walls of various shapes. It identifies symbols by matching vector-primitive groups to patterns consisting of sequences of geometric characteristics (constraints).

Although using a simple symbol recognizer simplifies the system, it limits its flexibility and applicability. Recognition quality relies heavily on the vectorization algorithm's robustness. To improve performance, the developers introduced a raster image denoising process. Before extruding the 3D model, the system identifies rooms as enclosed spaces, which provides useful information for downstream analysis and applications.

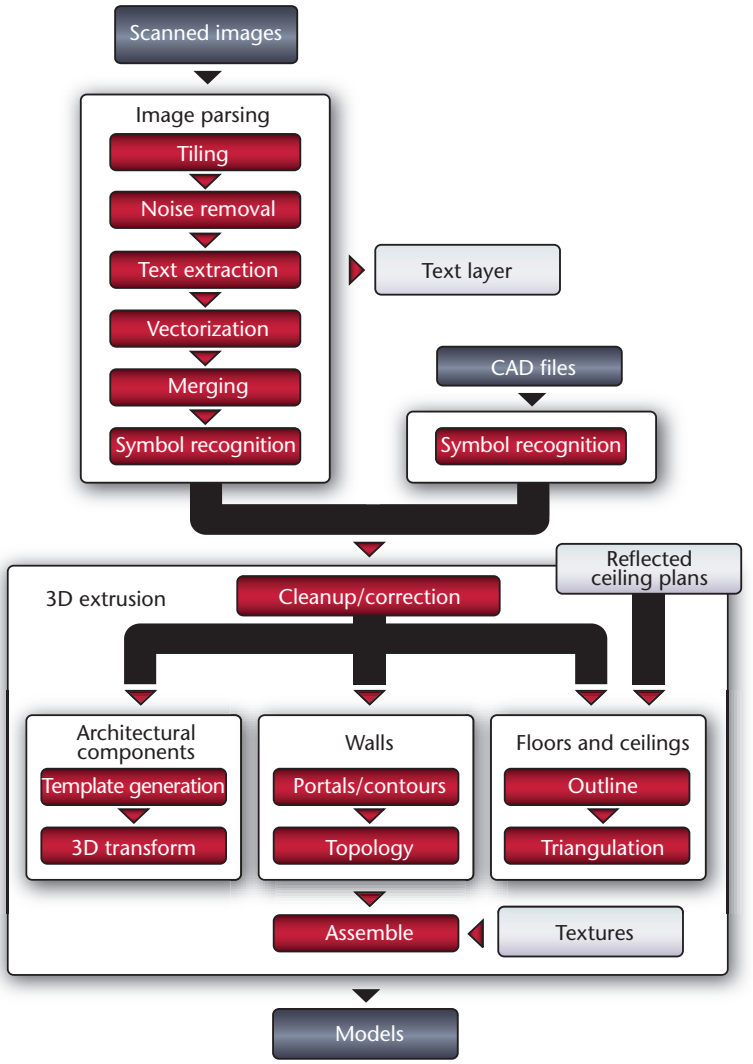

Figure 4. The pipeline for a complete solution. This idealized pipeline combines ideas from various systems to create a framework that can help developers structure their own solutions or compare existing solutions.

System Comparison

Table 1 (next page) summarizes the systems and their processes and relates them to Figure 3's pipeline. As the table shows, each system comprises a unique set of processes. Combining them offers a complete pipeline that covers all aspects of model generation from floor plans.

Some processes, such as image denoising and topology construction, are mature and effective; others are ineffective or not robust enough for fully automated execution. For systems using raster images, symbol recognition is a bottleneck. The graphical symbols' flexible nature and subtle shape differences make achieving satisfactory precision difficult. For both system categories, correcting geometry flaws

Table 1. Comparing the systems on image parsing, 3D extrusion, and overall performance.

