The ModelCamera: A Hand-Held Device for Interactive Modeling

Voicu Popescu, Elisha Sacks and Gleb Bahmutov
Computer Science Department, Purdue University
{popescu|sacks|bahmutov}@cs.purdue.edu

Abstract

An important goal of automated modeling is to provide computer graphics applications with high quality models of complex real-world scenes. Prior systems have one or more of the following disadvantages: slow modeling pipeline, applicability restricted to small scenes, no direct color acquisition, and high cost. We describe a hand-held scene modeling device that operates at five frames per second and that costs $2,000. The device consists of a digital video camera with 16 laser pointers attached to it. As the operator scans the scene, the pointers cast blobs that are detected and triangulated to provide sparse, evenly spaced depth samples. The frames are registered and merged into an evolving model, which is rendered continually to provide immediate operator feedback.

1. Introduction

Many computer graphics applications involve complex real-world scenes. Modeling such scenes is extremely difficult. One challenge is to acquire depth. Depth acquisition methods are slow enough that only a few views can be acquired. Even with view planning, these are too few for complete coverage of complex scenes. Another challenge is to register depth and color from multiple views. A third challenge is to build a model that can be rendered interactively with standard graphics hardware. Due to these factors, modeling a complex scene takes days or even weeks. The high cost in time, equipment, and logistics limits the role of scene modeling in computer graphics.

We have designed a hand-held scene modeling device that operates at five frames per second and that costs $2,000. The *ModelCamera* (Figure 1) consists of a video camera with 16 laser pointers attached to it. As the operator scans the scene, the laser beams produce blobs in the video frames where they hit scene surfaces. The frames are read into a computer, the blobs are detected in the frames, and their 3D positions are inferred by triangulation. Each frame is registered with respect to the previous frame using the color data and the blob positions. The registered frames are merged into an evolving model that is rendered to provide immediate operator feedback.

Depth acquisition is fast because only a few depth samples are acquired per frame. Moreover the laser pointers are fixed with respect to the camera, hence the blobs fall on known epipolar segments. Fast registration is performed with a novel algorithm based on dense color and sparse depth. View planning is avoided because data is acquired and registered in real time from a continuum of views.

Our modeling technique poses several research problems. The first problem is to register with sparse depth and without scene anchored fiducials. The second problem is to model complex geometry with sparse depth. The third problem is to merge the registered frames into the model in real time, which entails efficiently discarding the redundant, overlapping data. Our solutions rely on two fundamental properties of interactive modeling.

Coherent computation The blob detection and view registration algorithms exploit the fact that each video frame is similar to the previous frame. The search for a blob starts from the previous blob, which is normally within a few pixels of the answer. Registration is performed by minimizing an error function whose

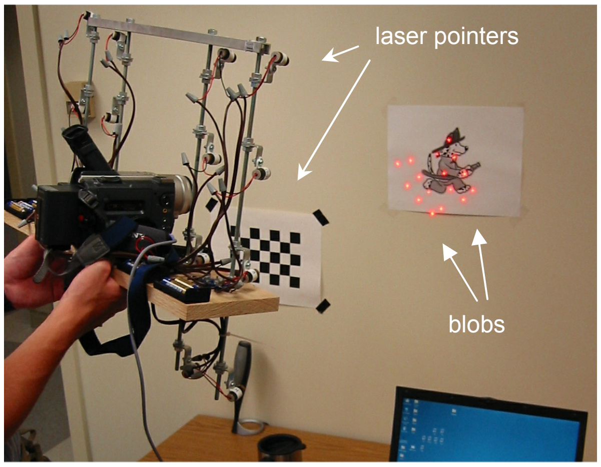

Figure 1 ModelCamera prototype.

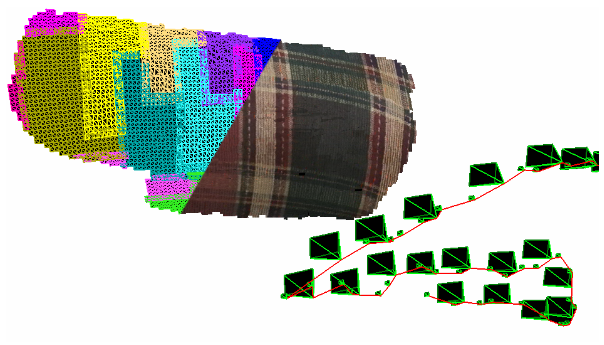

Figure 2 Model obtained from sequence of frames, and ModelCamera trajectory.

variables represent the camera motion between two frames. Coherence implies that an initial guess of constant velocity is close to the solution.

Interactive control The operator controls the system via a graphical interface that displays the current model, monitors its quality, and reports registration failures. If the color or depth sampling is insufficient, the system prompts the operator to bring the camera closer to the problematic surface. If registration is lost, it returns the operator to the last registered frame and modeling resumes from there.

Figure 2 illustrates modeling with the ModelCamera. The camera positions are shown with (green and black) frusta connected with a (red) line. Two out of three frusta are scaled down for clarity. We refer the readers to a video that we prepared to illustrate this paper [27]. The segments that illustrate the modeling were obtained by directly videotaping the computer monitor.

