A Simple Approach for Point-Based Object Capturing and Rendering

Miguel Sainz, Renato Pajarola, and Albert Mercade
University of California, Irvine

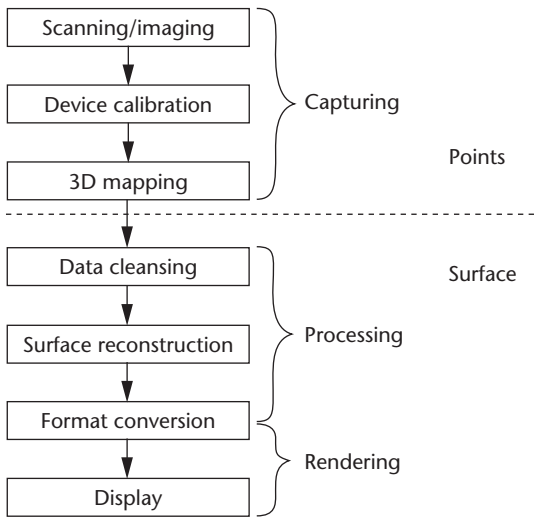
Antonio Susin
Technical University of Catalonia

A complete, automatic pipeline captures, processes, and renders point-based models from real objects. Input is a video sequence showing different object perspectives.

Point-based object representations are a powerful alternative to traditional polygonal object representations. Points are, in fact, the natural raw output data from the capturing stage in most 3D geometry acquisition systems, as Figure 1 shows.

Capturing 3D geometry is a mission-critical content acquisition technique in application domains such as virtual reality, CAD/CAM, and physical asset management. The advantage in reducing the geometry to points is that it permits various processing and display simplifications. Furthermore, a closed and continuous surface may not always be necessary for the visualization tasks involved in a given application. However, generating usable point-based models isn't easy, particularly with a system built around simple, low-cost components.

In this article, we describe SPOC, a simple, point-based object capturing system we've developed to automatically capture and process 3D geometry for point-based image rendering. We also identify the problems involved and describe the technical solutions we've implemented. Finally, we address several algorithmic issues in capturing point models using a simple digital camera and turntable setup, and in processing and rendering point clouds.



1 Points within the conceptual stages of a traditional 3D shape acquisition pipeline.

Capturing system

Unlike complex capturing systems that use active sensor devices or computer-controlled gantries, our point-based object capturing system is built on top of a low-cost hardware setup, as Figure 2 shows. The system consists of a digital camera (Web cam, camcorder, or still camera) and a turntable with calibration fiducials. This simple, inexpensive hardware configuration, together with image-based computer vision techniques, makes our proposed solution widely accessible—the components aren't complex and they're easy to find, install, and use.

The input to our proposed system is a video sequence of images taken when we rotate the turntable 360 degrees or move the camera around the physical object to be reconstructed as a cloud of 3D points. The reconstruction algorithm should guarantee that the reconstructed model will closely match the original images when seen from each of the reference image viewpoints. This approach addresses a well-known limitation of image-based model reconstruction algorithms, because information not captured in the images can't be reconstructed without additional assumptions. Consequently, the more images provided, and more importantly, the more coverage there is of the object's surface, the better the reconstruction.

We've designed the capturing system as an automatic process. First, the system feature tracks the image sequence to obtain the 2D positions of the scene's fiducials and their correspondence relations between frames. Next, the system uses the 2D tracked information to obtain a 3D scene structure (it finds the 3D positions of the point fiducials and the camera's position, orientation, and internal parameters for each image). Finally, the system obtains a dense point sampling of the

 Published by the IEEE Computer Society

2 Two examples of the proposed setup for simple object capturing setup with digital camera and turntable.

object's surface by conducting an image-based volumetric analysis of the scene.

Feature tracking

Our system uses off-the-shelf automatic feature-trackers[1] to identify features in consecutive frames j and $j + 1$. For the turntable setup, we add artificial fiducials to the scene to help detect features wherever the objects to reconstruct present low-frequency textures, which would otherwise complicate feature detection.