Process	Univ. of California, Berkeley	Hong Kong Univ. of Science and Technology	Nanjing Univ.	Loria	Chinese Univ. of Hong Kong
Image parsing					
Tiling/merging				Automatic	
Noise removal				Not specified	Semiautomatic with filtering and manual intervention
Text extraction				Pixel-based	Statistical techniques
Vectorization				Skeletonization and polygon approximation	Outline extraction
Symbol recognition	Automatically collected from Data Exchange Format (DXF) layers	Manually collected	A sequence of geometric and other constraints	A constraint network	A sequence of geometric constraints
3D extrusion					
Cleanup	Automatic		Manual	Manual	
3D transformation		Semiautomatic			
Portals/contour/ topology	Automatic	Manual	Automatic		Automatic
Outline	Automatic	Manual	Automatic		Automatic
Triangulation	Polygon-based	Advancing front			
Assembly	Can handle different-size floors			Uses intrusion structures to match adjacent floors	
Overall performance					
Automation	High	Low	High	High	High
Robustness	High	High	Medium	High	Medium

without human intervention also remains difficult. An optimal approach would integrate complementary systems and employ more recent and advanced algorithms in some of the key pipeline processes.

To build an ideal system, we must synthesize various processes. To that end, we now introduce the existing systems' algorithms for the two main pipeline phases—image parsing and 3D model extrusion—and briefly explore other choices.

Image Parsing and Drawing Analysis

This phase aims to analyze an input raster floor plan and extract the layout information it represents. In other words, the goal is to parse the floor plan's architectural semantics. As Table 2 shows, this phase features several major challenges.

To analyze and parse image floor plans, systems rely on graphical document analysis. This typically involves two major steps: cleaning and graphical-symbol recognition (also known as graphics recognition). Cleaning aims to remove noise and unnecessary information from the image to improve graphics recognition quality. In graphical-symbol recognition, the system groups neighboring pixels and interprets them as instances of graphical symbols. The system collects and organizes each recognized symbol's location, orientation, and scale information.

As an outcome of floor plan analysis, designers expect an object-orientated geometrical description of the floor's architectural layout. Floor plans differ from other graphical documents in several ways. One is the presence of lines that represent walls; such lines can be large spans, be straight or curved, and have varied shapes. Another difference is the presence of highly localized architectural symbols composed mostly of simple geometric primitives. Typically, graphics recognition would incorporate vectorization to deal with this type of input. In all the systems, the overall analysis process starts with cleaning (including noise removal and text extraction), followed by vectorization and recognition.

Noise Removal

In scanned images, one of the most common types of noise is sampling noise introduced by digital scanning. This is a well-studied problem in image processing, and researchers have proposed many algorithms to solve it.

However, in floor plan analysis, noise has a broader definition. In addition to scan noise, de-

signers consider all pixels that lack information directly useful for model generation as noise. Examples include annotation leading lines; dimension lines; furniture and hardware symbols, such as for tubs and chairs; and, in some cases, decorative patterns in the background. Designers also consider text strings as noise, although they typically use dedicated algorithms to deal with them.

Sometimes, there's a fine line between noise and useful pixels, and segmentation remains an unsolved problem in floor plan image analysis. The Loria system uses morphological filtering to segment an image into thick lines against thin lines. This approach assumes that background patterns and dimension leading lines differ from useful lines in thickness and style. Other researchers also make this assumption,[7] putting a threshold on the input that preserves only thick construction lines. For all such systems, however, human intervention in this step is unavoidable.

Text Extraction

The ideal algorithm should be not only efficient but also independent of the text font, size, and orientation and should require minimal human intervention. Text intermingled with the geometric shapes poses additional challenges in terms of separation and extraction. Researchers have studied text separation in detail. Most of the resulting algorithms fall into two families. Structure-based (or curvature-based) algorithms focus on the structural differences between graphical symbols and characters. These algorithms are inspired by the idea that a character is always more structurally complex than a graphical symbol. By separating all linear shapes—using approaches such as directional morphological filtering[8] or distance transform[9]—these algorithms separate graphics content from characters.

The second algorithm family is pixel based.[10,11] For example, Lloyd Fletcher and Rangachar Kasturi presented an algorithm[10] that researchers have used in various document analysis systems, either directly or by adjusting input characteristics. The algorithm first collects black pixels (eight connected pixels) and encloses their circumscribing rectangles as a single connected component. Next, it filters connected components through several metrics to be either rejected or accepted as part of text strings. (Attributes include size, black-pixel density, ratio of dimension, area, and position within the image.)

