EUROGRAPHICS Symposium on Sketch-Based Interfaces and Modeling (2009)
C. Grimm and J. J. LaViola Jr. (Editors)

A Sketch-based Interface for Photo Pop-up

J. Ventura,[1] S. DiVerdi[2] and T. Höllerer[1]

[1]Department of Computer Science, University of California, Santa Barbara
[2]Adobe Systems, Inc.

Abstract

We present sketch-based tools for single-view modeling which allow for quick 3D mark-up of a photograph. With our interface, detailed 3D models can be produced quickly and easily. After establishing the background geometry, foreground objects can be cut out using our novel sketch-based segmentation tools. These tools make use of the stroke speed and length to help determine the user's intentions. Depth detail is added to the scene by drawing occlusion edges. Such edges play an important part in human scene understanding, and thus provide an intuitive form of input to the modeling system. Initial results and evaluation show that our methods produce good 3D results in a short amount of time and with little user effort, demonstrating the usefulness of an intelligent sketching interface for this application domain.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Object Modeling—Implicit surfaces I.3.6 [Computer Graphics]: Techniques—Sketch-based interfaces I.4.6 [Image Processing]: Segmentation—Hierarchical segmentation

1. Introduction

The advent of digital photography has led to an explosion of personal photograph collections. Websites such as Flickr and Facebook allow for easy sharing of photographs with family and friends. They also provide tools to annotate photographs with extra information such as a caption, keywords, names of people and their place in the photograph, and tagged objects of interest.

Although much progress has been made in adding semantic information to digital photographs, they remain two-dimensional images which are generally viewed in the same way as the always have been. If we can recover the depths in an image, we can treat the photograph as a 3D scene. This effect has been described as "photo pop-up" because it is similar to creating a pop-up book out of a picture [HEH05]. This makes for a new kind of viewing experience where the depth in an image can be perceived, for example using stereoscopic or head-tracked rendering. Depth information is also useful for image-editing operations, as has been shown before [OCDD01]. 3D movies are also starting to make a comeback, and many movie theaters are adding stereo projection technologies. The depth-enhanced image may become a more commonplace form of media in the near future.

However, unless specialized equipment such as a stereo camera or a laser scanner has been used to record depth information at the time of image capture, existing images have no depth information. We need interaction techniques for adding this depth information easily and quickly, given the great abundance of imagery in existence today.

Existing 3D modeling solutions require too much user effort to produce results, and have a steep learning curve. We envision an interface which is simple enough to be embedded into photo sharing websites such as those mentioned above, for quick 3D markup of photographs by novice users. Although automatic methods are being developed to turn 2D images into 3D models, they cannot handle a wide range of images, including many of the kinds of images commonly found in personal photo collections.

In this paper we explore a sketch-based interface for adding depth information to an image. We feel a sketching interface is appropriate to this task for several reasons. Often we want to segment out organic shapes such as people and trees, which can have a difficult contour to trace manually. As an alternative, we introduce novel segmentation techniques which uses free-form strokes on the interior of foreground objects, taking advantage of the properties of pen-based input to improve the interface. We targeted a system for producing visually compelling 3D photographs rather than exactly accurate range images. Our sketch-based interface for adding depth allows for easy interaction and encourages creativity, without requiring slow or overly precise input.

Specifically, our contributions in this paper include two sketch-based segmentation techniques with incremental feedback. These techniques make use of stroke properties to guide the segmentation, going beyond prior art which uses only marked pixel sets to seed the segmentation. Another contribution is a depth refinement tool which lets users draw depth edges rather than paint the depth at every pixel. In a formative evaluation we observed participants using these tools. Observed ease of use and feedback was positive throughout, and specific suggestions indicate opportunities for future work.

We discuss previous work in single-image modeling in Section 2. The workflow of our interface begins with setting up the background geometry, which is described in Section 3. Then foreground objects are selected for pop-up, using the tools presented in Section 4. Depth detail can then be added using the edge brush, described in Section 5. Finally the full 3D model is generated and rendered, as described in Section 6. We discuss our informal evaluation of our techniques in Section 7, and present conclusions in Section 8.