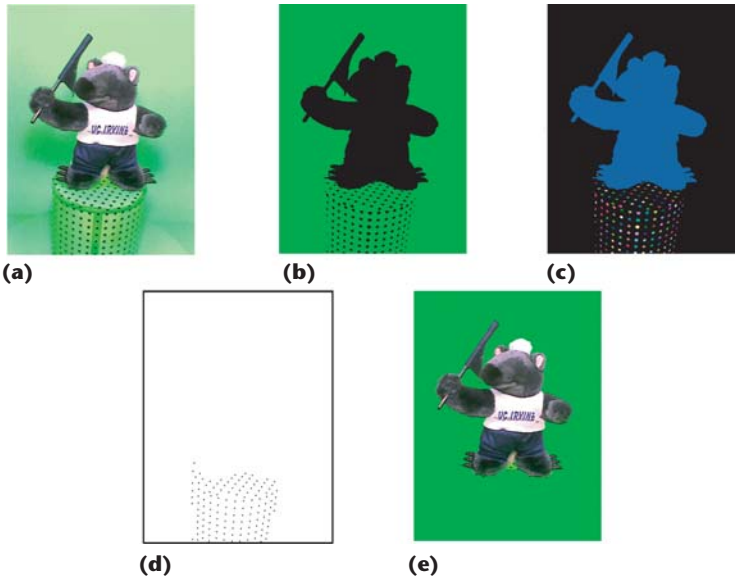
We generate 2D feature candidates in each frame by locating a simple pattern on top of the turntable (see Figure 2) and applying an automatic image-background and object-removal filter based on computer vision techniques. Region analysis techniques are used to detect the fiducials, which are then replaced by their centers, since they're circular marks, and tracked with motion prediction and neighborhood analysis.

We can also use region analysis to remove all fiducials from the original sequence, leaving clean segmented images of the object, which facilitates the reconstruction process. Figure 3 shows an example of fiducial tracker inputs and outputs.

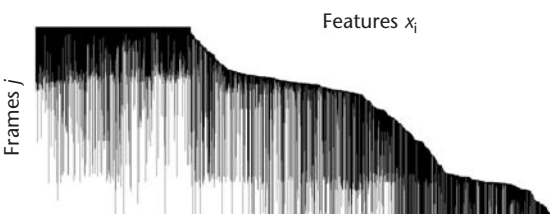
Our system yields a set of 2D coordinates $\mathbf{x}_i$ and their correspondence relations between consecutive frames, much like an automatic tracker's typical output. These measurements are assembled together into the measurement matrix $\boldsymbol{W}$, where columns represent the position of each detected feature in all the frames and rows, grouped into triads representing the visible features in one frame. In general, for long sequences the tracker's output can't be used without further processing because of occlusion and outliers.

To address occlusion, we use the common approach of having the tracker track a fixed number of features. If a feature is lost due to occlusion, the tracker finds a new replacement feature. As a result, the measurement matrix $\boldsymbol{W}$ is only partially filled, as Figure 4 shows. In the next section, we propose a segmentation into subsequences with common features.

Outliers are noisy or not properly tracked features that derange the calibration system. To minimize this problem, we use the random sampling consensus approach, a robust estimator that detects and eliminates outliers. This iterative approach produces a robust set of in- and outliers used for keyframe selection.



3 Stages of feature tracking: (a) original image input, (b) background/foreground mask, (c) region analysis result, (d) detected fiducials, and (e) final segmented image.



4 Partially filled measurement matrix.

We use the percentage of feature outliers as a measurement of how good the correspondence is between frames j and $j + 1$. Whenever this value is lower than a given threshold, the pair of images is considered to have an underlying motion not mappable by a 2D homography, and we mark frame $j + 1$ as a keyframe.

Camera calibration

An important step in capturing models of real objects from a set of still images or video sequences is scene cal-

In our case, object reconstruction isn't based on dense point correspondences but on a volumetric reconstruction …

ibration, which consists of obtaining the configuration of the camera's intrinsic parameters—focal length, principal point, and optical aberrations, for example—as well as extrinsic parameters—3D position and orientation relative to the object.

In this context, the most common approach is to perform 2D feature detection and tracking over the image sequence, and to extract the calibration information by solving a set of nonlinear equations.

$$\boldsymbol{W} = \begin{bmatrix} \lambda_{11}\mathbf{x}_{11} \cdots \lambda_{1n}\mathbf{x}_{1n} \\ \cdots \quad \cdots \quad \cdots \\ \lambda_{m1}\mathbf{x}_{m1} \cdots \lambda_{mn}\mathbf{x}_{mn} \end{bmatrix} = \begin{bmatrix} \boldsymbol{M}_1 \\ \cdots \\ \boldsymbol{M}_m \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \cdots \mathbf{p}_n \end{bmatrix}$$

The solution of these equations provides the perspective camera parameter matrices $\boldsymbol{M}_i$ of all m frames and provides the 3D points $\mathbf{p}_i$ corresponding to the n tracked 2D features $\mathbf{x}_i$.