For complex drawings, pixel-based algorithms are more stable than structure-based algorithms. Pixel-based algorithms work extremely well when text and graphics don't touch or overlap. However, they will likely classify dashed lines as characters.

Table 2. The challenges of image parsing and drawing analysis.

Step	Issues
Noise removal	The leading lines of notations could be easily confused with wall lines. The background might contain a grid or decorative pattern.
Text extraction	Text font, size, and orientation might vary. Text and graphical symbols might share pixels (overlapping or touching). Many algorithms classify dashed lines (commonly used in the staircase symbol) as text.
Vectorization	Most algorithms recover only lines and arcs. Free-form curves continue to be a challenge. Noise greatly affects the result. Vectorization might give bad results at junction points.
Symbol recognition	The symbols might not comply with the standards. There might be a large pool of symbols, and differences between two symbols could be subtle.

Because dashed lines often denote staircases and hidden structures, postprocessing must reclaim them as graphical symbols.

In their system, the Loria researchers implemented the algorithm[10] with a postprocessing step for dashed lines. They later made the algorithm more suitable for graphics-rich documents.[11] Their improvements included a postprocessing step that uses local segmentation of the distance skeleton to retrieve text components that touch graphics.

Systems typically fail to adequately exploit the text layer because adding this functionality makes the system more complex. However, text string size, location, and orientation can provide important clues about a building's structure even when the semantic meanings are unknown. For example, the label "patio" for a part could be very informative about the space that part represents in the drawing.

Graphics Recognition

Once the system separates text from graphics, it must extract the pixels' embedded architectural information and organize the pixels into a complete object-based geometrical description of the building layout.

A drawing contains two major kinds of information:

- structural information, represented by walls, and
- local architectural components (or accessories), represented as parameterized instances of standard templates.

Architectural design is essentially a partition of space, and walls define the building's spatial structure. Walls are therefore better preserved as geometric

polylines for the extrusion step. This is one reason all systems incorporate vectorization and work on geometric primitives rather than perform symbol recognition based directly on pixels.

Vectorization. This process, also called raster-to-vector conversion, transforms image pixels to the geometric primitives they represent. Theoretically analyzing a vectorization algorithm is nontrivial. Such an algorithm's most important criteria are efficiency, robustness, and accuracy.

Traditional line-drawing vectorization includes two steps:[12]

1. The raster-to-chain step converts the raster bitmap to a set of pixel chains.
2. The chain-to-segment step transforms the set of pixel chains to polylines or arcs.

After each step, various postprocesses are needed to fix joint errors. However, most vectorization algorithms find only line segments and circular arcs. Algorithms for more complex curves are rare.

For the first step, systems typically use three groups of algorithms: parametric model fitting, contour tracking, and skeletonization.[12] Parametric model fitting uses a Hough transform to detect lines in the image. This method's disadvantage is huge memory consumption and the lack of generality.

Contour tracking works especially well for simple floor plans. Instead of dealing with black pixels, this algorithm searches the contour of white pixels and identifies connected regions as rooms on the basis of the assumption that, in floor plans, white spaces are partitioned by black wall lines. This method doesn't work when the structure gets complicated; it's also sensitive to noise.

Skeletonization finds a curve's bones, or skeleton, by thinning or by searching for its medial axis.[13] Thinning-based algorithms iteratively peel off boundary pixels until only a one-pixel-wide skeleton remains.[14] However, these algorithms can give bad results at intersections, especially when distortions exist. Also, they aren't very efficient because they visit each pixel multiple times. Typical medial-axis-based algorithms include pixel tracking[15] and run-graph-based algorithms.[16] Medial-axis-based algorithms treat a line with thickness as a solid shape, with the medial axis as its skeleton. The medial axis of a 2D polygonal shape is defined as the locus of the centers of all inscribed spheres of maximal radius.

Vectorization's second step segments point chains into sets of lines, polylines, and circular arcs by using polygonal approximation or estimating the curvature to find the critical points.

Loria's system uses a skeletonization technique for vectorization's first step and polygonal approximation for the second step. Similarly to contour tracking, the CUHK system tracks the contour of black pixels and organizes them into blocks of walls and symbols.