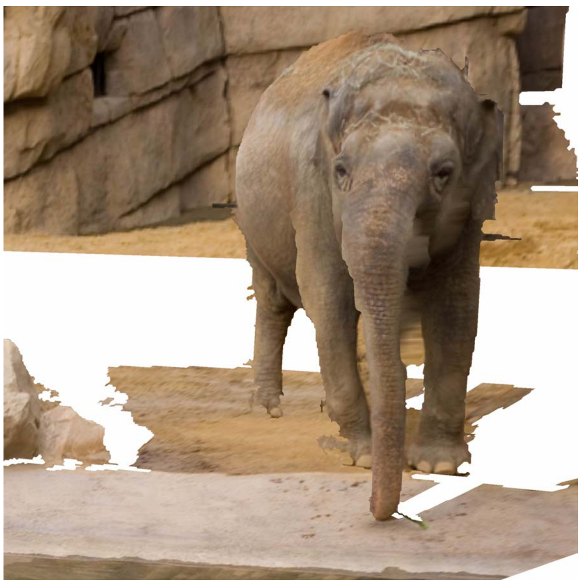

Figure 1: *A synthesized rendering of the photograph in Figure 5 from a novel viewpoint. This model was made in forty-five seconds using our techniques. Afterwards, hole filling was used behind the elephant's trunk [Vis09].*

2. Related Work

Early user interface work for 3D modeling from images used simple tools, but relied on complex external data sources. The *Tour into the Picture* system by Horry, Arai and Anjyo introduced a diorama-like interface for images with single-point perspective and flat foreground objects [HAA97]. Foreground masks for pop-up layers have to be created externally and imported along with the image. Around the same time, Debevec, Taylor and Malik created the *Façade* system for image-based modeling. With this program, users can register familiar geometric primitives to the input images to build up a 3D model. However, the system relies on a large amount of images from varying viewpoints to reconstruct the complete structure.

Later, Oh et al. introduced an interface for single-view modeling which can produce more detailed scenes than *Tour into the Picture* [OCDD01]. This interface is similar to a paint program for depth. They include a color-to-depth tool which is useful when the source of light is close to the camera (the dark-is-deep assumption). They also include a level set method to specify objects that generally bulge in the middle, and a specialized tool for faces. However, their system is not made for quick interaction. They report spending thirteen hours on a single picture, with ten of those hours spent on separating out the layers in the image. Similarly, mesh-based modeling systems such as that of Zhang et al. or Joshi and Carr can be used to create a complex mesh from a photograph [ZDPSS01, JC08]. These interfaces involve setting detailed constraint lines around and inside of objects, and are not exactly suited to the pop-up scenario. Zhang et al. report taking up to 1.5 hours on a single image. In our work we present an interface for single-view modeling which only requires a few minutes of interaction for acceptable results, by making use of both sketch-based input and image features.

There also has been work in automatic systems for estimating the 3D scene in a photograph [HEH05, SSN09]. These systems have a low success rate on images with prominent foreground objects such as people, and also fail on uncommon images. The task of correctly identifying and segmenting foreground objects is still outside the realm of automatic methods. The LabelMe project provides a web interface where users are asked to specify polygons around objects in images and supply tags for each object [RTMF08]. This database is being used to improve automatic object segmentation and recognition systems. The authors of LabelMe have also developed a set of heuristics to infer the 3D scene from these polygons, without explicit 3D input from the user [RT09]. However, the output doesn't have the same depth detail as can be achieved with some level of user interaction.

Many image segmentation techniques require users to draw curves inside and outside of the target object [RKB04, LSTS04]. Then, a global integration technique such as graph cut [BJ01] is employed to integrate the user input with the image information and choose a boundary for the object. Even more automatic techniques, such as GrabCut, require the user to add foreground and background strokes when the initial segmentation is incorrect [RKB04]. The recent work of Olsen Jr. and Harris on edge-aware brushes provides a good comparison of such techniques and an explanation of their weaknesses [OH08]. In this work we explore using sketch-based techniques for image segmentation, and take advantage of the properties of the stroke itself (such as length and speed) to make image segmentation more controllable.

In our work we adapt recent developments in gradient-domain image editing to create a tool for editing depth detail. Thanks to the emergence of fast GPU hardware for parallel processing, an image gradient can be integrated at real-time rates [GWL*03]. Recent work has exploited such a gradient-domain solver to create novel color painting and image editing systems [MP08, OBW*08]. Our contribution is in the adaptation of a gradient-domain solver to sketch-based 3D object modeling.