2. Prior work

The usual steps of automated modeling are view planning, depth acquisition, registration, and model construction. We review prior work in depth acquisition and in registration. We omit view planning, which is irrelevant to interactive modeling. We omit model construction because prior work presupposes off-line processing, whereas we require real-time algorithms. We then review hand-held modeling devices and image-based modeling methods.

Depth acquisition The main approaches are stereo, structured light, and (non-triangulation) laser rangefinding. Stereo is well suited for complex real-world scenes ([22], [15]), but correspondences cannot be established quickly enough for interactive modeling. Structured light approaches acquire accurate depth, but require calibrated motion and thus have a short range. Moreover the light pattern interferes with color acquisition. Time-of-flight (for example DeltaSphere [23]) and phase shift (Surphaser [24]) rangefinders acquire dense, precise depth images, but take several minutes per view. Some systems do not acquire color and those that do must register the color with the depth. This task is sometimes simplified by collocating the color camera with the depth sensor [10], but this comes at the cost of uneven color sampling.

View registration Registration can be performed with a separate device that tracks the position and orientation of the scene acquisition device. The relationship between tracking and depth measurement has been explored in [19] where a tracker was built using a real-time rangefinder. Coupling the acquisition device with a tracker has the disadvantages of limited range of motion, limited precision, and high cost. Another registration approach developed in the context of computer vision and augmented reality uses fiducials placed in the scene ([9], [14], [13]). The fiducials are easily detectable in the acquired frames and provide points with known 3D scene coordinates and known image projections, which are used for estimating pose. The fiducials have to cover the entire scene or have to be moved along with the acquisition device, hence are impractical for large scenes.

Most current registration methods are variants of the iterative closest point (ICP) algorithm ([1], [5], [21]), which iteratively minimizes the distance between the overlapping parts of two or more depth maps. Real-time ICP has been demonstrated on small objects using dense depth from structured light [17]. The method requires dense depth and is not applicable to our system.

Hand-held devices The advantages of interactive scene modeling have motivated the recent development of several hand-held devices. One type of device consists of a fixed camera and a mobile light-pattern source. One variant [20] uses a hand-held laser point projector on which three green LED's are mounted. The position of the LED's in the camera frame is used to infer the position and orientation of the laser beam. The red laser blob is detected in the frame and is then triangulated as the intersection between the pixel ray and the laser beam. Another variant [3] extracts depth from the shadow of a rod captured by a camera under calibrated lighting. The Autoscan [2] system uses two cameras mounted on a tripod and a hand-held laser point projector. The main problem with these systems is that they are limited to a single view by the fixed camera.

Hebert [7] describes a system where the operator can freely change the view. The device consists of two cameras and a cross-hair laser light projector. Frame to frame registration is achieved using a set of fixed points (fiducials) projected with an additional, fixed laser system. The system acquires depth over a narrow field of view at each frame, which implies long acquisition times for large scenes. The system also has the disadvantages associated with fiducials and does not acquire color. Rusinkiewicz [17] presents a structured light system where the object being scanned is hand-held in the fields of view of a fixed projector and fixed camera. The modeling pipeline is very fast; the object is modeled in real time and the evolving model is rendered to provide immediate feedback to the operator. The system is limited to the outside-looking-in modeling case, and does not acquire color, but it clearly demonstrates the advantages of real-time modeling.

Image-based rendering (IBR) We conclude the previous work discussion with a review of IBR approaches for modeling real-world scenes. Panoramas [4] are created by stitching together photographs from a single center of projection. The scene can be rendered efficiently and realistically from the center of the panorama. Other viewpoints cannot be rendered because the panoramas lack 3D information. Concentric mosaics [18] are an alternate representation that provides partial parallax with models that are built by moving a camera in calibrated circles. Another IBR approach uses images enhanced with per-pixel depth (*depth images*) to produce novel views by 3D warping [12]. Current depth-acquisition technology cannot provide enough depth images for large scenes. We use the depth image representation for fast incremental transformation during registration and for frame merging during model construction. The light field approach ([8], [6]) uses image data exclusively to provide novel views of a scene, but unlike panoramas, supports arbitrary camera views. The color data is stored in a 4D database that grows impractically large when modeling complex scenes.

3. The ModelCamera system

The device is assembled from a digital video camera, 16 laser pointers, and standard components (brackets, wires, clips). The camera is high-end consumer-level: progressive-scan, 720 x 480 x 3 pixel resolution, $1,500 cost. The lasers are red (635 nm), have an emitting power of 5mW (class IIIa), a spot size of 6 mm/12 mm at 5m/15m, and cost $15 apiece. The camera is linked to a PC via a Firewire interface. The lasers are mounted in a matrix pattern around the camera and generate 16 distinct blobs in its field of view. Figure 3 shows the blob patterns when the camera is aimed at a wall at three different depths. For clarity, all but 4 laser beams are clipped. The convergent and then divergent setup of the lasers allows the operator to vary the sampling rate by zooming in and out. A quick scan from afar suffices for simple surfaces, such as walls, whereas a slow, close scan is needed for complex shapes.