A major problem is calibrating long image sequences with many feature occlusions, which is the case for objects on rotating platforms. Furthermore, in continuous sequences, consecutive frames tend to reflect few scene changes. Recently,[2] we introduced a novel approach to solve these problems by a divide-and-conquer strategy that automatically divides the complete sequence into disjoint subsequences. This segmentation has two parts: finding the first and last frame of a subsequence to be calibrated, and finding a good set of keyframes within this sequence. Our system solves the first part by tracking a set of features $\mathbf{x}_i$ starting with a frame j until more than a given percentage of features $\mathbf{x}_i$ don't appear in frame $j + k$.

For the second part of segmentation, in each such image subsequence S_j, the system selects and calibrates a set of statistically robust keyframes, recovering both internal and external camera parameters as well as the scene's structure. The set of keyframes for a specific video segment S_j will contain frames j and $j + k$ as the lower and upper bounds, and all the frames in between that were selected during the random sampling consensus analysis.

We calibrate the video segment S_j based on the measurement matrix $\boldsymbol{W}_j$'s containing features visible in all frames. We need another outlier removal process to select the minimal set of measurements $\mathbf{x}_i$ that define a projective solution in all frames S_j. For this, we use an affine reconstruction algorithm[3] that presents an efficient closed-form least-squares solution, which removes the most obvious outliers. We then achieve an improved classification by performing an iterative projective reconstruction algorithm in the self-calibration step.

Self-calibration consists of computing a projective decomposition of the measurement matrix $\boldsymbol{W} = \boldsymbol{M} \cdot \boldsymbol{P}^{\mathrm{T}}$ containing the 2D coordinates $\mathbf{x}_i$ of the object features $\mathbf{p}_i$ projected according to the camera parameter projection matrices $\boldsymbol{M}_i$. This is achieved using a singular value decomposition of $\boldsymbol{W}$. We then upgrade from projective to metric reconstruction by recovering the projective distortion matrix, given that the projection matrix and the object features are known up to a projective homography $\boldsymbol{H}$, $\boldsymbol{W} = \boldsymbol{M} \cdot \boldsymbol{H} \cdot \boldsymbol{H}^{-1} \cdot \boldsymbol{P}^{\mathrm{T}}$.

In contrast to other approaches, our solution is essentially a linear process and doesn't require an initial estimation of the solution. Finally, we perform a nonlinear minimization over the reprojection error to improve the obtained solutions.

Point reconstruction

In our case, object reconstruction isn't based on dense point correspondences but on a volumetric reconstruction, which is more flexible in terms of baseline distance between views. We use a novel technique, inspired by a hardware-accelerated voxel carving approach,[4] to directly obtain a cloud of points on the surface of the volume occupied by the object.

We define the scene's bounding box, subdivide it into voxels to a desired resolution, and, using the segmented images from the video sequence, apply a visual-hull[5] technique to obtain the object's surface points. The visual hull can be described as the maximal shape that generates the same silhouette for every reference view, and its determination consists of removing all voxels of the volume corresponding to background color in any of the captured frames.

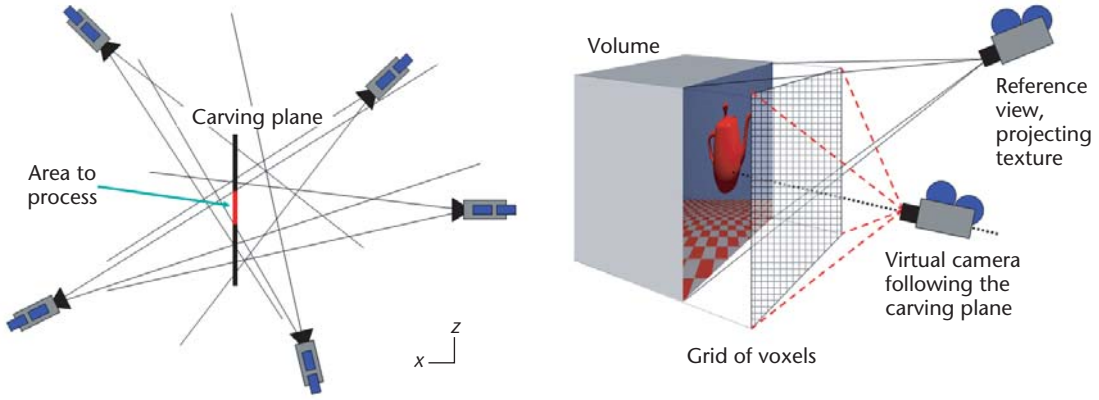
The visual-hull calculation consists of determining which voxels of the volume project on the foreground in every reference view. Using transparency, projective texture mapping, and stenciling, we implemented an algorithm to obtain the visual hull from a set of segmented images in real time.