3. Specifying background

Most pictures can be divided into background and foreground, where the background has a simple geometry consisting of planes. In single-point perspective, dominant lines converge to a vanishing point lying on the horizon (vanishing line). In two-point perspective, two sets of dominant lines converge to separate vanishing points on the horizon. Previous systems have realized the usefulness of these geometric primitives for scene modeling [HAA97, KPAS01]. By either specifying the vanishing point, sets of converging lines, or the vanishing line itself, we can estimate the pitch and roll of the camera with respect to the ground plane. The other piece of information needed is the camera's focal length, which can be provided either by EXIF tags or manual input.

For our interface the height horizon is set by single click on the image. We assume a flat horizon (no camera roll) which is in the field of view of the camera. Other cases are easily handled, but this was sufficient for our experiments. Kang's paper gives details on how to determine the background geometry once the horizon is set [KPAS01]. Once we have the ground plane, the depth of objects which are resting on the ground is easily determined by their point of contact with the ground in the image. We leave out of consideration pictures where this point of contact is not visible; for these, the depth would have to estimated by dead reckoning or other means. We assume an object touches the ground at its lowest contour point; alternately we could have the user draw the line of contact as has been previously implemented [OCDD01].

4. Labeling foreground objects

After the background geometry has been specified, the next step is to label foreground objects which will be given depth values based on their contact with the ground. Directly drawing the contour of an object can be difficult, especially in the case of organic forms. Instead, we created a image-based selection tool for foreground objects. The user selects areas of the image by drawing strokes inside and outside objects, and uses properties of the stroke to control the selection.

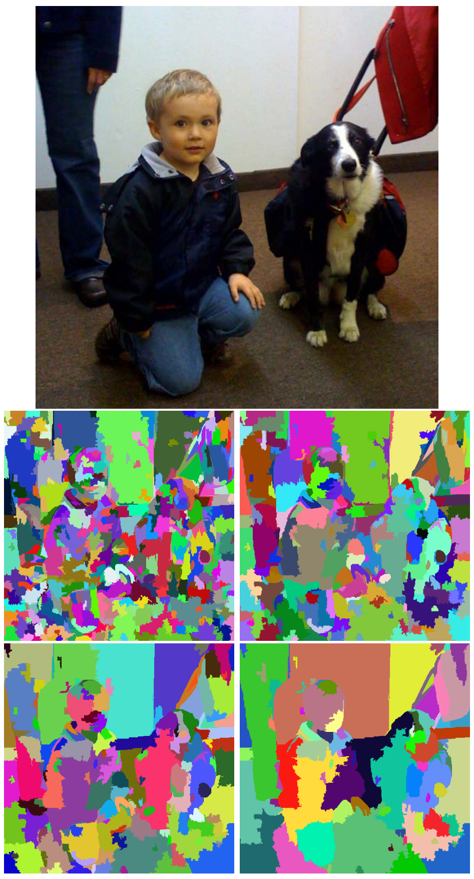
Figure 2: *Image and segmentation pyramid. Small segments in level zero (top left) are progressively joined together in levels one (top right), two (bottom left), and three (bottom right).*

As described in the following sections, this tool uses a pre-segmentation of the image at several scales. Often the scale parameter of a segmentation needs to be tuned for different images, different objects in an image, or even different parts of the same object. The advantage of a multi-scale segmentation is that we can capture all of these different possible segmentations, and then allow the user to guide the algorithm in choosing the correct scale in different regions of the image. Our sketch-based tools allow the user to collect together patches from different scales into a complete object segmentation, in a quick and controllable manner.

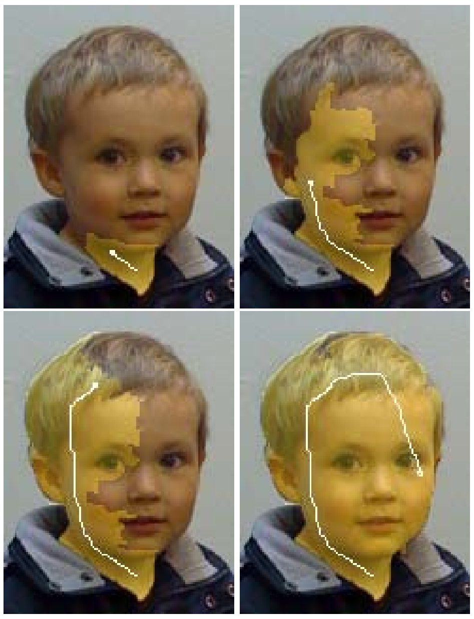

Figure 3: *Selection example. Here we are selecting a person and starting with the face. This example uses the length-based stroke method. As the stroke progresses, patches are chosen from increasingly higher levels of the pyramid.*