Symbol recognition. This process is at the core of graphical document analysis. The ideal graphic symbol recognizer (GSR) is efficient, robust, independent of context, and immune to affine transformation. Several existing methods work well in particular areas and offer a satisfactory performance overall.

Most GSRs are either vector based (oriented toward structure) or pixel based (oriented toward statistics). Vector-based GSRs work on vectorized images composed of primitives such as points, line segments, arcs, and circles. The GSR identifies a symbol by checking the structural characteristic of a group of neighboring primitives. Vector-based approaches include region adjacency graphs,[17] graphical-knowledge-guided reasoning,[18] constraint networks,[6] and deformable templates.[19] Such approaches require good vectorization; their advantage is that they're affine invariant.

Pixel-based GSRs work directly on raster images, focusing on statistical features of a symbol's pixel formation. Pixel-based algorithms include plain binary images,[20] living projection, and shape contexts.[21] Because this approach doesn't involve vectorization, it has higher precision and accuracy than vector-based approaches, but its performance is vulnerable to scaling and rotation. Su Yang has been working to merge the vector- and pixel-based approaches.[22]

Because (as we discussed earlier) all existing systems use vectors as GSR input, they implement GSRs using structural approaches. The Loria project uses constraint networks, which view a symbol as a set of constraints that the vectorized image's primitives must fulfill. This approach uses a network to model the features and constraints, and propagates vectorized floor plan segments through the network to search for terminal symbols. CUHK's system adopts a similar, but simpler, approach that uses a sequence of geometric constraints as symbol patterns. Systems could use both raster and vector copies of a given floor plan and use both approaches to increase recognition precision.

The International Association on Pattern Recognition's Workshop on Graphics Recognition has held several international symbol recognition contests, the most recent occurring in 2005. Ernest

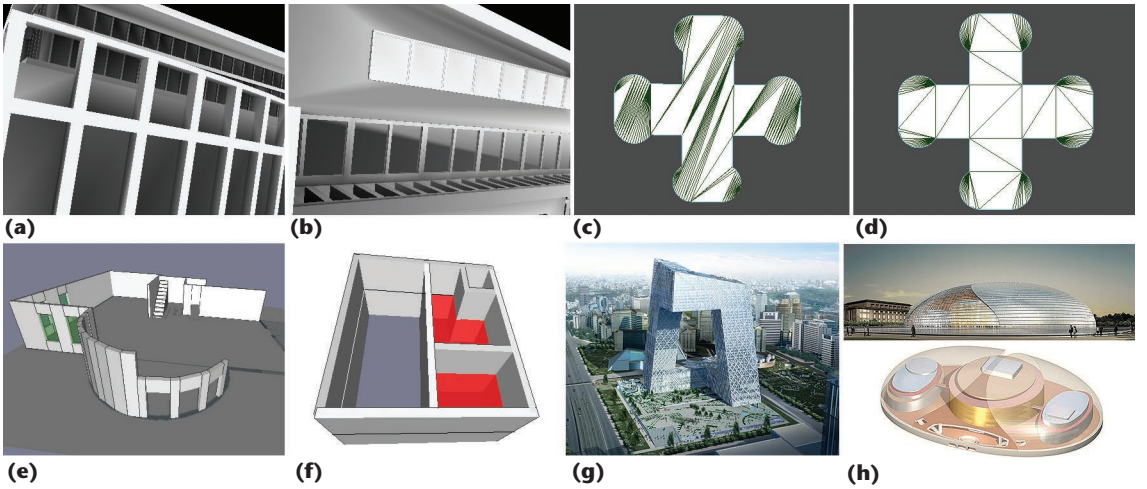

Figure 5. Models with inconsistent normals are unacceptable for many applications. (a) The rendering result of a model with correct normals using the ambient-occlusion technique. (b) Ambient occlusion on a model whose normals aren't consistent. (c) A low-quality triangulation of a cross-shaped building outline with too many slim triangles. A smarter tessellation would use a convex polygon with high edge count. (d) The same shape tessellated by a constrained Delaunay triangulation. Polygons in (c) and (d) have the same number of triangles; however, (d) has much better quality with respect to the triangles' shape. Ambiguity is introduced by (e) a projection object or (f) a penetrating structure, such as an atrium or lobby. Modern architectures place higher demands on system flexibility and intelligence. Examples of unconventional designs include (g) China Central Television's new headquarters (under construction) and (h) the complex inner structure of Beijing's National Center for the Performing Arts. (Figure 5g courtesy of the Office for Metropolitan Architecture; Figure 5h courtesy of the Artists Rights Society.)