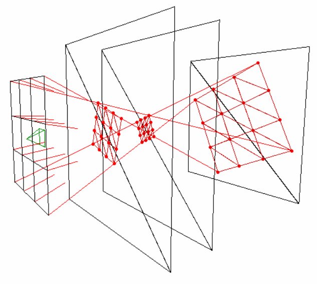
Figure 3 Laser beam pattern.

3.1 Calibration

The ModelCamera is calibrated in three steps that take 5 minutes and are fully automated except for moving the camera. The video camera is calibrated using the toolkit developed by Jean-Yves Bouguet [25] and included in Intel's OpenCV library [26]. The calibration error is in the order of 0.1 pixels. Using the toolkit, subsequent frames are undistorted with the calibrated coefficients. Next, two custom procedures establish the epipolar lines and the laser ray equations.

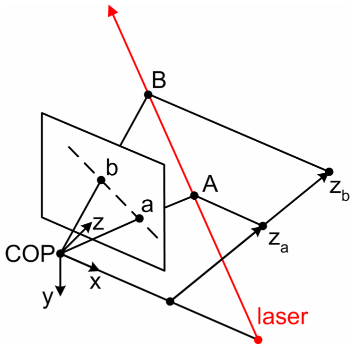

Figure 4 Epipolar line and triangulation.

The epipolar lines are the projections of the laser beams onto the camera image plane (line *ab* in Figure 4). They are determined from a sequence of frames in which the camera views a white wall. The blobs are found by an exhaustive search for intensity peaks. The frames are filtered then the intensity threshold is adjusted until 16 blobs of appropriate size are found in each frame. The blobs are assigned to lasers according to their horizontal and vertical ordering in the first frame and then using coherence. An epipolar line is least-squares fitted to each blob set. We use 200 frames. The mean/maximum distances between the blobs and the lines are 0.3/0.8 pixels. There is no visible systematic error. The epipolar lines are insensitive to residual radial distortion because they are close to the image center

The laser rays are determined from a sequence of frames in which the camera moves towards a calibration checkerboard. At each frame, the blobs are detected on their respective epipolar lines using the algorithm described below and the camera pose is inferred from the checkerboard (with the lasers turned off to prevent interference) using Bouguet's toolkit. The 3D position of each blob is determined by intersecting the camera ray with the checkerboard plane. A 3D ray equation is least-squares fitted to the point set of each laser. We use 10 frames; the mean/maximum distances between the points and the rays are 1.5 / 3.0 mm.

3.2 Depth acquisition

The ModelCamera acquires one depth sample per blob per frame by finding the blobs on the epipolar lines

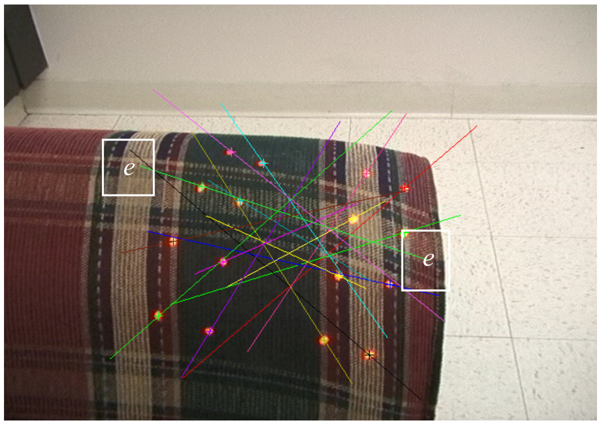

Figure 5 Frame with blobs detected on epipolar lines.

(Figure 5) and triangulating their 3D positions in camera coordinates (Figure 4). The blob detector searches the epipolar lines for intensity peaks. We exploit coherent camera motion by starting the search at the peaks from the previous frame. The current peaks are normally detected near the previous ones with minimal search. This heuristic fails when a blob jumps from one surface to another, and its entire epipolar line is then searched.

Detecting the blobs robustly is crucial. The first priority is to avoid false positives because they disrupt registration and severely distort the model. Minimizing false negatives is a lower priority because registration works well with 12–16 blobs. The intensities are bilinearly interpolated along the epipolar line and are smoothed with a 1D raised cosine filter. Peaks must exceed a threshold, which varies linearly along the epipolar line to allow for dimmer blobs at a distance. The intensity must fall off rapidly on either side of the peak. If a peak passes these tests, we test that the surrounding bright region is roughly circular along 8 evenly spaced spokes centered at the peak. Figure 6 shows the intensity along epipolar line *e* seen in Figure 5. Four peaks pass the epipolar line tests but F_1, F_2, and F_3 fail the symmetry test.

Each epipolar line intersects several other lines, so several blobs can fall on a single epipolar line. This can lead to ambiguity in assigning the blobs when one or more entire epipolar lines are searched. In this case, the unambiguous blobs are assigned first, which leads to a unique assignment for the rest of the blobs. Blob detection works well on our test scenes: 99.3% success at 70 cm and 85% at 200 cm.