4.1. Scale-space segmentation pyramid

We use a graph-based, globally optimal segmentation method [FH04]. The algorithm starts with one segment per pixel, and iteratively joins neighbors with edge weight below a threshold. A scale parameter k determines the strength of boundaries between segments in relation to their similarity. Increasing k allows lower contrast segments to be joined together. We first set a small value of $k_0 = 40$ and join all possible segments. Then we increase k where $k_i = 4 \times k_{i-1}$ and, starting from the previous segmentation, join more segments as possible. This is repeated two more times to produce a four-level segmentation pyramid. Each segment in a lower pyramid level is contained in a larger segment in a higher level.

We generate the segmentation pyramid when the image is loaded. Processing takes less than one second for a 512×512 color image on a 2.16 GHz machine. Figure 2 shows an example segmentation pyramid. The parameters of the segmentation, such as the choice of k values and the number of pyramid levels, could be changed depending on the image. These issues will be discussed later in the evaluation section.

4.2. Two methods for object selection

To label a foreground object, the user needs to collect together appropriate patches from the segmentation pyramid. We could allow for explicit selection of a segmentation scale, but the scale may need to be changed in between every stroke, or even during the stroke itself. Instead, we offer a sketch-based interface for object selection, which uses properties of the stroke to determine an appropriate patch scale.

The basic assumption of our selection tools is that patches under the stroke will be selected. These patches will be added or removed from an object depending on the tool mode. Intuitively, a longer stroke could be used to select larger patches, and a shorter stroke for smaller patches. On the other hand, we could argue that fast movements will be used for larger patches, and slow movements for detailed work. We implemented both ideas to compare them.

For the length-based stroke, a series of thresholds determines the patch scale. These thresholds are given in Table 4. Any patches in the chosen level touched by the stroke are selected. As the user draws the stroke, the selection is recomputed using the update stroke length and shown to the user in a transparent overlay.

In early testing, we noticed that sometimes the user might extend the stroke too far, and select either unwanted patches or too high of a pyramid level. Because of this, we added an "undo" operation to the length-based selection tool. The stroke is shown in white as it is drawn. If the pen stroke goes back on itself, we shorten and redraw the stroke, and recalculate the selection. This helps to make the tool more controllable, without requiring extra strokes to remove unwanted patches.

The speed-based stroke similarly uses thresholds to determine the patch scale, given in Table 4. The speed is calculated by averaging over five consecutive segments of a stroke. Unlike the length-based stroke, we do not choose one pyramid level for the entire stroke. Since speed can vary across the stroke, we instead choose a level for each chain of five stroke segments.

Level	Stroke length (px)	Stroke speed (px/s)
0	0	0
1	50	50
2	100	100
3	150	150

Figure 4: *Thresholds for length-based and speed-based selection stroke. For example, a length-based stroke at least 50 pixels long but shorter than 100 pixels will choose patches from pyramid level one.*

We expected that the length-based tool would lend itself to making several strokes to complete the segmentation, first starting with longer strokes for larger patches, and smaller strokes for cleanup work. As an alternative, the speed-based tool allows for fewer, longer strokes to select an entire object. The stroke speed can be changed as needed to select larger or smaller patches, without lifting the pen. We tested these hypotheses in our informal evaluation, described in

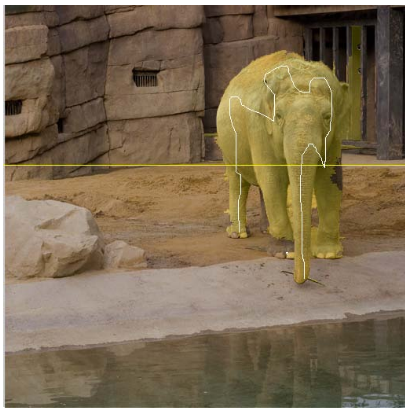

Figure 5: *Selection example. Here the speed-based stroke was used to select the elephant. Afterwards, small strokes can be used to select the missing patch on the leg, and remove the patch of background that was mistakenly selected.*

