EUROGRAPHICS Symposium on Sketch-Based Interfaces and Modeling (2010)
M. Alexa and E. Do (Editors)

Modeling-in-Context: User Design of Complementary Objects with a Single Photo

Manfred Lau[1], Greg Saul[1], Jun Mitani[1,2], and Takeo Igarashi[1,3]

[1]JST ERATO Igarashi Design Interface Project, Tokyo Japan
[2]University of Tsukuba [3]The University of Tokyo

Abstract

The products that we use everyday are typically designed and produced for mass consumption. However, it is difficult for such products to satisfy the needs of individual users. We present a framework that allows the end-user to participate in the entire process of designing their own objects, from the initial concept stage to the production of a new real-world object that fits well with the existing complementary objects. We advocate using a single photo as a rough guide for the user to sketch a new customized object that does not exist in the photo. Our system provides a 2D interface for sketching the outline of the new object and annotating certain geometric properties of it directly on the photo. We introduce a Modified Lipson optimization method for generating the 3D shape. We design a variety of real-world everyday objects that are complementary to the existing objects and environment in the photo. We show that novice users can learn and create new objects with our system within minutes.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Modeling Packages; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques

1. Introduction

The products that we use everyday are typically designed and produced for mass consumption. While this approach works well from an economical point of view, it is difficult to customize such products to fit each user's preference [Gro07, Lan09]. For example, a user may want to create a new and personalized lid to fit an existing cup. In this paper, we present a framework that allows the end-user to participate in the entire process of designing their own customized objects: from the initial concept stage of thinking about the new object and taking a photo of any existing complementary objects, to sketching the new object, to adjusting and viewing it virtually before production, and finally to production of the real-world object that fits well with the existing ones.

The PhotoModeler software [Eos07] is closely related to our work. Their method is *vision-based* and requires an initial camera calibration step that takes user-marked lines or points from a realistic photo as input. In particular, their method cannot take any user 2D drawing as input. Furthermore, for the case of a single photo, many photos cannot be used with their software [Eos07]. In general, PhotoModeler and other vision-based methods ask the user to *trace over existing objects* in photos to recover their 3D shapes. On the other hand, our system is *user-based* and we advocate the use of a single photo for providing some "context" for sketching a *new customized object that does not exist*. Since the new object does not exist in the photo (or the real-world), we cannot apply vision-based techniques. As an example, a possible photo may include a desk with some books and stationery on it to provide the context for drawing a new and personalized desktop organizer. The new object is complementary to the other existing objects and should fit well

Figure 1: *(a) Photo and 2D user sketch of a new and personalized desktop organizer. (b) 3D virtual model visualized and adjusted before production of actual object. (c) New real-world object fits well with other existing objects.*

with them. The photo acts as a reference for the user and is not used in our algorithm, again in contrast to vision-based methods. The advantages of having the photo are that it provide hints about the perspective for drawing the 2D outlines, and the relative dimensions of the new object can be easily assessed by the user.

Given the photo, the user sketches the 2D outline of the new object and certain annotations about the geometric properties of its 3D structure (Figure 1a). Since the 2D sketch is user-drawn and may be inaccurate, the annotations are important and necessary for producing the 3D shape. The idea of using annotations is inspired by Gingold et al. [GIZ09]. To make it easy for novices, our input interface is entirely in 2D. This is in contrast to common CAD modeling programs and SketchUp [Ske09] which require manipulations in 3D that are known to be difficult for beginning users [SSB08]. In addition, although Lipson's method [LS96] tries to automatically recognize certain annotations from the user input, it is not necessarily robust as it depends on the camera perspective and complexity of the shape. We thereby leverage human ability to annotate the 2D sketch.

We take the user input, extract a 2D graph along with the corresponding annotations, and "expand" this 2D input into a 3D structure. We found that a direct application of Lipson's approach [LS96] for performing this expansion does not robustly produce precise 3D shapes and does not provide interactive results. Hence we introduce a Modified Lipson method that improves upon Lipson's algorithm [LS96] in two main ways. First, we use a two-step optimization process, in contrast to Lipson's method which takes a complete 2D sketch as input and performs one optimization step. Our initial optimization finds a depth value for each vertex while trying to satisfy the annotated geometric constraints. It provides an estimate of the solution. The full optimization then finds a 3D position for each vertex, while trying to satisfy the geometric and projection constraints (Figure 1b). Second, we optimize in a reduced dimensional space, as many CAD models have a regular structure and we need not optimize with three dimensions for every vertex. Our algorithm does not perform camera calibration with the information in the photo, but the projection constraint also finds a set of camera parameters such that the projection of the 3D vertices onto the 2D screen matches with the original 2D sketch as much as possible. Our system provides support for virtually displaying and adjusting the dimensions and thickness of the 3D shape before production of the actual object. We convert the 3D shape to either a polygonal mesh for 3D printing or a set of coplanar profiles for laser cutting (Figure 1c).