The presented algorithm sweeps a carving plane along one of the bounding box axes to determine which voxels are contained in the visual hull and which are outside, as Figure 5a shows. The algorithm exploits the OpenGL API and uses the 3D hardware acceleration provided on typical video cards to perform the visual-hull approximation's most computationally expensive operations.

As Figure 5b shows, we situate a virtual camera at a constant perpendicular distance to the carving plane. During each sweep iteration, the camera projects the selected keyframes' images onto the carving plane, using projective texture mapping, and the system analyzes the result.

During initialization, our system clears OpenGL buffers and enables stencil and alpha tests to paint only the images' foreground. The system marks background pixels with an alpha value of 0 (fully transparent). Next, the system renders all views while the stencil buffer accumulates the number of times a given pixel has been painted. If all n views project foreground pixels to a voxel on the plane, the corresponding stencil values will be equal to the number of cameras, meaning that the voxel is visible as part of the object in all views; therefore, it belongs to the visual hull.

Once all n views have been projected, the system



(a) **(b)**

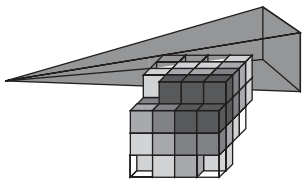
5 Visual hull concept. (a) shows the input views frusta intersection and how the sweeping plane traverses through it. (b) illustrates the use of projective texture mapping to project a reference view onto the carving plane, viewed from a virtual position perpendicular to it.

changes the stencil test function to pass only those pixels having the maximum value in the stencil buffer, and renders the carving plane. To easily identify which voxels of the carving plane belong to the visual hull, we define the OpenGL rendering buffer to match the voxel resolution in pixels. Accordingly, each pixel's 2D coordinates identify univocally which voxel is carved. Subsequently, every time a primitive is rendered, the system queries the visual hull using the calculated stencil buffer to determine the part of the primitive that's within the real object.

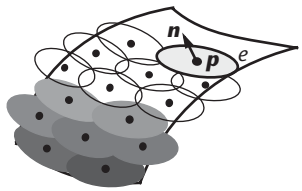
The main drawback of the visual-hull method is that when low-resolution images are used, such as those from a camcorder, the reconstructed volume presents linear bump stripes on the surface, due to the pixelization of the object's silhouettes. Inspired by Matusik and colleagues' work,[6] we can reduce these artifacts by adding a fuzzy boundary between the image's background and foreground. We can achieve the boundary by implementing alpha blending, just as if we'd considered each pixel to be an antialiased beam that perspectively intersects multiple voxels, as Figure 6 shows.

Subsequently, our method then directly obtains the surface points without performing a volumetric object reconstruction. This is achieved by the system's directly detecting the surface voxels for each carving plane slice via a simple 2D neighborhood analysis. During this analysis, the system detects which pixels of the visual hull slice have at least one of the four connected neighbors empty. The system extends this test to the full 3D neighborhood by considering the previous and the next slice. This approach requires only that we store three rendered images of consecutive visual-hull sections instead of a full voxelized volume.

Ultimately, the extracted geometry is a set of 3D point samples $\mathbf{p}_i$ of the captured solid object corresponding to the visible corners of the surface voxels. In addition, we estimate color attributes $\mathbf{c}_i$ by a weighted contribution from all frames j in which the sample $\mathbf{p}_i$ has been visible. If necessary, the voxel-based reconstruction can also provide coarse normal estimates $\mathbf{n}_i$, which may be used to consistently determine inside or outside pointing orientations.



6 Antialiased pixel beam intersecting multiple voxels.



7 Point-based object displayed as a set of elliptical disks discretely covering the object's surface.

Point processing

A point capturing system's outputs will ultimately be a set of 3D coordinates $\mathbf{p}_1, \ldots, \mathbf{p}_n \in \mathbf{R}^3$, possibly including color attributes $\mathbf{c}_i$, representing a densely sampled set of points on the object's surface. For display purposes, as Figure 7 shows, we must further process this point cloud data to determine the normal orientation $\mathbf{n}_i$ and elliptical disk e_i of each individual point sample $\mathbf{p}_i$. We'll describe the different tasks and our implementation choices in this section.