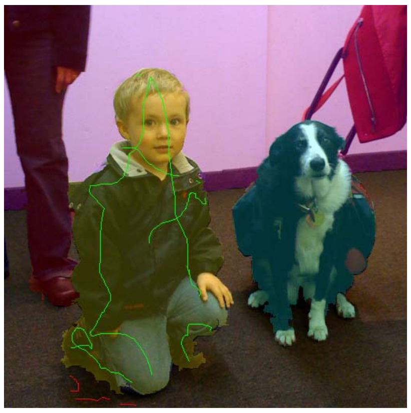

Figure 6: *Completed segmentation example. The strokes used to segment the boy are shown. The green strokes added patches to the selection and the red strokes removed patches. The length-based technique was used here.*

Section 7. Figures 3, 5 and 6 illustrate the length-based and speed-based tools in use.

4.3. Setting initial object depth

Separate objects are selected using differently colored labels from a palette. Figure 6 shows a completed segmentation where all objects have been labeled with different colors. Once all of the objects have been cut out, we use each object's lowest point in the image as the point of contact with the ground. The depth value is read from the OpenGL depth buffer, and applied to the object. At this point, we have a complete scene with ground, background, and pop-up objects. However, the objects are flat, which may not be appropriate for most images. Section 5 describes our drawing tools for adding depth detail to the initially flat layers.

5. Adding depth detail

Prior work in editing depth maps has used tools for directly specifying depth values [OCDD01]. However, humans are better at estimating relative depth than absolute depth [WH05]. Also, occlusion edges (which delineate a overlapping relationship between two surfaces) have been hypothesized to be one of the fundamental components of visual scene understanding [Gib68]. This motivated us to create a tool for marking occlusion edges on the image, where the system solves for the resulting depth map. Unmarked areas are assumed to be smooth. The simplifying assumption of a smoothness prior has the potential to make depth editing faster and easier than painting depth on a surface directly.

In a depth map, an occlusion edge is a line where the depth gradient is non-zero. In our system, the user traces occlusion edges for each foreground object directly on the image. Intermediate values are then computed automatically by the Poisson solver, which is described in the following section. By specifying the depth gradient completely, we can recover the entire depth map (up to an additive constant). Recent gradient-domain painting systems have shown that the gradient can be interactively edited and integrated in real-time [OBW*08, MP08].

5.1. Gradient-domain depth editing

The output of our depth detail tool is a per-pixel offset map which is added to the depth map acquired from the established background and foreground geometry. For the offset map u, we maintain G^x and G^y, the x and y components of the gradient of u. We can solve for u from G^x and G^y by solving Poisson's equation:

$$\nabla^2 u = f \tag{1}$$

where f is the divergence of the gradient, computed by:

$$f_{i,j} = G^x_{i+1,j} - G^y_{i-1,j} + G^y_{i,j+1} - G^y_{i,j-1} \tag{2}$$

We set $f = 0$ at the edges of the image (Neumann boundary condition). Initially, $f = 0$ inside the image as well. Through our sketch interface (described below), the values of G^x and G^y can be set by the user for input to the solver.

We solve the Poisson equation using standard multigrid methods implemented on the GPU. We achieve performance of about 30 frames per second on a GeForce 8600M card, sufficient for real-time interaction with the system. The *Numerical Recipes* book gives a good explanation of the basics of a multigrid solver for Poisson's equation [PTVF07]. Details on the GPU implementation can be found in McCann's paper [MP08].

5.2. The depth refinement tool

We developed an edge "brush" that specifies occlusion edges by setting the gradient to non-zero values. The edge brush is directional, so that by default the depth will decrease from left to right across the stroke. This means that clockwise and counterclockwise strokes have opposite effects. If the depth change is in the wrong direction, a single click will flip the sign of the gradient, so that the stroke will have the correct effect. The magnitude of the depth change across the stroke can be varied to create larger or smaller edges.

Example interactions on a face might be to give a contour to the nose or bring out the chin from the neck. On the front of a house, we may want to mark the windowsills or the doorknob on a door. These edits are local to the object's surface, and represent relatively small changes in depth compared to the background and foreground geometry specified in earlier steps. Thus achieving the correct gradient magnitude along internal edges is usually not as important as it is for the contour between foreground and background.