We show a variety of new real-world objects made with our system. We can sketch each of these objects in between two and ten minutes. Our results demonstrate the concept of modeling-in-context with new objects that fit well with existing ones. Our system supports the creation of user-customizable objects, as shown by the different types of desktop organizers that we made.

Our contributions are: (1) We introduce a Modified Lipson method to create precise 3D shapes interactively; (2) We advocate the use of a single photo as a reference for the user (instead of the algorithm) to sketch a new customized object that does not exist. The photo provides references about the perspective and relative dimensions of the new object; and (3) The end-user participates in the entire process from the initial concept stage to the final production of the new complementary real-world object.

2. Related Work

Modeling-in-context. The majority of commercial modeling software provides an empty space for the user to begin from, with the exception of a 2D grid that can be used as a reference in the 3D space. It is typical for modelers and artists to take photos of the objects to be modeled and use them as a reference. For example, Tsang et al. [TBSR04] placed images of objects in 3D space to guide the modeling process. Thormahlen and Seidel [TS08] demonstrated the use of a set of ortho-images for 3D modeling. These previous methods ask the user to trace over existing objects on the photo, while the user of our system sketches a new object that does not exist. Furthermore, previous methods require that the images be in the canonical (front/side/top) views, while our method accepts one photo in an oblique view.

Sketch-based modeling. Sketch-based methods [ZHH96, IMT99, NISA07] and 3D sketching systems [DXS*07, BBS08, SKSK09] provide simple interfaces for the user to create 3D models or curves. These methods require frequent camera rotation, which makes them difficult to use for novices. The frequent rotation also makes it hard to apply them to work with a single photo. The Smartpaper system [SC04] extends Lipson's method [LS96] to create 3D shapes from 2D sketches interactively. However, they produce rough 3D shapes. These sketch-based systems are for coarse or approximate shapes, and they do not satisfy the constraints necessary for building precise real-world objects.

Image-based modeling. Image-based methods use multiple photos as input for modeling architecture [DTM96, SSS*08], plants [QTZ*06], trees [TZW*07], and facades [XFT*08]. Their goal is to reconstruct the 3D shape of objects already in the photos, while our method displays one photo to the user as a reference. Many single image-based methods ask the user to carefully trace over existing objects, while other methods semi-automatically reconstruct the existing 3D scene in a photo [HAA97, OCDD01]. In contrast, our system allows the user to draw inaccurate sketches of new objects that do not exist in the photo. Recent methods [TFX*08, JTC09] successfully build 3D models of trees and architectures by allowing the user to sketch on one photo. These methods make assumptions about the properties of trees and symmetric architectures, and exploit these

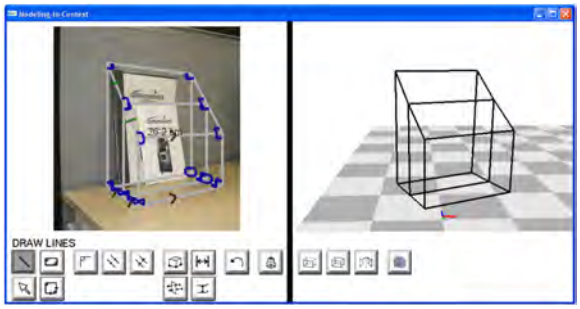

Figure 2: *A screenshot of our system.*

assumptions to reconstruct the 3D shape. Our system is user-oriented, and it is the user-specified information that we exploit to create the 3D shape.

Some single image-based methods [ZDPSS01, WTBS07] ask the user to annotate constraints such as normals to recover depth information. Gingold et al. [GIZ09] ask the user to annotate a single image to build rough 3D character models. Our system is different in that our target is to build precise CAD-like models. Other image-based methods use templates of specific object types [YSvdP05], a shape grammar of architectural primitives [MZWVG07], or templates of basic 3D shapes [Eos07] to help in creating the 3D model. While our system can additionally have such templates to make it easier to sketch specific shapes, we have decided to keep our interface more general as our *Lines* primitive together with the appropriate annotations can create many basic shapes.

3. User Interface

Our system (Figure 2) includes a left window where the user can directly sketch in 2D on top of a photo, and a right window where the generated 3D shape is displayed. The user starts by loading a photo consisting of objects that are complementary to the new object to be created. For example, if the user wants to create a new personalized lid for a teapot, the photo can show the teapot without any lid. If the user wants to create a desktop organizer, the photo can show a desk with the books, stationery, and other objects to be placed in the organizer. The objects in the photo provide a sense of what the size of the new object should be and the perspective that it can be drawn in. They are intended for providing clues to novice users. The user sketches the 2D outline (Section 3.1) of the new object, and annotations (Section 3.2) specifying some geometric properties of the 3D shape to be generated.