Valveny and Philippe Dosch describe the different algorithms submitted to the workshop and how they performed under various conditions.[23]

Tiling and Merging

A drawing's size can also create issues. Sometimes, the image file might be prohibitively large. Although users can overcome this obstacle by reducing the image's scale, such downsampling can cause information loss. This is where tiling comes in handy. Tiling tessellates the original input into smaller parts, processes them individually, and merges them back together.

Dosch and his colleagues, for example, split the original image into partially overlapping tiles, carefully selecting the width of overlapping zones to achieve maximum performance.[5] After vectorization, they merge the tiles by matching the vector content in neighboring tiles. This highly automated procedure requires minimal user interaction and reportedly has a low error rate.

3D Model Extrusion

The input to the pipeline's model extrusion phase can be either a geometric and component-wise building layout description from the image-parsing phase or a well-organized CAD document with a dedicated layer for each symbol type. The goal is to automatically create a 3D building model in the form of a polygonal mesh.

Model extrusion entails six major challenges:

- The extrusion should consistently orient facet normals.
- Creating details of architectural entities relies heavily on empirical assumptions. Any template library that tries to cover all styles and designs will inevitably be huge and have potentially conflicting architectural styles.
- Assembling multiple building levels to form a complete building can be problematic because individual floor plans might use different scales and orientations.
- The search for exterior outlines can be complicated if the exterior walls have projecting objects (such as balconies).
- The selected approach must accommodate buildings with unconventional designs.
- Extrusion can be complicated when buildings have multiple stories. If two adjacent floors have different footprints, one floor might be exposed. To avoid gaps, the model must incorporate additional polygons.

Figure 5 illustrates some of these challenges.

Error Cleanup

Both vectorized, hand-drawn images and computer-sketched drawings suffer from disjointed lines, overlapping vertices, and false intersections.

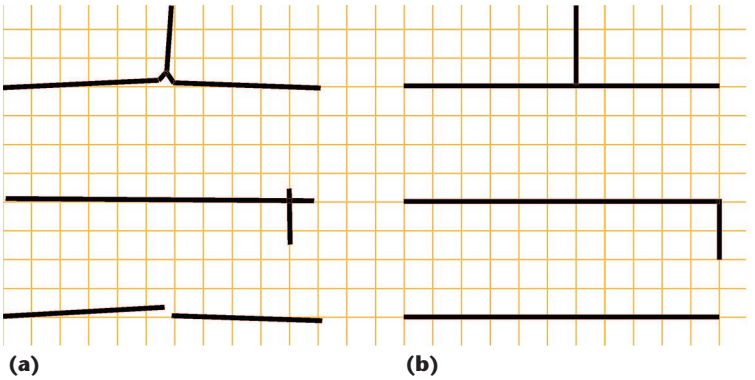

Figure 6. Dealing with connectivity errors: (a) three common errors and (b) the correct joints using the coerce-to-grid algorithm.

Before working with polygons, designers must launch certain operations to clean up geometry errors. They can do this cleanup manually or by using algorithms such as coerce-to-grid,[2] which puts a uniform grid with optimized spacing over the floor plan and snaps vertices to their nearest grid points (see Figure 6).

Extrusion

A complete 3D building model has three major assemblies: walls, architectural components, and floors and ceilings. Extrusion should handle each assembly differently according to its unique characteristics and the specific application needs.

Walls form the building's structural framework. Generating a section of 3D wall from its 2D projection is fairly easy. However, figuring out the normals isn't trivial. There are several ways to solve this problem. One way is contour searching, which is a guided traversal of wall vertices with sealed portals. This process is essentially the same as determining facets in a 2D mesh represented by a half-edge data structure. Contour searching can not only help identify facet normals and the building outline (that is, the mesh boundary) but also provide an object-orientated building representation in terms of rooms and open spaces. Such knowledge can greatly accelerate propagation simulation and potentially visible sets. Figure 7 compares a contour search operation with a simpler solution that decomposes walls into segments and extrudes them as separated wall blocks.