An interesting feature of the edge brush is it allows for incremental edits to the depth map. Because we are setting relative as opposed to absolute depth values, edges added later in the process will properly affect areas edited earlier. This behavior is illustrated in Figure 7, which shows a spiral being drawn with the edge tool. As the stroke progresses, the added layers of depth edges push the interior region of spiral higher.

The incremental nature of this tool is useful when building up a depth map. For example, in Figure 8 we show the edges drawn on a picture of the buildings on a rocky slope. The edges on the buildings serve to cut them out from the side of the hill. However, the series of strokes on the rocks below ensures that the hill is pushed back from the cliff.

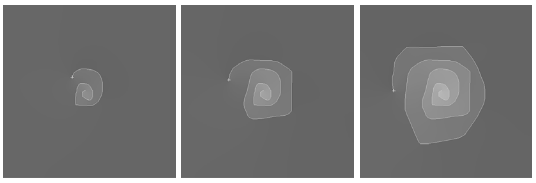

Figure 7: *Edge brush example. Lighter pixels are closer, and the stroke is drawn in white. As the stroke spirals on itself, the depth of the interior regions is progressively decreased.*

Another important quality of the edge brush is that partial occlusions in the interior of objects can be specified. For example, the line between the arm and the torso can be drawn, although at the shoulder the two surfaces are connected. Thus the system will put the arm in front of the torso, but will smoothly connect them at the shoulder. This is a nice property that would be difficult to achieve without a system for automatically interpolating depth values.

We also include an eraser which returns the gradients under the stroke to zero, removing constraints set by the edge brush.

Occlusion edges are only added to the area of the currently selected object, which prevents edges from crossing the object boundary. As occlusion edges are drawn on the image, the display can be switched between the image and the offset map. The offset map is tone mapped by adding a constant so that the average display value is half of the maximum intensity. Edge strokes are overlaid in white.

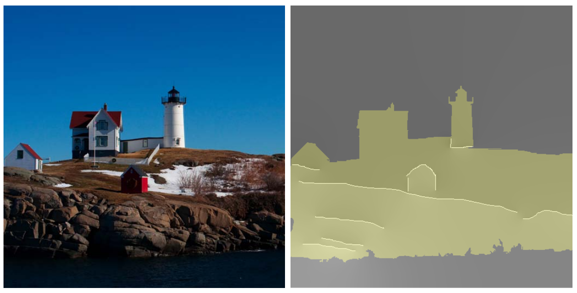

Figure 8: *Edge strokes applied to a picture. The yellow overlay shows the segmentation result. Note the composite effect of the layered horizontal edges.*

6. Meshing and rendering

After the segmentation and occlusion edges have been completely specified, we convert the depth map to a triangle strip mesh. We use the TriMesh2 software library by Szymon Rusinkiewicz, which is intended for meshing laser scanner output [Rus09]. Neighboring pixels separated by a distance greater than a threshold are not joined in the mesh. Each pixel is assigned its corresponding color from the image. The textured mesh model can be rendered from novel viewpoints using any 3D model viewer. Refer to Figure 1 for an example synthesized image.

7. Evaluation

We ran an informal evaluation to test the effectiveness of our sketch interface for photo pop-up. We wanted to explore the different options for the selection tool, and see which design users preferred. We also wanted to test whether users would understand the edge brush and how to use it to add depth detail, and see what accuracy they would achieve.

We asked three participants to test the system with the image shown in Figure 2. We used a small Wacom tablet for input, with the eraser tip used for the erasing technique in both the selection and depth edge mode. One participant was an expert in sketch-based interfaces and had significant experience using a pen and tablet. For the other two participants, this was their first time using a tablet, which they reported made the interface slightly uncomfortable at first.

For each participant, first the general goals of the system was explained and demonstrated. Then, before each brush was used, the mechanics of the brush was explained and demonstrated to the participant.

Participants were shown the length-based selection tool, and asked to try segmenting out the boy in the picture with it. With the length-based stroke, participants often reported that too large of patches were being selected. One participant said that they did not expect such large patches to be selected on this small of an image (512 pixels square). This suggests that it might be useful to allow the parameters of the segmentation pyramid to be adjusted according to the image.