3.1. Primitives

Lines. This tool is for drawing 2D straight line segments of various lengths. The user can draw the lines and edit them afterwards by moving their endpoints. We find that users are capable of drawing both visible and hidden lines, and we do not distinguish between them. If needed, the user can annotate symmetric planes (Section 3.2) so that the system can automatically generate some of the hidden vertices and edges from the visible ones.

Rectangles and Squares. The user draws a rectangular shape by clicking and dragging on the 2D screen to specify (but without drawing) one of the diagonals of the rectangle. The *Rectangles* tool automatically creates four connected straight line segments, and annotates the four angles to be right angles. The *Squares* tool additionally annotates the four sides to be of the same length. The user can then adjust the positions of any of the four vertices by dragging.

Circles. When 3D circles are projected onto the 2D screen, they become 2D ellipses in general. Hence we ask the user to draw ellipses in 2D. The user sketches a rectangular shape with one click-and-drag stroke and the corresponding ellipse that is bounded by the rectangle is drawn. After the initial sketch, the user can edit an ellipse by adjusting any of the four points on its major and minor axes. We assume that the 3D circles are horizontal since this is common to most cases. Our examples show the use of this tool as part of the *Surface of Revolution* primitive (Figure 3).

Figure 3: *We draw three circles and a set of lines for the "side" with the Surface of Revolution tool.*

Surface of Revolution. We use this primitive to create shapes consisting of a symmetric rotational axis with concentric circles along the axis. Figure 3 shows an example of drawing a teapot lid. The user draws three 2D ellipses (or circles in 3D): one to identify the ground or table in this case, one to represent the bottom of the lid, and one to represent the top of the lid. As the third ellipse is drawn, the display shows a line to represent the rotational axis. The user then draws the "side" of the object by sketching a series of connected line segments. These segments join the top and bottom circles. Each vertex of this set of lines represents a concentric 3D circle in the new object. The sketching of the line segments uses the same implementation as the *Lines* tool.

3.2. Annotations

There can be many 3D shapes that can match the 2D outline. In addition, we cannot expect users to draw perfectly. The user-drawn 2D outline may not be accurate compared to the intended 3D shape's projection onto the screen. The annotations are important for resolving these inconsistencies.

Right Angles. We find that this geometric relationship is

most useful for recovering the 3D shape. We provide two ways to specify right angles. First, the user can simply click an angle (Figure 4 left), and our system will show that angle as being annotated with the usual right angle symbol (Figure 4 right). Second, the user can specify all the angles of a quadrilateral with one mouse click (Figure 4 middle).

Figure 4: *Two different ways for specifying right angles. It takes 15 mouse clicks to annotate 36 angles (last image).*

Same Length Lines, and Parallel Lines. A mouse click near a line or edge allows the user to select it (Figure 5 left). The user can select groups of any number of each type of lines, by clicking a "done" button after selecting each group. Selected edges are displayed with corresponding geometric symbols (Figure 5 middle).

Figure 5: *Left: Selecting edges with the mouse cursor. Middle: Three groups (each with different color) of two annotated lines. Green dash symbols are for same length lines and red arrows are for parallel lines. Right: Green highlighted vertices are ground vertices. Each vertex is labeled with the number of dimensions used in the full optimization. The total number of dimensions is reduced from 36 to 18.*

Ground Vertices. The user can click on and select 2D vertices (Figure 5 right) that lie on the ground (in 3D). We translate and rotate the current 3D shape so that the selected vertices lie on the ground plane. If there are more than three vertices, we project the remaining ones to the ground.

Symmetric Planes. If the occluded or hidden lines are difficult to draw, we can specify symmetric planes and generate them automatically from the visible ones (Figure 6). The user can select symmetric planes with the mouse cursor with just one click. A symmetric plane is represented by three or more midpoints of the user-drawn 2D lines. If the slopes of a set of 2D lines are similar and the mouse cursor is near one of these midpoints, our system highlights all these midpoints to "suggest" the corresponding symmetric plane. If the perspective of the drawn object makes it difficult to automatically suggest these planes, the user can also select any three or more midpoints directly.

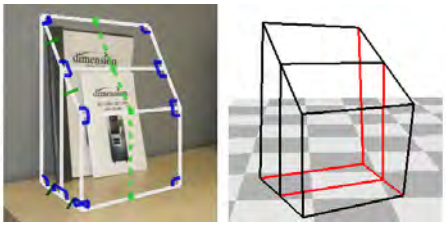

Figure 6: *Left: Symmetric plane (green dotted line) selected with one mouse click. Right: 3D result with visible (black) and occluded (red) edges.*

Dimensions, and Thickness. These tools allow the user to visualize the 3D model before production of the actual object. For annotating dimensions, the user selects an edge and a dialog box is provided for entering its length. The lengths of the other edges are automatically generated, and can be displayed in the right window. The thicknesses of all the polygons have a default value. The user can select a 2D polygon by placing the mouse cursor near the mean position of the polygon, and a dialog box is provided for entering a new thickness. The user can also select polygons that should not be included in the final 3D model (ie. top of a box should be open).