Designers generally view architectural components—such as doors, windows, and staircases—as model accessories. The most intuitive way to deal with them is to define a standard template for each entity class and provide parameters to specify and customize instances. These include shape parameters, such as height and width, and a transformation matrix that transforms the object from the object coordinate system to the world coordinate system. If the application focuses solely on the building's structure and space arrangement, it might ignore the architectural components.

Ceilings and floors are important model parts that link different levels together. The first step in dealing with them is to find each level's exterior outline. Typically, this outline consists of walls. However, objects such as balconies can create ambiguities.

Also, many buildings have concave polygon outlines. The modeler's job is to deliver a tessellated model comprising a set of convex shapes. Depending on the application's requirements, designers can use a sophisticated algorithm, such as constrained Delaunay triangulation, or a naive greedy approach. However, for quality purposes, long, thin triangles should always be avoided.

After tessellation, the floor and ceiling from neighboring levels should fit each other. The process gets complicated when they differ in scale or orientation. In some cases, neighboring levels' exterior outlines don't have the same shape, which makes model assembly more difficult. In such cases, users should be able to select several pivots to perform registration. This will let them coherently line up different levels into a whole model.

In the final step, users assign materials and at-

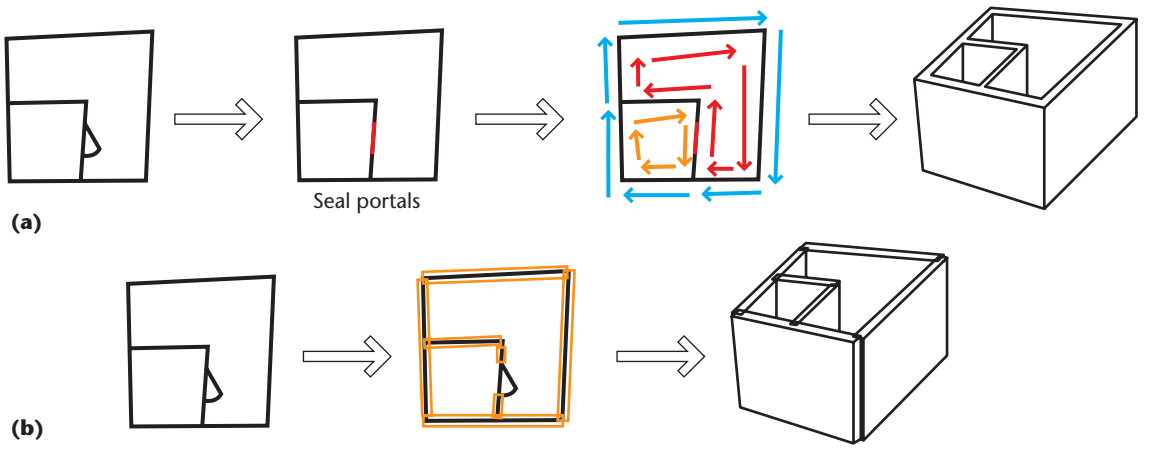

Figure 7. Two wall extrusion algorithms: (a) outline and contour search and (b) block modeling. Two algorithms produce output models of different quality, and users might prefer one style over the other. Contour search is more topologically sound; block modeling is straightforward and runs quickly.

tach textures to make the interior and the facade more persuasive and aesthetically appealing. The system can generate materials and textures procedurally or extract them from image sources.

Commercial Software

Besides the research prototypes, there are many commercial software packages for generating 3D building models. In the AEC industry, software packages fall into three families: full-fledged architectural design packages, general-purpose CAD tools, and plug-ins. None of them combine great efficiency with high automation, so finding a complete problem solution is an ongoing quest.

Product Overview

Instead of using geometric primitives—such as points, vectors, and polygons—as building blocks, modern AEC software uses the building-information-modeling (BIM) paradigm. BIM is a 3D, object-oriented, AEC-specific CAD technique. It covers geometry, spatial relationships, geographic information, and building component quantities and properties. BIM represents a building project as a combination of its parts. To assemble a design, BIM software users select a predefined component template and place it in the drafting window.