Depth accuracy The depth accuracy is a function of the blob detection accuracy. The inner 4 / outer 12 lasers have an average baseline of 12 cm / 22 cm. For these two baselines, a one-pixel blob detection error translates into a depth (*z*) error of 0.1 cm / 0.2 cm at 50 cm, 0.35 cm / 0.7 cm at 100 cm, 1.5 cm / 3 cm at 200 cm and 3.5 cm / 7 cm at 300 cm. We increase blob detection accuracy by supersampling the epipolar line four times per pixel. If the peak is flat (saturated to 255), the midpoint is used.

We estimated blob detection accuracy by scanning a white wall from several distances and measuring the out-of-plane displacements of the triangulated 3D points. At 200 cm, the average/maximum displacements were 0.33 cm/1.1 cm, which indicates a blob detection error of 0.5 pixels in the absence of systematic errors. Better results were obtained at shorter distances. More work is required to quantify the blob detection accuracy as a function of surface properties (color, texture, specularity), angle of incidence, and laser properties.

3.3 Depth-then-color registration

Before a frame can be added to the scene model, it has to be registered with respect to the prior frames. Registration is performed by computing a camera motion that minimizes the depth and color errors between the current and the previous frame. This is a six-dimensional nonlinear optimization problem whose variables are the camera's degrees of freedom. A good starting point for the optimization is obtained by assuming a constant camera velocity and extrapolating from the previous pair of frames. The challenge is to find an error function and a minimization algorithm for fast, accurate registration.

We have developed the depth-then-color algorithm (Figure 7) that achieves good registration in real time. The motion is expressed as $p' = t + q \times p$, where p is a point obtained by triangulation in frame $i+1$ coordinates, p' is the same point in frame i coordinates, $t = (a, b, c)$ is a translation vector, and $q = (u, v, w, s)$ is a unit quaternion with $s = \sqrt{1-u^2-v^2-w^2}$. (The rotation angle is less than 90 degrees by coherence, which implies that s is positive.) The first stage minimizes a depth error function over the

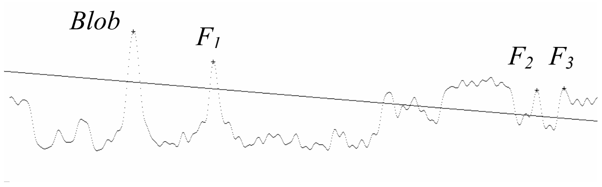

Figure 6 Intensity along epipolar line with blob and false peaks. Line indicates threshold.

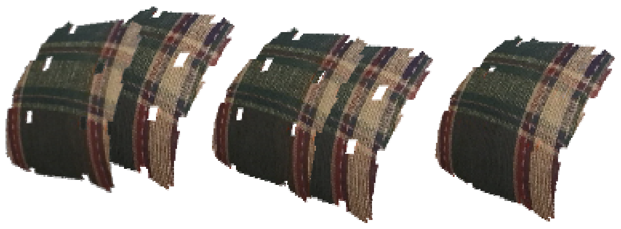
Figure 7 Consecutive frames before reg. (left), after depth reg. (middle), and after color reg. (right).

vector $m = (a, b, c, u, v, w)$. The minimum is obtained quickly because the function is smooth and involves only 16 points.

Symmetric surfaces have motions that do not affect the depth error, for example translation parallel to a plane or rotation around the center of a sphere. With the exception of planes, symmetric surfaces are uncommon, but approximately symmetric surfaces pose the same problem. The depth error varies too little to allow the depth registration stage to establish all six degrees of freedom. For example, Figure 8 shows a couch armrest that is locally cylindrical, so the depth error is almost constant when the camera translates along it or rotates around it.

The invariant motions are linear combinations of 1, 2 or 3 components of m. The second stage computes those variables by minimizing a color error function. The function is more expensive to evaluate than the depth error function because it involves thousands of pixels. Moreover it has many local minima, which increases the number of error evaluations required for convergence. Real-time minimization is achievable for three variables, but is impossible for six. Hence, the depth stage fully registers asymmetric surfaces and makes color registration practical for symmetric surfaces.

Depth registration The depth error and the symmetries are computed from quadratic surfaces that are fitted to the triangulated 3D points (blob points). In frame 1, a single quadratic, $z = f(x,y) = k_0 + k_1x + k_2y + k_3 x^2 + k_4xy + k_5y^2$, is least-squares fitted to all the blob points. In frame $i+1$, each set of blob points that shares an i surface is refitted to an $i+1$ surface then the unassigned blob points are fitted. A fit succeeds if the maximum distance from a blob point to the surface is less than the depth resolution (e.g. 0.5 cm at 100 cm). The blob points are replaced with their projections on the fitted surface. If a fit fails, the worst blob is discarded and the fit is retried. The cycle ends in failure when fewer than 8 blobs remain. The system discards the frame and prompts the operator to bring the camera closer to the scene.

The depth error of blob point p is defined as $g_p = p'_z - f(p'_x, p'_y)$. It equals zero when p is on the surface and is nonzero otherwise, so g_p^2 has a global minimum when p is on the surface. (Although g_p does not equal the distance from p to the surface, the two functions are approximately equivalent near zero.) The depth error of frame $i+1$ is $e(m) = \sum_p g_p^2$ with the summation over the assigned blobs.