4. Implementation

Given the user-drawn 2D outline and annotations, we parse this input into a 2D graph. We introduce a Modified Lipson method to "inflate" the 2D structure into a 3D shape. An initial optimization that finds a depth value for each vertex generates an approximate shape, which is then used as the starting point for the full optimization to compute 3D positions. We describe two simple methods for finding camera parameters used in the optimization. Finally, we generate the real-world object from the virtual 3D shape with a 3D printer or laser cutter.

4.1. Parse to 2D Graph

As the user sketches the outline of the object and annotates it, we parse the user-drawn lines into a set of 2D vertices and edges. Let $\mathbf{p}$ be a vector containing the positions of the 2D vertices. The set of edges is the set of user-drawn lines, and is the same in 2D and 3D. We also compute and store additional information used during the annotation process, such as explicitly maintaining a list of polygons for efficient detection. For 2D ellipses, we store the two antipodal points (a_1, a_2) on the major axis and the two antipodal points (b_1, b_2) on the minor axis. In our implementation, we only allow these points of the ellipses to connect to the user-drawn lines.

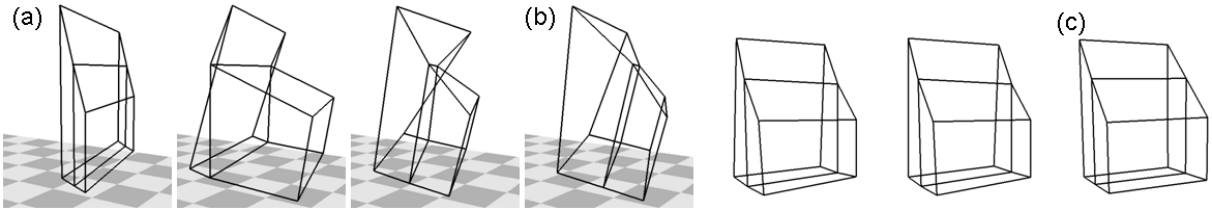

Figure 7: *Our Modified Lipson method "inflates" the original 2D shape in (a) to the desired 3D shape in (c). The geometric constraints here are the annotated right angles shown in Figure 4 (right). (b) is after the initial optimization. (c) is after the full optimization. The camera viewpoint is a result of the optimization.*

4.2. Modified Lipson Optimization

Lipson's original method [LS96] takes a complete sketch as input, and performs one optimization with three dimensions per vertex. We found that a direct application of Lipson's approach generates rough 3D shapes and requires a long runtime. We thereby introduce a Modified Lipson method that performs a two-step optimization, and optimizes in a reduced dimensional space (if possible). For the user-drawn lines, let $\mathbf{v}$ be a vector containing the positions of each 3D vertex. For 3D circles, we assume that they are horizontal and we represent a circle by its center and radius (x,y,z,r).

Initial Optimization. The initial step finds only the depth of each 3D vertex, and there is no dimension reduction in this step. The other two components take the 2D values from the user-drawn sketch and remain constant. We initialize depth values to 0. This first step generates an approximate 3D shape (Figure 7a to b).

The function that we want to minimize uses the annotated geometric relationships, and we call the following energy terms the geometric constraints. The energy term for right angles is $\sum_i (angle_i - 90^o)^2$, where i indexes the angles that are annotated as right angles. The energy term for same length lines is $\sum_s \sum_{i=0}^{n_s-2} (len(vec_i) - len(vec_{i+1}))^2$, where the number of sets of annotated lines is s. For each set, the number of lines is n_s. For each line, vec_i is the vector computed by subtracting one endpoint of the line from the other. $len()$ returns the length of the vector. The energy term for parallel lines is $\sum_s \sum_{i=0}^{n_s-2} \frac{1}{(angle(vec_i,vec_{i+1})-90^o)^2}$, where the notation is the same as for same length lines and $angle()$ returns the angle between two vectors. The overall function to minimize is a weighted sum of the above energy terms. The weights are 0.5, 10, and 10, respectively. The result of the initial step is stored in $\mathbf{v}$. We use the GSL Scientific Library and its multidimensional minimization routines. The optimization method uses the Nelder-Mead Simplex algorithm. The algorithm is not globally optimal, and sometimes may not return the desired shape. Hence we first perform the initial optimization once. Given this solution, we iterate through each vertex and randomly re-initialize its depth, and perform an initial optimization for each case. We choose the best result among all cases.