Local least-squares fitting

An elliptical disk e consists of major and minor axis directions $\mathbf{e}_1$ and $\mathbf{e}_2$ and their lengths. Together with the surface normal $\mathbf{n}$, the axis directions $\mathbf{e}_1$ and $\mathbf{e}_2$ define the local tangential coordinate system of a point sample $\mathbf{p}$. We can effectively estimate the normal vector $\mathbf{n}$ of a point by a local least-squares fit of a plane to the point cloud.[7] Such a local least-squares fit is determined by the principal component analysis (PCA) of the covariance matrix of the k-nearest neighbors around a given point. In fact, the eigenvectors of the covariance form a local orthogonal frame corresponding to the PCA—and thus define the vectors $\mathbf{n}$, $\mathbf{e}_1$, and $\mathbf{e}_2$. Moreover, the eigenvalues

Because of its noniterative nature and independency of mesh connectivity information, the fairing operator we introduced elsewhere is perfectly suited for application to point models …

determine the elliptical anisotropic distribution of points in the tangential coordinate system, and thus define the axis length aspect ratio along $\mathbf{e}_1$ and $\mathbf{e}_2$. Note also that the normal of a local least-squares fitted plane can be considered a smoothed normal due to the properties of least-squares fitting, even if computed from a noisy point set. This makes it suitable for the fairing step outlined below and which is susceptible to noisy normals.

Therefore, all parameter estimations are performed locally and based on the *k*-nearest neighbor information. We developed and implemented efficient algorithms for finding all *k*-nearest neighbors, computing the covariance matrix for each point and its neighbors, as well as determining the elliptical disk parameters.

All k-nearest neighbors

Finding all *k*-nearest neighbors of a set of points is a computationally intensive task for large *n*. To effectively cope with large point clouds, we solve this all-*k*-nearest-neighbor search problem efficiently using a balanced *k*D-tree with a longest-side split strategy.[8]

We can readily construct a balanced *k*D-tree with a divide-and-conquer approach. The method used in our system consists of first sorting the input points $\mathbf{p}_1, \ldots, \mathbf{p}_n$ along each coordinate axis into arrays $\mathbf{X} = \{x_1, \ldots, x_n\}$—as well as **Y** and **Z** for the other axis—containing only indices x_i to the actual data points $\mathbf{p}_i$. Then, starting with the root node, at each step the method determines the longest side of the current point set, uses the median along the corresponding axis as the split position (and removes it from the point set), and divides, then recursively processes, the points.

For each point $\mathbf{p}_i$ the system iteratively performs a *k*-nearest neighbor query on the previously constructed *k*D-search tree *T*. For efficient back-tracking, *T* is traversed by a depth-first in-order traversal. Then the system checks at each node *t* of *T* whether it can possibly contribute a new *k*-nearest neighbor, and if not, stops the recursive traversal. Otherwise, it updates the *k*-nearest neighbor set of $\mathbf{p}_i$, taking the split point of *t* into account, and recursively processes the child nodes of *t* in in-order succession.

Covariance analysis

The *k*-nearest neighbor set $K_i = \{\mathbf{p}_{i_1}, \ldots, \mathbf{p}_{i_k}\}$ for each point $\mathbf{p}_i$ is the basis for a local least-squares fit of a plane to the point cloud. The corresponding normal $\mathbf{n}_i$ is found from the PCA of the standard covariance matrix

$$\boldsymbol{M}_i = k^{-1} \sum_{j=1}^{k} \left(\mathbf{p}_{i_j} - \mathbf{p}_i\right) \cdot \left(\mathbf{p}_{i_j} - \mathbf{p}_i\right)^T$$

and a straightforward implementation requires the computation of $k \times n$ outer vector products.

To avoid extensive calculations of outer vector products, we exploit the transformation invariant covariance formulation introduced elsewhere.[9] In homogeneous space we can express the generic homogeneous covariance as

$$\overline{\boldsymbol{M}}_i = \sum_{j=i_1}^{i_k} \boldsymbol{C}_j \quad \text{with } \boldsymbol{C}_i = \mathbf{p}_i \cdot \mathbf{p}_i^T$$

This reduces the computation cost of outer products to only *n*, once only for each $\boldsymbol{C}_i$. Representing the subtraction of $\mathbf{p}_i$ as translation matrix $\boldsymbol{T}_i$, we get $\boldsymbol{M}_i = k^{-1}\boldsymbol{T}_i \cdot \overline{\boldsymbol{M}}_i \cdot \boldsymbol{T}_i^{\mathrm{T}}$. Our system further exploits this transformation invariance for elliptical splat size estimation, as we explain.