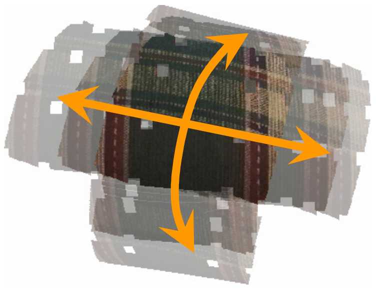
Figure 8 Depth-invariant degrees of freedom.

The symmetric surfaces that we model are planes, spheres, cylinders, and cones. We fit each type of surface to the blobs and test if the depth error is approximately constant as the blobs move along the symmetry axes. A symmetry is accepted if the depth error of the maximum allowable one-frame motion (5cm per translation; 0.2 radians per rotation) is less than the depth resolution. The depth error is computed in a coordinate system in which the symmetry axes coincide with coordinate axes. The variables that represent motion along symmetry axes are set to zero and are treated as constants during depth error minimization.

The depth error is minimized by sequential quadratic programming. This is the natural choice because e is smooth and its first and second derivatives are easy to compute. The iteration finds a zero of the gradient ∇e by repeatedly solving $H\partial m = -\nabla e$ where H is the Hessian matrix of e. This linear system is solved by singular value decomposition, which is more robust than the standard LU decomposition.

Color registration The remaining variables are computed by color error minimization. The error of pixel p in frame $i+1$ is defined as the RGB distance between its color and the color where it projects in frame i. The i color is computed by bilinear interpolation. The color error is hard to minimize because small camera motions produce rapid, erratic changes in its value. We reduce the variability by convolving the relevant region in each frame with a constant 11-by-11 filter (Figure 9 *a* and *b*). This is done efficiently by computing the convolution sum incrementally: for each pixel the sum of the least recent filter column is replaced with the sum of the new column. We select a set of pixels in frame $i+1$ and minimize the sum of the squares of their color errors by the downhill simplex method. This method is the natural first choice because the gradient of the error function cannot be computed analytically.

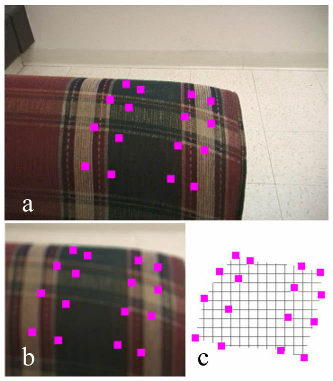

Figure 9 Frame with masked blobs (a), filtered region (b) and registration pattern (c).

The pixels are selected from every twentieth row and column of the axis-aligned bounding box of the blobs. The rows and columns are split into segments (Figure 9 *c*). A segment is a maximal sequence of pixels that are in the blob region, are blob free and lie on a single quadratic. The blob region is defined by the 18 triangles formed by the 3 x 3 groups of 4 immediate-neighbor blobs (Figure 3). Blob region membership is tested per pixel. The triangle containing the previous pixel is tested first, which reduces the number of tests. Blob pixels are excluded because their color comes from the lasers, rather than from the surface. The segment must lie on one quadratic, so that its *z* values can be computed.

The registration pattern is built once per frame, which takes negligible time. Every pixel (1000 - 4000), has to be projected into frame *i* for every error evaluation. We do so efficiently by incrementally 3D warping the horizontal and vertical segments, which has an amortized per pixel cost of 3 adds, 5 multiplies and 1 divide [11]. Warped-image reconstruction is unnecessary for error evaluation, so this approach does not incur the full cost of IBR by 3D warping [16]. Segments of planar patches are projected by affine transformation followed by a perspective divide (texture mapping), which evaluated incrementally requires only 3 adds, 2 multiplies and 1 divide per pixel. In the planar case, the perspective divide could be avoided if the frames were unprojected to a regular surface-defined orthographic grid first. Then a pixel could be transformed with only 2 adds, but this advantage is counterbalanced by the disadvantage of having to unproject *all* the pixels in the relevant region of frame *i*.

For an armrest (Figure 5) sequence of 63 frames, registration succeeded on 59 frames with average / maximum per times of 70 ms / 172 ms. The color registration pattern consisted of 1946 / 2709 samples per frame. The depth error was 0.066 cm / 0.201 cm per frame. The depth error for a frame was measured as the average of the depth errors of the depth-and-color samples in the color registration pattern. The depth error of a sample is given by the distance from the sample to the previous frame surface after registration. The depth error after depth registration (and before color registration) was 0.051 cm / 0.202 cm, which verifies that the degrees of freedom found using color do not affect depth. The color error was 2.8 / 5.5 versus 11.3 / 22.9 before color registration. The larger final errors are occur for the frames where the camera adjusts the picture brightness. The automatic compensation does not disturb registration and is necessary for acquiring high-quality textures.