Full Optimization. For the full optimization, each vertex can have three dimensions in general. However, we can represent each vertex with a smaller number of dimensions if we have additional knowledge about the overall shape. Figure 5(right) shows an example. Each annotated ground vertex has two dimensions. For each quadrilateral annotated as having all right angles, we can compute one of the vertex positions given the other three. This is why some of the vertices have zero dimensions as they are not directly optimized but computed with the other vertices. We store in advance the order of quadrilaterals used in this computation. The vertices with zero dimensions are generated from the other vertices before computing the geometric constraints. We run the full optimization with both geometric and projection constraints, using the solution from the initial optimization as the starting point (Figure 7b to c). For circles, we use the 2D position of the circle center from the user sketch, a depth of 0 and a small default radius as the starting point.

The projection constraint projects the 3D shape to the screen, and compares between the projected 2D shape and the original user-drawn 2D sketch. The motivation is to use the 2D sketch to help in generating a 3D shape that is closer in dimensions and overall shape to what the user intended. Let $\mathbf{v}'$ be the result of the full optimization. We can project each vertex in $\mathbf{v}'$ to the screen to obtain a 2D point. Let $\mathbf{p}'$ be a vector consisting of these 2D points. The energy term for the projection constraint is $\|\mathbf{p} - \mathbf{p}'\|^2$. For each 3D circle, we compute the antipodal points (c_1,c_2,d_1,d_2) on the major and minor axes from (x,y,z,r) during the optimization. The projection constraint energy term for each circle is the least squares difference between (a_1,a_2,b_1,b_2) and (c_1,c_2,d_1,d_2). For a surface of revolution, we assume that the first circle is on the ground plane and the three circles are on top of each other. The projection term solves for (x_g,y_g,r_g), (z_b,r_b) and (z_t,r_t) where the subscripts correspond to the ground, bottom and top circles. The overall function is a weighted sum of the projection term and the geometric terms. We use a weight of 0.1 to 0.8 for the projection constraint. In practice, we found that running the full optimization multiple times with the previous solution as the starting point and gradually lowering the weight on the projection constraint works well. For example, we start

with a weight of 0.4 and lower this by 0.1 each time until we reach 0. This works well because the projection constraint depends on the 2D user sketch of a non-existing object, and is therefore at least somewhat inconsistent with any 3D shape. Hence we put more emphasis on the geometric constraints to produce a precise 3D shape.

4.3. Camera Parameters

Our camera model includes its position, orientation and focal length. We have two methods for computing camera parameters which are needed for the projection constraint. For the first method, we take the photo with a special marker (Figure 11 cup saucer example). The marker's dimensions are known, and we can find the grid's location on the 2D screen. We use this information to compute the camera parameters [FP02]. The second method is more general and we do not use a marker. Instead, the full optimization optimizes for the camera parameters in addition to the 3D shape. The camera is thus "calibrated" with the 2D user sketch and not the photo. Since the user sketch can be inconsistent with any 3D shape, there is no "correct" camera, but the optimization finds a reasonable solution.

4.4. 3D Printer and Laser Cutter

We produce the real-world objects with a 3D printer or laser cutter. We expect the prices of these technologies to reduce in the future, allowing the end-user to create customized objects in their own home. We use the 3D printer for objects that are composed of non-planar shapes (Figure 11 teapot lid). We triangulate the 3D shape to form a surface mesh. To create a polygonal mesh, we add a thickness to the surface by automatically generating a slightly smaller (or larger) copy inside (or outside) the original shape. For the teapot lid, the user naturally draws the outside part and our algorithm generates a smaller inside copy. However, for the scissors cover, the user naturally draws the inside part. The user has the option to specify the thickness and to select which copy to generate. For a surface of revolution, the user-drawn side is revolved around the rotational axis to form other circles. We connect and triangulate the circles to form a mesh. We use the laser cutter for objects that are composed of planar shapes (Figure 11 bookshelf). Our system automatically converts the 3D shape into 2D coplanar profiles, and adds the appropriate finger joints based on the profiles' thicknesses. We generate the real-world pieces with the laser cutter, and assemble the pieces to form the real-world object.

5. Results

We demonstrate our approach with a number of real-world objects (Figures 1 and 11) made with our system. All new objects fit well or exactly with the existing objects. In particular, everyone who saw the real teapot lid was satisfied with the way it fits perfectly with the teapot. For all examples,

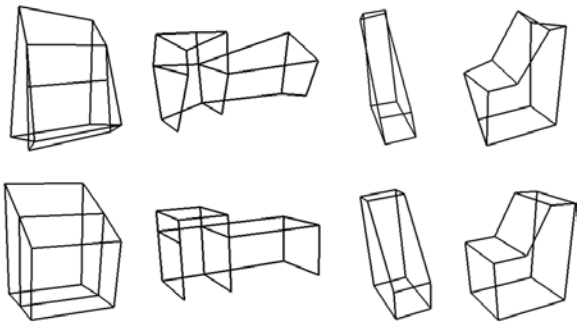

Figure 8: *Top row shows results from Lipson's method. Bottom row shows results from our Modified Lipson method. Each column shows a different example. In each column, the same inputs were used to generate the two results.*