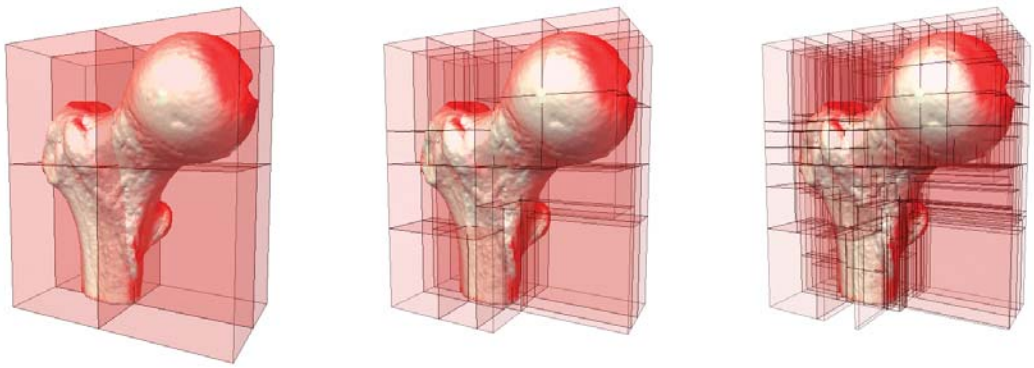
To get a normal estimation $\mathbf{n}_i$, we first compute the outer product $\boldsymbol{C}_i$ for each point $\mathbf{p}_i$ and then accumulate its generic homogeneous covariance in $\overline{\boldsymbol{M}}_i$. Second, we apply the translation by $-\mathbf{p}_i$ and division by *k* to $\overline{\boldsymbol{M}}_i$ to get $\boldsymbol{M}_i$. Third, for the PCA we perform a Jacobi transformation on $\boldsymbol{M}_i$, and the resulting vector corresponding to the smallest principal component denotes the normal direction $\mathbf{n}_i$. For improved numerical stability, we compute the splat axis in the tangent plane, as we explain later, instead of taking the eigenvectors from this numerical PCA.

In the last step we determine the ellipse axis $\mathbf{e}_1$ and $\mathbf{e}_2$ lying in a plane perpendicular to $\mathbf{n}_i$. For this we follow the steps detailed previously[9] and transform the coordinate system, and thus $\overline{\boldsymbol{M}}_i$, by a rotation $\boldsymbol{R}_i$ to align the *z*-axis with $\mathbf{n}_i$. Thus we get $\boldsymbol{M}_i = k^{-1}\boldsymbol{R}_i \cdot \boldsymbol{T}_i \cdot \overline{\boldsymbol{M}}_i \cdot \boldsymbol{T}_i^{\mathrm{T}} \cdot \boldsymbol{R}_i^{\mathrm{T}}$ given in tangent-space at $\mathbf{p}_i$. The 2D covariance in the tangent plane is then given by the upper-left 2×2 submatrix of $\boldsymbol{M}_i$. Moreover, we derive the ellipse axes $\mathbf{e}_1$ and $\mathbf{e}_2$ from the eigenvectors of this 2D covariance matrix, with inverse rotation $\boldsymbol{R}_i^{-1}$ applied. The ellipse aspect ratio is given by the eigenvalues, however, we must scale the major axis length to include all *k*-nearest neighbors K_i projected in the tangent plane as proposed.[9]

Fairing

The raw output of the 3D reconstruction phase in general exhibits a certain degree of roughness. In particular, the volume carving method we use suffers from aliasing artifacts due to the resolution mismatch between captured images (low) and 3D spatial discretization (high). To reduce these artifacts, we apply a fairing filter to the point data set, which we'll briefly outline. This fairing operation requires per-point normal and elliptical splat size information; therefore, our system performs the fairing after an initial splat parameter estimation step that computes normal and ellipse attributes as outlined. After the fairing operation, the system performs the splat parameter estimation again to compute the final result.

Because of its noniterative nature and independency of mesh connectivity information, the fairing operator we introduced elsewhere[10] is perfectly suited for application to point models, which contrasts with most other



8 A dense point sample set of a hip bone model, showing the subdivision planes corresponding to the point-octree space partitioning.

methods proposed in the literature. Given a point $\mathbf{p}_i$, we define the fairing operation Φ on points as

$$\Phi(\mathbf{p}_i) = k_i^{-1} \cdot \sum \Pi_j(\mathbf{p}_i) a_j \cdot f(\|\mathbf{p}_j - \mathbf{p}_i\|) \cdot g(\|\Pi_j(\mathbf{p}_i) - \mathbf{p}_i\|)$$

The operator takes neighboring points $\mathbf{p}_j$ into account with a Gaussian weight function f decreasing with distance $|\mathbf{p}_j - \mathbf{p}_i|$. The factor a_j further takes the elliptical area of point sample $\mathbf{p}_j$ into account. The operator $\Pi_j(\mathbf{p}_i)$ is the projection of $\mathbf{p}_i$ onto the tangent plane at $\mathbf{p}_j$ with normal $\mathbf{n}_j$ and thus is the main smoothing force. Finally, the Gaussian weight function g on the distance $|\Pi_j(\mathbf{p}_i) - \mathbf{p}_i|$ penalizes fairing across sharp features. The normalization term k_i is simply the sum of weights in Φ. The variance defining the Gaussian f determines the spatial size of features that are smoothed while g affects the coarseness of noise.