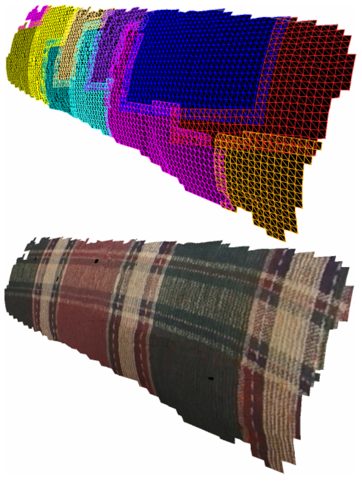
Figure 10 Surface model; wireframe and textured.

The depth-then-color registration algorithm is inappropriate for fragmented scenes where the blobs jump from tiny surface to tiny surface at every frame. Such scenes are challenging for any automated modeling system not only during registration but also during model construction. We present our solutions for such scenes in section 3.5 after we discuss our real-time incremental modeling algorithm for structured scenes.

3.4 Incremental modeling

Each registered frame contributes one or several quadratic patches with color, like the ones seen in Figure 7. As in color registration, the patches are delimited by the triangles formed by neighboring blobs. Since a new patch is acquired and registered approximately every 200 ms, the patches have considerable overlap. The first task of the incremental modeling algorithm is to eliminate the redundant data and append the contribution of the current frame to the scene model. This is done efficiently by representing the frame patches and the evolving model with depth images. Figure 10 shows a surface modeled with several depth images.

The depth image of a patch (*PDI*) is built similarly to the color registration pattern. Every row is used, so no vertical segments are needed. The color comes from the frame. The depth is inferred from the quadratic. Each scene surface is represented by one or several depth images (*SDIs*), which are created on demand as the scanning progresses.

The incremental modeling proceeds according to the algorithm sketched in Pseudocode 1. The 3D bounding box of the *PDI* is projected in all *SDIs* to establish the *SDIs* potentially affected by the current frame. Each depth-enhanced pixel *P* of *PDI* is warped incrementally to each *SDI* in the set. If *P* is clipped by the SDI frustum or maps to a location *P'* that contains a sample clearly closer or farther than *P*, the next *SDI* is considered. A sample at *P'* at approximately the same depth as *P* indicates redundant sampling. The algorithm selects the best sample and discards the worst. A sample is preferred if it originates from a

```
For each frame
  Build PDI
  Find affected SDIs
  For each segment S of PDI
    For each pixel P of S
      For each SDI in SDIs
        P' = P warped to SDI
        If P outside SDI next SDI
        If P' > P next P
        If P' < P P' = P, next P
        If P' is empty and no $A_P$
          $A_P$ =(SDI, P'), next SDI
      End for each SDI
      If $A_P$ commit $A_P$, next P
      Create new SDI, assign P
    End for each pixel
  End for each segment
End for each frame
```
Pseudocode 1

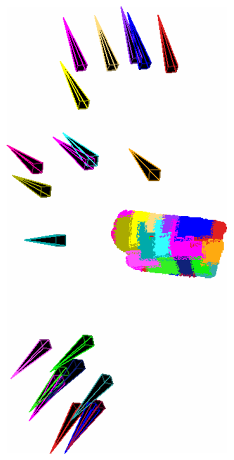
Figure 11 Depth image placement.

patch with a sampling rate closer to the desired sampling rate. We compute one sampling rate per segment (measured in pixels / cm) as the average of the sampling rates at the two end points. Each *SDI* stores the sampling rate of each pixel in an additional buffer. If there is no sample at *P'* and no potential assignment has been yet found for *P*, *SDI* and *P'* are recorded as a potential assignment A_P.

If *P* has not been used by any *SDI,* the potential assignment is finalized. To avoid holes in the constructed surface, P is splatted in the *SDI*. Splatting is a warped-image reconstruction technique that approximates the footprint of a warped sample [11]. For efficiency, we use square splats with a size derived from the sampling rate of the sample. If no potential assignment has been found, a new *SDI* is constructed with the look-at vector given by the surface normal at *P*. The *SDIs* sample the surface at the desired rate, and have a narrow field of view (10 degrees) for uniform sampling. The *SDI* frusta are shown in Figure 11. Pixels that warp to the border of an *SDI* are also assigned to a second *SDI* to avoid gaps (Figure 10).

The parts of the *SDIs* that are populated with samples are triangulated into a regular mesh. The texture mapped triangles are rendered by the graphics hardware. We use *SDIs* of 256 x 256 pixels and a triangulation step of 8 pixels, which yields 2K triangles and 256 KB of texture per *SDI*. Current graphics hardware can easily handle 100 *SDIs*. If necessary, the geometry load could be reduced by triangulating adaptively.

For the 63 frame sequence discussed earlier, the average / maximum model construction time was 63 ms / 125 ms per frame and 19 *SDIs* were built. Depth extraction, registration, model construction and rendering took 145 ms / 297 ms per frame. The total frame time was 209 ms / 391 ms. The time spent outside the modeling module is mainly due to frame transfer and undistortion.

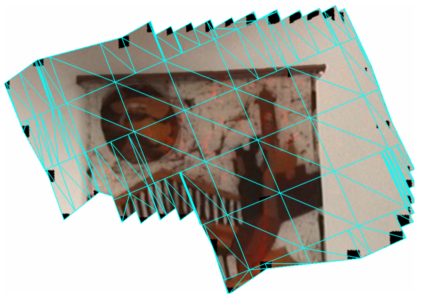
Figure 12 Planar scene model.