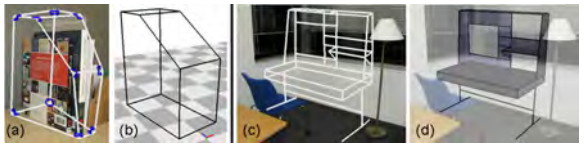

Figure 9: *(a) Inaccurate 2D sketch. (b) Precise 3D shape. (c) 2D sketch of table (annotations not shown). (d) 3D result.*

the runtimes are interactive (less than one second). Figure 8 demonstrates that our Modified Lipson method is more robust than Lipson's method. For each example in the top row, we ran the optimization for more than ten seconds, and the result reached a local minimum. Figure 9(a,b) shows that our algorithm can handle cases where the 2D user sketch is inconsistent with any 3D shape. We created an example (Figure 9(c,d)) with over fifty vertices, and the optimization steps took less than one second.

We performed a user study with four novice users to demonstrate the capabilities of our system. We gave each user a 5-10 minute tutorial of how to draw a cube. Each user first sketches without any photo to create a bookshelf, and then sketches with the photo to create another bookshelf. Figure 10(a,b) shows one user's result. Without the photo, the user intuitively draws the bookshelf's shape in 2D and then "adds a depth" to it. Having the photo provides a rough guide of the *perspective* for the *user* to draw the new object in. The average time taken to create a bookshelf with the photo was 4 minutes 43 seconds. We next gave each user a 5 minute tutorial of sketching a simple teapot lid. Each user then created their own lid. The average time taken was 3 minutes 15 seconds. We show an interesting result (Figure 10c,d) of a customized lid created by one user.

We next describe the relative dimensions of the user-created bookshelves. As a basis for comparison, we measured the dimensions of the books in the photo, and the height to length ratio is 1.35 while the height to width ratio

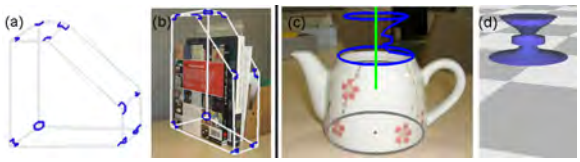

Figure 10: *(a) 2D user sketch without photo. (b) 2D user sketch with photo. (c) 2D user sketch of teapot lid. (d) 3D result.*

is 3.6. Without the photo, the range of the height/length ratio of the user-created bookshelves is 0.93 to 2.27, and the range of the height/width ratio is 1.77 to 2.91. With the photo, the range of the height/length is 1.24 to 1.40, and the range of the height/width ratio is 2.89 to 3.98. Therefore, having just one photo allows the user to more accurately draw the object's *relative dimensions*.

6. Discussion, Limitations, and Future Work

We have presented a system in which a single photo guides the user in sketching a new object, in contrast to vision-based methods that use photos to recover the shape of an existing object. Our algorithm does not explicitly use the information in the photo. For future work, it is possible to use computer vision methods to pre-process the information in the photo. We can then use the information to assist the users or make suggestions to them while they sketch on the photo, as in [TBSR04].

We do not compare our system with SketchUp [Ske09] as it requires manipulations in 3D, while our interface is in 2D. In addition, our system is designed for the user sketch of a *new* object with a photo as a guide, while existing methods such as SketchUp ask the user to trace over existing objects.

One limitation of our work is that the current system does not handle some types of objects that users might want to design. For example, non-rigid objects such as tents and clothing cannot be built with our system. Handling them may require a model for non-rigid objects, and different user interfaces and algorithms for manipulating them. Another example is objects with moving parts. We currently build only static objects, but there are objects with dynamic functional parts such as a closet door or a foldable chair. For future work, there are many issues that can be explored in terms of the interfaces and methods needed for handling these objects.

Another limitation of our method is that we do not consider the structural or aesthetic design of the actual objects. Even though a 3D model can be physically built, it does not mean that the object will be structurally stable. Future work includes building systems for visualizing and evaluating various designs virtually before production of the actual object. The asethetic design of products is also important. A future system that considers the user's preferences and perceptions of aesthetics would be useful. There is still much work to be done towards achieving the goal of enabling the average person design his or her own products, and we hope that our system will inspire future work towards this goal.