However, large patches were being selected because participants tended to want to draw one long stroke, rather than many strokes of variable length, as was the original intention of the brush. The natural mode of interaction seems to be to not lift the pen for as long as possible, rather than to make a small stroke and see what the effect is. The undo feature is available with the length-based stroke, which allows the stroke length to be reduced after drawing. However, once the stroke path is resumed, large patches will again be selected, which keeps users from making one long stroke to select all the desired patches. This suggests that the length-based stroke may be improved by resetting the segmentation level back to the smallest scale when an undo is invoked, and resetting the length counter to zero. In this the way the user could continue selecting appropriate patches without having to lift the pen.

As one participant noted, another issue with the length-based stroke is that patches at the beginning of the stroke can be affected by movement at the end of the stroke, since as the stroke is lengthened, the entire selection is re-calculated. We also observed that our participants did not make much use of the undo feature. This may have been because the undo could not be invoked once the pen has been lifted. It may require more practice for participants to get used to this tool.

Next, the speed-basd selection tool was shown to participants. The previous segmentation was cleared and users were asked to again try selecting the same foreground object. The concept of the speed-based stroke seemed to make more sense to participants; in fact, one participant had independently suggested the idea while using the length-based tool. With this tool, users were able to hold the pen down for a very long stroke. They tended to move slowly, trying to get each region of the object perfectly as they went. Rather than first selecting everything, and then going back with the eraser, participants would stop and fix problems as they occurred, which may have made the overall segmentation time longer than it had to be, since they would repeatedly switch between selecting and deselecting patches. However, this strategy may change after extended use of the interface.

In our evaluation we found that the length-based stroke made it easier for participants to understand and control which level of the pyramid was being used for patch selection. Generally the length of the stroke is easier to see and control, as opposed to the speed. We tried to choose speed thresholds which were appropriately far enough apart, based on early experiments with the system. However, these could be further tuned, or adjusted explicitly by the user. The expert participant suggested visually displaying the stroke speed, so that it is easier to predict which level will be used.

The visual feedback of the segmentation system also affects performance. In our experiments we did not show the user the segmentation pyramid while they were using the selection tools. We only showed the current object labeling, transparently overlaid on top of the image with each object in a different color, and the current stroke line in white. It may be beneficial for the patch edges to be visible. We noticed that some time was spent by participants trying to find the extent of patches in the segmentation, so that the correct ones could be selected. This becomes an issue in areas with edges that have too low contrast to be detected by the segmentation algorithm, because the desired segmentation cannot be achieved unless the parameters of the segmentation are changed.

We also had participants try the edge brush on the same image. They were told to draw in any important occlusion edges, and to check the depth map to make sure the correct gradients were being added. All of the participants understood the idea of the brush, although the edges that were actually drawn varied among participants. Some tried to mark all of the depth edges to make a very detailed image, and others only put in the essential lines. This result is similar to the segmentation study by Martin et al., where some participants made very detailed segmentations, and others only outlined the high level objects [MFTM01].

At first, participants would switch between the image and the depth map to check if the gradient was in the correct direction, and would click the pen to flip the edge if necessary. However, after a while, all of the participants stopped checking this and just drew on the image itself, which resulted in many wrongly specified edges. This suggests that a less cumbersome interface would be useful for checking the result of added depth edges. Also, participants had a hard time accurately drawing depth edges directly on the contours of the image. The inclusion of an edge snapping technique would probably improve this interface, if it proved sufficiently accurate.

8. Conclusions and future work

In this work we described an interface for creating 3D scenes from photographs which is arguably faster and easier to use than previous systems. We developed two novel sketch-based object selection techniques which make intelligent use of stroke properties, in conjunction with a hierarchical segmentation obtained through iterative graph-based clustering.

We also developed an interface for adding depth detail which allows drawing of depth edges, and solves for the resulting surface.

Using our interface, visually compelling 3D models can be created from a wide range of photographs. Interaction time is usually under ten minutes, while simple scenes can be constructed within a minute. Our interface occupies a comfortable middle ground between complex systems that require substantial user effort [OCDD01, ZDPSS01, JC08], and semi-automated systems which lack the detailed control necessary to arrive at satisfactory results [HEH05, RT09].

Our formative evaluation indicated the usefulness of these techniques, and suggests several avenues of future work. There are many possibilities to augment the visual feedback of the system, for example by explicitly showing the segmentation pyramid and the currently chosen patch scale. The depth edge tool might also benefit from a different visual interface, where the effect of strokes can be seen together with the image itself. We would like to follow up our formative evaluation with specific controlled experiments to gain more insight into these issues.