For planar surfaces, per-pixel depth is unnecessary. A planar surface is modeled with a set of texture-mapped squares, called *tiles* (Figure 12). The algorithm is similar to the case of curved surfaces, so we will not describe it in detail. The resolution of the texture is given by the desired sampling rate. The tiles are created on demand as scanning progresses. The frame patch is represented by the set of triangles connecting the blobs. The textures of the tiles affected by the current patch are grown by rendering the patch triangles in the textures. The geometry of partially covered tiles is modeled with a triangulated polygon. The polygon is unioned with each new patch and is re-triangulated. When the tile texture is filled (we use a threshold of 99%), the polygon is replaced with the two triangles defining the tile.

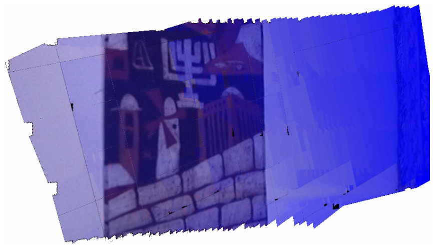
Figure 13 Sampling rate visualization.

The model of the part of the scene currently scanned is rendered continually and provides immediate feedback to the operator. The operator can select a mode where the sampling rate is visualized. Red / blue highlights indicate over- / under- sampling (Figure 13).

3.5 Fragmented scenes

The depth-then-color algorithm does not work for fragmented scenes because the frame cannot be approximated with a few quadratics. The frames can be registered using color only if the ModelCamera rotates about its center of projection. Except for having to mask the blobs, the procedure is identical to stitching photographs together to form panoramas [4]. To achieve real-time registration we use a registration pattern consisting of horizontal and vertical pixel segments similar to the one described earlier. The pixels are transformed from the current frame to the previous frames incrementally with an affine transformation followed by a perspective divide.

A sequence of registered frames is transformed in a cubical panorama. The triangulated blob points are projected onto the faces of the panorama. The cubical panorama is unfolded and the projections are triangulated. Triangles that cross from one panorama face to another are divided along the edge. Each face defines a texture map that is applied to its corresponding triangles. Figure 14 shows the frames registered in real-time as the operator rotates the camera (top) and the depth-enhanced panorama (bottom) that is computed in 5 seconds after the sequence is scanned.

Because the scene is close to the camera residual translation affects registration. We use a tripod to avoid translation. The tripod slows down acquisition because it has to be repositioned. It does not allow tilting the camera about its COP so the operator cannot cover the entire panorama with blobs.

4. Discussion

We have presented an interactive modeling device based on sparse depth and dense color. The ModelCamera acquires 16 depth samples per frame, registers the frames, merges them into an evolving texture-mapped scene model, and renders the model for operator feedback. Structured scenes are modeled via a freehand scan, while unstructured scenes require a tripod. We have demonstrated fast, accurate modeling of surfaces. Our immediate goal is to model room-sized scenes.

The depth-then-color algorithm requires more color variation than some scenes contain. We can compensate by deriving more degrees of freedom from geometry. If two surfaces are visible, at most one degree of freedom needs to come from color. If three are visible, color is not required. The 16 laser configuration of the ModelCamera is barely adequate for two surfaces, since we need 8 blobs per surface for robust quadratic fitting. We will experiment with more lasers in the next design cycle.

We must also address registration drift over long sequences of frames due to depth and color error accumulation. We found little drift on sequences of 20 frames. For example, the wall hanging in Figure 12 measures 45.1 cm by 83.2 cm in the model versus 46.2 by 83.9 cm in reality. But much longer sequences are required for room modeling. If drift hinders modeling, it must be monitored interactively, perhaps by computing the color error of each registered frame relative to the current model. Otherwise, it can be corrected offline.

We will improve the ModelCamera based on our experience with the first prototype. The new design will be more rigid to prevent laser motion relative to the camera, which degrades blob detection. It will be modular to support experiments with alternate cameras and lasers. We are considering designing a light-weight camera mount with shoulder straps that allows panning and tilting the camera around its COP. This will improve the maneuverability of the ModelCamera during the acquisition of fragmented surfaces. Eventually, we plan to move the graphical interface onboard the video camera to improve mobility. We could use the LCD of the video camera, a wearable display, or a wearable computer. A better interface is also a priority. It should help the operator undo frames, start a scan at the end of a prior scan, and hide / show / save / load parts of the model.

Another goal is to model outdoor scenes. The current lasers are invisible in sunlight. Greater power or alternate wavelengths are a possibility, but would make the device eye unsafe. Interactive modeling of surfaces with view dependent appearance is a long term goal. Blobs are difficult to detect, highlights and reflections confuse color registration, and the current models have to be extended.

5. Acknowledgements

We would like to thank Chun Jia for her help with implementing the panorama registration and construction, Chris Hoffmann for useful discussions. This work was supported by NSF grant IIS-0082339 and Intel Corporation.

References

[1] P. Besl, N. McKay. A method for registration of 3-d shapes. IEEE Trans. Patt. Anal. Mach. Intell., 14(2):239-256, 1992.