References

[BBS08] BAE S.-H., BALAKRISHNAN R., SINGH K.: Ilovesketch: As-natural-as-possible sketching system for creating 3d curve models. *ACM Symposium on User Interface Software and Technology (UIST)* (2008), 151–160. 2

[DTM96] DEBEVEC P., TAYLOR C., MALIK J.: Modeling and rendering architecture from photographs: A hybrid geometry and image-based approach. *ACM SIGGRAPH* (1996), 11–20. 2

[DXS*07] DORSEY J., XU S., SMEDRESMAN G., RUSHMEIER H., MCMILLAN L.: The mental canvas: A tool for conceptual architectural design and analysis. In *Proceedings of Pacific Graphics* (2007), pp. 201–210. 2

[Eos07] EOSSYSTEMS: Photomodeler, 2007. 1, 3

[FP02] FORSYTH D. A., PONCE J.: *Computer Vision: A Modern Approach*. Prentice Hall, 2002. 6

[GIZ09] GINGOLD Y., IGARASHI T., ZORIN D.: Structured annotations for 2D-to-3D modeling. *ACM Transactions on Graphics 28*, 5 (2009), 148. 2, 3

[Gro07] GROSS M.: Now more than ever: computational thinking and a science of design. *Japan Society for the Science of Design 16*, 2 (2007), 50–54. 1

[HAA97] HORRY Y., ANJYO K.-I., ARAI K.: Tour into the picture: using a spidery mesh interface to make animation from a single image. In *ACM SIGGRAPH* (1997), pp. 225–232. 2

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: a sketching interface for 3d freeform design. In *ACM SIGGRAPH* (1999), pp. 409–416. 2

[JTC09] JIANG N., TAN P., CHEONG L. F.: Symmetric architecture modeling with a single image. *ACM Transactions on Graphics 28*, 5 (2009), 113. 2

[Lan09] LANDAY J.: Design tools for the rest of us. *Communications of the ACM 52*, 12 (2009), 80. 1

[LS96] LIPSON H., SHPITALNI M.: Optimization-based reconstruction of a 3d object from a single freehand line drawing. *Computer-Aided Design 28*, 8 (1996), 651–663. 2, 5

[MZWVG07] MÜLLER P., ZENG G., WONKA P., VAN GOOL L.: Image-based procedural modeling of facades. *ACM Transactions on Graphics 26*, 3 (2007), 85. 3

[NISA07] NEALEN A., IGARASHI T., SORKINE O., ALEXA M.: Fibermesh: designing freeform surfaces with 3d curves. *ACM Transactions on Graphics 26*, 3 (2007), 41. 2

[OCDD01] OH B. M., CHEN M., DORSEY J., DURAND F.: Image-based modeling and photo editing. *ACM Transactions on Graphics* (2001), 433–442. 2

[QTZ*06] QUAN L., TAN P., ZENG G., YUAN L., WANG J., KANG S. B.: Image-based plant modeling. *ACM Transactions on Graphics 25*, 3 (2006), 599–604. 2

[SC04] SHESH A., CHEN B.: Smartpaper: An interactive and user friendly sketching system. *Computer Graphics Forum (Eurographics) 23*, 3 (2004), 301–310. 2