Normal propagation

Generally, normal estimates computed by local properties of point cloud data, such as the covariance analysis we've outlined, don't provide a consistent orientation of the normal with respect to pointing inward or outward. Thus neighboring points might have similar normal directions but be oriented in opposite ways. Propagating a consistent normal orientation is tricky. Similar to many other approaches, we use a minimum spanning tree normal-propagation algorithm to solve this problem. Note, however, that this minimum spanning tree isn't based on the Euclidean distance between two neighboring points but on the angular deviation of their normal estimates.[11]

The normal propagation process is basically a breadth-first traversal of a graph $G(V, E)$ with vertices $V = \{\mathbf{p}_i\}$ and edges $E = \cup_{Ki}\{\mathbf{p}_i, \mathbf{p}_j) | \mathbf{p}_j \in K_i\}$. The process is initiated at a seed point $\mathbf{p}_s$, and a priority queue Q is initialized with its neighbors $\mathbf{p}_j \in K_s$ and corresponding priorities $q_j = |\mathbf{n}_s \times \mathbf{n}_i|$. We assume the seed point to be the first element of the oriented set while all others are considered nonoriented. The propagation then iteratively proceeds as follows. At each step, if Q isn't empty, the process removes the highest priority point $\mathbf{p}_j$ from Q. If $\mathbf{p}_j$ has already been oriented, we continue with the next element in Q. If $\mathbf{p}_j$ isn't oriented, then the process analyzes its priority q_j, the measure of smallest angular deviation to a previously oriented point. If q_j is negative, the normal $\mathbf{n}_j$ is inverted. Additionally, the process adds the k-nearest neighbors K_j to the priority queue Q.

Multiresolution modeling

In our proposed processing pipeline, the last task is to generate a multiresolution representation. Although this step is optional, it's important for real-time rendering of large point sets. At this stage, the points $\mathbf{p}_1, \ldots, \mathbf{p}_n$ now all have oriented normals and well-defined elliptical extents to cover the object as Figure 7 shows. To handle multiresolution representation, we chose a point-octree hierarchy as Figure 8 shows.

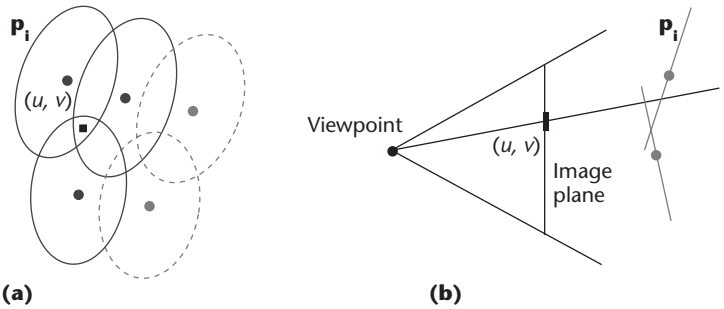
To generate this point-octree hierarchy, we followed the steps we have defined elsewhere.[9] In a depth-first traversal, we recursively split the point set with respect to its center of mass. Although the leaf nodes of this octree represent the initial input point set, the inner nodes represent aggregates of all points within these subtrees. Therefore, for each inner octree node r and the point set S_r it represents, we must find the aggregate normal and ellipse information. The normal $\mathbf{n}_r$ is set to be the average over all normals $\mathbf{n}_j$ of points $\mathbf{p}_j \in S_r$. We find the elliptical extent of r via the transformation invariant covariance. We can easily compute the generic homogeneous covariance $\overline{\boldsymbol{M}}_r$ as $\overline{\boldsymbol{M}}_r = \sum_{p_j \in s_r} \boldsymbol{C}_j$ without any recomputation of outer vector products. We then analogously compute the ellipse axis directions and aspect ratio as described earlier for the k-nearest neighbors.

Rendering

The major challenge in rendering a set of point splats—points with color, and normal and elliptical spatial extent—is to achieve a smooth and continuous interpolation between discrete points projected on screen. For rendering captured and processed point clouds, we use the approach we've explained elsewhere[12] and briefly outline the point blending and splatting techniques here.