[2] N. A. Borghese et al., Autoscan: A Flexible and Portable 3D Scanner, IEEE Computer Graphics and Applications, Vol.18, No.3, May/ Jun. 1998, pages 38-41.

[3] J.-Y. Bouguet and P. Perona, 3D Photography using Shadows in Dual-Space Geometry, International Journal of Computer Vision, Vol. 35, No. 2, Nov. 1999, pages 129-149.

[4] S. Chen. QuicktimeVR- an image-base approach to virtual environment navigation. In Proc. SIGG. '95, pages 29-38.

[5] Y. Chen and G. Medioni. Object modeling by registration of multiple range images. Image and Vision Computing, 10(3):145-155, 1992.

[6] S. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen. The lumigraph. In Proc. of SIGGRAPH '96, pages 43-54, 1996.

[7] P. Hebert. A self-referenced hand-held range sensor. In Proceedings of Third International Conference on3-D Digital Imaging and Modeling, pages 5-12, 2001.

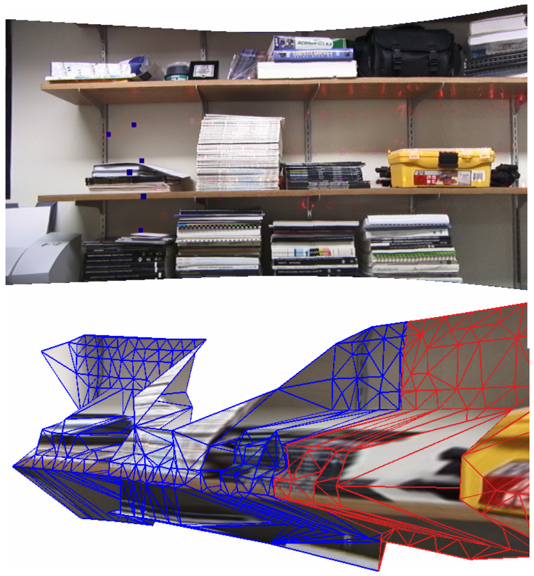

Figure 14 Registration (top) and model (bottom) of fragmented scenes.

[8] M. Levoy and P. Hanrahan. Light field rendering. In Proc. of SIGGRAPH '96, pages 31-42, 1996.

[9] R. W. Malz, "High Dynamic Codes, Self-Calibration and Autonomous 3D Sensor orientation: Three Steps towards Fast Optical Reverse Engineering Without Mechanical CMMs", in *Optical 3-D Measurement Techniques III*,Gruen/ Kahmen eds., Wichmann, 1995, pages 194-202.

[10] D. McAllister, L. Nyland, V. Popescu, A. Lastra, C. McCue. Real-Time Rendering of Real-World Environments. Proceedings of the Eurographics Workshop on Rendering, June 21-23, 1999.

[11] L. McMillan. An image-based approach to three dimensional computer graphics. Ph.d., University of North Carolina at Chapel Hill, 1997.

[12] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In Proc. SIGGRAPH '95, pages 39-46, 1995.

[13] U. Neumann and Y. Cho. A Self-Tracking Augmented Reality System. ACM International Symposium on Virtual Reality and Applications, pages 109-115, July 1996.

[14] W. Niem and J. Wingbermuhle, Automatic Reconstruction of 3D Objects Using a Mobile Monoscopic Camera, in *Proc. of the First International Conference on 3-D Digital Imaging and Modeling*", Ottawa, Canada, Oct. 1997, pages 173-180.

[15] M. Pollefeys and L. Van Gool. From Images to 3D Models, Communications of the ACM, July 2002/Vol. 45, No. 7, pages 50-55.

[16] V. Popescu et al. The Warpengine: An architecture for the post-polygonal age. In Proceedings of SIGGRAPH 2000, pages 433-442, 2000.

[17] S. Rusinkiewicz, O. Hall-Holt, and M. Levoy. Realtime 3d model acquisition. SIGGRAPH, 2002.

[18] H. Shum and L. He. Rendering with concentric mosaics. In Proc. SIGGRAPH '99, 1999.

[19] D. Simon, M. Hebert, and T. Kanade. Real-time 3-d pose estimation using a high-speed range sensor. In IEEE Int. Conf. Robot. Autom., pages 2235-2241, San Diego, 1994.

[20] M. Takatsuka et al., Low-cost Interactive Active Monocular Range Finder, in Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition, Fort Collins, CO, USA, Jun. 1999, pages 444-449.

[21] Z. Zhang. Iterative point matching for registration of free-form surfaces. Int. J. of Comp. Vision, 13(2):119-152, 1994.

[22] C. Zitnick and T. Kanade. A cooperative algorithm for stereo matching and occlusion detection. IEEE Trans. on Patt. Anal. and Mach. Intell., 22-7, July, 2000, pages 675 - 684.

[23] http://www.3rdtech.com

[24]http://www.surphaser.com

[25] http://www.vision.caltech.edu/bouguetj/calib_doc

[26] http://www.intel.com/research/mrl/research/opencv

[27] http://www.cs.purdue.edu/cgvlab/modelCamera/