BIM software systems include Autodesk's Revit Architectural, ArchiCAD, and Architectural Desktop (ADT; formerly Autodesk Architectural). PlanTracer, an architectural desktop plug-in, claims to be the first product to deal with raster image floor plans. With user assistance, PlanTracer converts 2D floor plans into intelligent objects, such as rooms, walls, and windows.

Almost all modern CAD software generates 3D models. Typically, such software stores architectural information—such as walls, windows, doors, and staircases—as customized architectural components. Because different software products define standard templates or paradigms for architectural entities in unique ways, system compatibility is fairly low.

Product Evaluation

We evaluated several commercial software packages for their strengths and weaknesses. We selected products for evaluation on the basis of completeness, usability, and product quality.

The PlanTracer plug-in runs with ADT and automatically converts vector drawings or raster images to ADT projects. It can also work semiautomatically, letting users select a region of interest to guide symbol recognition. PlanTracer carries out the pipeline's first phase. Using PlanTracer's output, ADT employs BIM to extrude a 3D model. Because PlanTracer requires in-depth knowledge of ADT, its learning curve is steep. Also, PlanTracer's geospatial integration is cumbersome.

Google SketchUp is a simple, efficient 3D modeling program with an intuitive, friendly interface for 3D-model design. Users can draw outlines in a 2D sketchpad and then use a push/pull tool to extrude corresponding 3D volumes and geometries. SketchUp can also export Keyhole Markup Language files for Google Earth, and users can place their building models in Google Earth with accurate georeferencing. Although SketchUp can quickly create a building's outside shell, using it to extrude a building model of a detailed interior structure is manually intensive.

Autodesk Revit is a popular architectural-design-and-modeling tool with full BIM support. Revit lets users design projects using drag-and-place tools with parametric components. Users can create their own object templates or use the tool's well-designed architectural component families. Revit is specifically for architectural purposes and covers every aspect of AEC workflow. The tool's bidirectional associativity lets users freely change their designs and then propagate such changes throughout the model. Revit creates a 3D project view and an exportable mesh model. However, Revit can't automatically create a 3D model from a raster image floor plan.

Only a few systems fully address the problem of generating 3D building models from 2D architectural drawings, and even they aren't completely automated. Vectorization and symbol recognition remain the open issues. Both tasks still require significant manual intervention and will continue to do so as long as architectural representations contain ambiguities or inconsistencies.

For buildings with complex shapes, the conversion is also complex; such shapes might include nonplanar and angled walls. Reconstruction therefore requires elaborate help from regular users or expert designers. However, we do foresee vertical solutions developed to address the needs of specific applications, including homeland security, interactive Web, commercial architecture, and real estate.

Acknowledgments

This work is being funded as a collaboration between Arizona State University and Kutta Technologies on Department of Homeland Security grant NBCHC070060.