Acknowledgments

This work was partially supported by NSF CAREER grant IIS-0747520, and NSF IGERT grant DGE-0221713.

References

[BJ01] BOYKOV Y., JOLLY M.-P.: Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on* (2001), vol. 1, pp. 105–112 vol.1.

[FH04] FELZENSZWALB P. F., HUTTENLOCHER D. P.: Efficient graph-based image segmentation. *Int. J. Comput. Vision 59*, 2 (2004), 167–181.

[Gib68] GIBSON J. J.: The perception of surface layout: A classification of types. Unpublished "Purple Perils" essay, November 1968.

[GWL*03] GOODNIGHT N., WOOLLEY C., LEWIN G., LUEBKE D., HUMPHREYS G.: A multigrid solver for boundary value problems using programmable graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 102–111.

[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 *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), ACM Press/Addison-Wesley Publishing Co., pp. 225–232.

[HEH05] HOIEM D., EFROS A. A., HEBERT M.: Automatic photo pop-up. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), ACM, pp. 577–584.

[JC08] JOSHI P., CARR N.: Repoussé: Automatic inflation of 2D artwork. In *EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling, SBIM 2008, June, 2008* (Annecy, France, June 2008), Alvarado C., Cani M.-P., (Eds.).

[KPAS01] KANG H. W., PYO S. H., ANJYO K., SHIN S. Y.: Tour into the picture using a vanishing line and its extension to panoramic images. *Computer Graphics Forum 20*, 3 (2001), 132–141.

[LSTS04] LI Y., SUN J., TANG C.-K., SHUM H.-Y.: Lazy snapping. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), ACM, pp. 303–308.

[MFTM01] MARTIN D., FOWLKES C., TAL D., MALIK J.: A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision* (July 2001), vol. 2, pp. 416–423.

[MP08] MCCANN J., POLLARD N. S.: Real-time gradient-domain painting. *ACM Transactions on Graphics (SIGGRAPH 2008) 27*, 3 (Aug. 2008).

[OBW*08] ORZAN A., BOUSSEAU A., WINNEMÖLLER H., BARLA P., THOLLOT J., SALESIN D.: Diffusion curves: A vector representation for smooth-shaded images. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)* (2008), vol. 27.

[OCDD01] OH B. M., CHEN M., DORSEY J., DURAND F.: Image-based modeling and photo editing. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), ACM, pp. 433–442.

[OH08] OLSEN JR. D. R., HARRIS M. K.: Edge-respecting brushes. In *UIST '08: Proceedings of the 21st annual ACM symposium on User interface software and technology* (New York, NY, USA, 2008), ACM, pp. 171–180.

[PTVF07] PRESS W. H., TEUKOLSKY S. A., VETTERLING W. T., FLANNERY B. P.: *Numerical recipes: the art of scientific computing*, 3 ed. Cambridge University Press, 2007.

[RKB04] ROTHER C., KOLMOGOROV V., BLAKE A.: "grabcut": interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph. 23*, 3 (2004), 309–314.

[RT09] RUSSELL B., TORRALLBA A.: Building a database of 3d scenes from user annotations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2009).

[RTMF08] RUSSELL B. C., TORRALBA A., MURPHY K. P., FREEMAN W. T.: Labelme: A database and web-based tool for image annotation. *Int. J. Comput. Vision 77*, 1-3 (2008), 157–173.

[Rus09] RUSINKIEWICZ S.: TriMesh2 software library, 2009. http://www.cs.princeton.edu/gfx/proj/trimesh2.

[SSN09] SAXENA A., SUN M., NG A.: Make3d: Learning 3d scene structure from a single still image. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 31*, 5 (May 2009), 824–840.

[Vis09] VISUAL COMPUTING LAB - ISTI - CNR: MeshLab, 2009. http://meshlab.sourceforge.net/.

[WH05] WITHER J., HOLLERER T.: Pictorial depth cues for outdoor augmented reality. In *Wearable Computers, 2005. Proceedings. Ninth IEEE International Symposium on* (Oct. 2005), pp. 92–99.

[ZDPSS01] ZHANG L., DUGAS-PHOCION G., SAMSON J.-S., SEITZ S.: Single view modeling of free-form scenes. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on* (2001), vol. 1, pp. I–990–I–997 vol.1.