[Ske09] SKETCHUP: Google, 2009. 2, 7

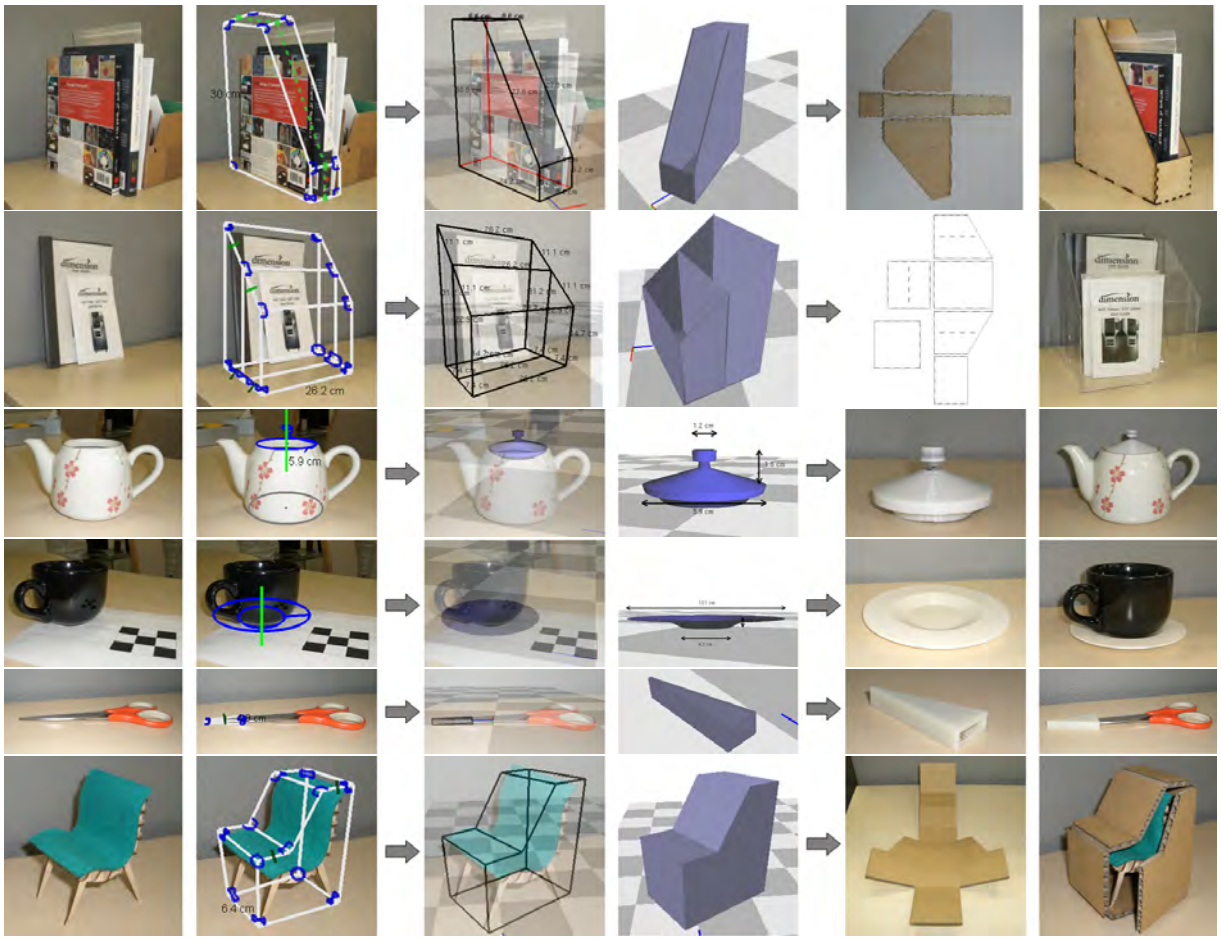

Figure 11: *Each row shows the original photo, the photo with 2D sketch, the resulting 3D virtual model (middle two), and the new real-world object (right two) fitting well or exactly with the existing objects. The objects are (from top to bottom): bookshelf (wood), desktop organizer (plastic), teapot lid, cup saucer, cover for scissors, and storage box (cardboard) for chair.*

[SKSK09] SCHMIDT R., KHAN A., SINGH K., KURTENBACH G.: Analytic drawing of 3d scaffolds. *ACM Transactions on Graphics 28*, 5 (2009), 149. 2

[SSB08] SCHMIDT R., SINGH K., BALAKRISHNAN R.: Sketching and composing widgets for 3d manipulation. *Computer Graphics Forum 27*, 2 (2008), 301–310. 2

[SSS*08] SINHA S., STEEDLY D., SZELISKI R., AGRAWALA M., POLLEFEYS M.: Interactive 3d architectural modeling from unordered photo collections. *ACM Transactions on Graphics 27*, 5 (2008), 159. 2

[TBSR04] TSANG S., BALAKRISHNAN R., SINGH K., RANJAN A.: A suggestive interface for image guided 3d sketching. In *Proceedings of ACM SIGCHI* (2004), pp. 591–598. 2, 7

[TFX*08] TAN P., FANG T., XIAO J., ZHAO P., QUAN L.: Single image tree modeling. *ACM Transactions on Graphics 27*, 5 (2008), 108. 2

[TS08] THORMAHLEN T., SEIDEL H.-P.: 3d-modeling by ortho-image generation from image sequences. *ACM Transactions on Graphics 27*, 3 (2008), 86. 2

[TZW*07] TAN P., ZENG G., WANG J., KANG S. B., QUAN L.: Image-based tree modeling. *ACM Transactions on Graphics 26*, 3 (2007), 87. 2

[WTBS07] WU T.-P., TANG C.-K., BROWN M., SHUM H.-Y.: Shapepalettes: Interactive normal transfer via sketching. *ACM Transactions on Graphics 26*, 3 (2007), 44. 3

[XFT*08] XIAO J., FANG T., TAN P., ZHAO P., OFEK E., QUAN L.: Image-based façade modeling. *ACM Transactions on Graphics 27*, 5 (2008), 161. 2

[YSvdP05] YANG C., SHARON D., VAN DE PANNE M.: Sketch-based modeling of parameterized objects. *Eurographics Workshop on Sketch-Based Interfaces and Modeling* (2005), 1–10. 3

[ZDPSS01] ZHANG L., DUGAS-PHOCION G., SAMSON J.-S., SEITZ S.: Single view modeling of free-form scenes. *IEEE CVPR* (2001), 990–997. 3

[ZHH96] ZELEZNIK R., HERNDON K., HUGHES J.: Sketch: An interface for sketching 3d scenes. *ACM SIGGRAPH* (1996), 163–170. 2