References

1. T. Lu et al., "A New Recognition Model for Electronic Architectural Drawings," *Computer-Aided Design*, vol. 37, no. 10, 2005, pp. 1053–1069.
2. R. Lewis and C. Séquin, "Generation of 3D Building Models from 2D Architectural Plans," *Computer-Aided Design*, vol. 30, no. 10, 1998, pp. 765–779.
3. C. So, G. Baciu, and H. Sun, "Reconstruction of 3D Virtual Buildings from 2D Architectural Floor Plans," *Proc. ACM Symp. Virtual Reality Software and Technology* (VRST 98), ACM Press, 1998, pp. 17–23.
4. T. Lu et al., "Automatic Analysis and Integration of Architectural Drawings," *Int'l J. Document Analysis and Recognition*, vol. 9, no. 1, 2007, pp. 31–47.
5. P. Dosch et al., "A Complete System for the Analysis of Architectural Drawings," *Int'l J. Document Analysis and Recognition*, vol. 3, no. 2, 2000, pp. 102–116.
6. C. Ah-Soon and K. Tombre, "Architectural Symbol Recognition Using a Network of Constraints," *Pattern Recognition Letters*, vol. 2, no. 2, 2001, pp. 231–248.
7. S.-H. Or et al., "Highly Automatic Approach to Architectural Floor Plan Image Understanding and Model Generation," *Proc. Vision, Modeling, and Visualization,* IOS Press, 2005, pp. 25–32.
8. H. Luo and R. Kasturi, "Improved Directional Morphological Operations for Separation of Characters from Maps/Graphics," *Proc. 2nd Int'l Workshop Graphics Recognition, Algorithms and Systems* (GREC 97), Springer, 1998, pp. 35–47.
9. T. Kaneko, "Line Structure Extraction from Line-Drawing Images," *Pattern Recognition*, vol. 25, no. 9, 1992, pp. 963–973.
10. L.A. Fletcher and R. Kasturi, "A Robust Algorithm for Text String Separation from Mixed Text/Graphics Images," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 10, no. 6, 1988, pp. 910–918.
11. K. Tombre et al., "Text/Graphics Separation Revisited," *Proc. 5th IAPR Int'l Workshop Document Analysis Systems*, Springer, 2002, pp. 200–211.
12. X. Hilaire and K. Tombre, "Robust and Accurate Vectorization of Line Drawings," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 28, no. 6, 2006, pp. 890–904.
13. W. Liu and D. Dori, "A Survey of Non-thinning Based Vectorization Methods," *Proc. Joint IAPR Int'l Workshops Advances in Pattern Recognition*, Springer, 1998, pp. 230–241.
14. L. Lam, S.-W. Lee, and C.Y. Suen, "Thinning Methodologies—a Comprehensive Survey," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 14, no. 9, 1992, pp. 869–885.
15. D. Dori and W. Liu, "Sparse Pixel Vectorization: An Algorithm and Its Performance Evaluation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 21, no. 3, 1999, pp. 202–215.
16. J.B. Roseborough and H. Murase, "Partial Eigenvalue Decomposition for Large Image Sets Using Run-Length Encoding," *Pattern Recognition*, vol. 28, no. 3, 1995, pp. 421–430.
17. J. Lladoós, E. Martí, and J.J. Villanueva, "Symbol Recognition by Error-Tolerant Subgraph Matching between Region Adjacency Graphs," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 23, no. 10, 2001, pp. 1137–1143.
18. L. Yan and L. Wenyin, "Engineering Drawings Recognition Using a Case-Based Approach," *Proc. 7th Int'l Conf. Document Analysis and Recognition* (ICDAR 03), IEE CS Press, 2003, pp. 190–194.
19. E. Valveny and E. Martí, "A Model for Image Generation and Symbol Recognition through the Deformation of Lineal Shapes," *Pattern Recognition Letters*, vol. 24, no. 15, 2003, pp. 2857–2867.
20. J. Schürmann, "Pattern Classification: A Unified View of Statistical and Neural Approaches," John Wiley & Sons, 1996.
21. S. Belongie, J. Malik, and J. Puzicha, "Shape Matching and Object Recognition Using Shape Contexts," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 24, no. 4, 2002, pp. 509–522.
22. S. Yang, "Symbol Recognition via Statistical Integration of Pixel-Level Constraint Histograms: A New Descriptor," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 27, no. 2, 2005, pp. 278–281.
23. E. Valveny and P. Dosch, "Symbol Recognition Contest: A Synthesis," *Graphics Recognition: Recent Advances and Perspectives*, Springer, 2004, pp. 368–385.

Xuetao Yin *is a PhD candidate in computer science at Arizona State University. His research interests include computer graphics and geometry processing. Yin received his BE in computer science from the University of Science and Technology of China. Contact him at xuetao.yin@asu.edu.*

Peter Wonka *is an assistant professor at Arizona State University. His research interests include computer graphics, visualization, and image processing. Wonka received his doctorate in computer science from the Technical University of Vienna. Contact him at peter.wonka@asu.edu.*

Anshuman Razdan *is an associate professor in the Division of Computing Studies and the Director of Advanced Technology Innovation Center and the I3DEA Lab (i3dea.asu.edu) at Arizona State University at Polytechnic. His research interests include geometric design, document exploitation, and geospatial visualization and analysis. Razdan received his PhD in computer science from Arizona State University. Contact him at razdan@asu.edu.*