Point blending

We achieve smooth and continuous interpolation between projected point splats on screen by a weighted blending of their (illuminated) colors. We express the color of a pixel (u, v) as $\mathbf{c}(u, v) = \sum \Psi_i(u, v) \cdot \mathbf{c}_i$, which is a weighted combination of the colors of the elliptical point splats $\boldsymbol{p}_i$ overlapping the pixel (u, v) in screen space, as Figure 9 shows. We define the blending



9 (a) Elliptical point splats p_i overlapping pixel (u, v) in screen space (dashed ones do not contribute). (b) Side view of overlapping point splats p_i projected on pixel (u, v).

weights Ψ_i as functions over the point splat ellipses in object space. We implement the blending by representing the blending functions Ψ_i as α-textures mapped onto appropriately oriented and scaled unit-sized triangles. Thus the Ψ_i parameterization, with respect to its on-screen projection, is given implicitly by texture mapping the projected triangle representing the point splat $\boldsymbol{p}_i$.

Furthermore, the irregular arrangement of elliptical splats covering the object means that the weighted blending is separated into the weighted fraction $\mathbf{c}(u, v) = \sum\Psi_i(u, v) \cdot \mathbf{c}_i / \sum\Psi_i(u, v)$, which allows using one simple and rotationally symmetric blending kernel Ψ for all points. Therefore, we achieve an intermediate per-pixel weighted blending by rendering the visible points $\mathbf{p}_i$ as triangles (centered on $\mathbf{p}_i$ and scaled with respect to the ellipse size and aspect ratio) with α-texture map corresponding to the blending kernel Ψ.

Splatting

The splatting process of on-screen projecting and blending point samples encompasses the following major steps:

1. View-dependent level-of-detail (LOD) selection of point samples.
2. Visibility splatting and blending of point splats.
3. Correction of intermediate blending result.

Step one is performed only if we want a point-based multiresolution representation. In this case, the LOD selection takes several view-dependent decision criteria into account such as view-frustum culling, back-face culling, and screen projection tolerance. These heuristics allow efficient back-tracking during a recursive top-down traversal of the point-octree hierarchy and have been explained in detail elsewhere.[9] In particular, the LOD traversal stops when the aggregate elliptical splat of an inner octree node is projected on screen to an area of less than a user-given threshold τ. This lets us adjust the rendering quality in image space.

Step two uses hardware acceleration to efficiently render and scan convert points as α-textured polygons. The α-texture represents the blending kernel Ψ, and we achieve the elliptical splat shape by appropriately orienting and scaling a unit-sized triangle. We use an ε-z-buffer technique to determine visibility and blend multiple points $\mathbf{p}_i$ overlapping a pixel (u, v), as Figure 9 shows.[13] This modified z-buffer visibility test allows accumulation and blending of color information from multiple point splats (α-textured triangles) overlapping a given pixel within a small depth-range ε. After this stage, the rendered image contains per-pixel-correct, proportionally blended colors $\mathbf{c}'(u, v) = \sum\Psi_i(u, v) \cdot \mathbf{c}_i$.

However, at this point the accumulated per-pixel blending weights generally don't sum up to unity, so our last step is to perform a per-pixel postprocess normalization. We obtain the final color $\mathbf{c}(u, v) = \mathbf{c}'(u, v) / \sum\Psi_i(u, v)$ by dividing the intermediate color by the accumulated per-pixel blending weights. We arrive at this final color by continually accumulating $\sum\Psi_i(u, v)$ in the α-channel of pixel (u, v) and finally dividing the intermediate color (the RGB values) by this α-value.

Results

We've performed several 3D-object-capturing experiments with the proposed system and algorithms we've described. The testing equipment consists of a 1.5-GHz Pentium 4 PC running Windows XP. As Table 1 shows, all test objects were processed and fully reconstructed in a matter of minutes. This doesn't include feature tracking, which adds about one second per input frame. The table data was derived from the monster model, a concrete statue of an elf; the ku model, a wood carved Hawaiian god; and the stegosaur model, a plastic model of a dinosaur. Table 1 reports the following statistics for the models:

- number of total frames in the captured video sequence,
- number of selected keyframes for tracking and volume carving,
- mean 2D reprojection pixel error μ of tracked features (both in x and y),

Table 1. Experimental data for data sets monster, ku, and stego.

Model	Frames	keyframes	μ	σ	t1 (sec)	t2 (sec)	t3 (sec)	Points
Monster	16	16	0.509 0.663	0.531 0.519	3.98	65.95	110.06	603,795
Ku	339	55	0.739 0.887	0.745 0.899	11.07	116.03	116.04	468,275
Stego	258	77	0.226 0.286	0.233 0.254	11.23	365.48	221.23	586,188

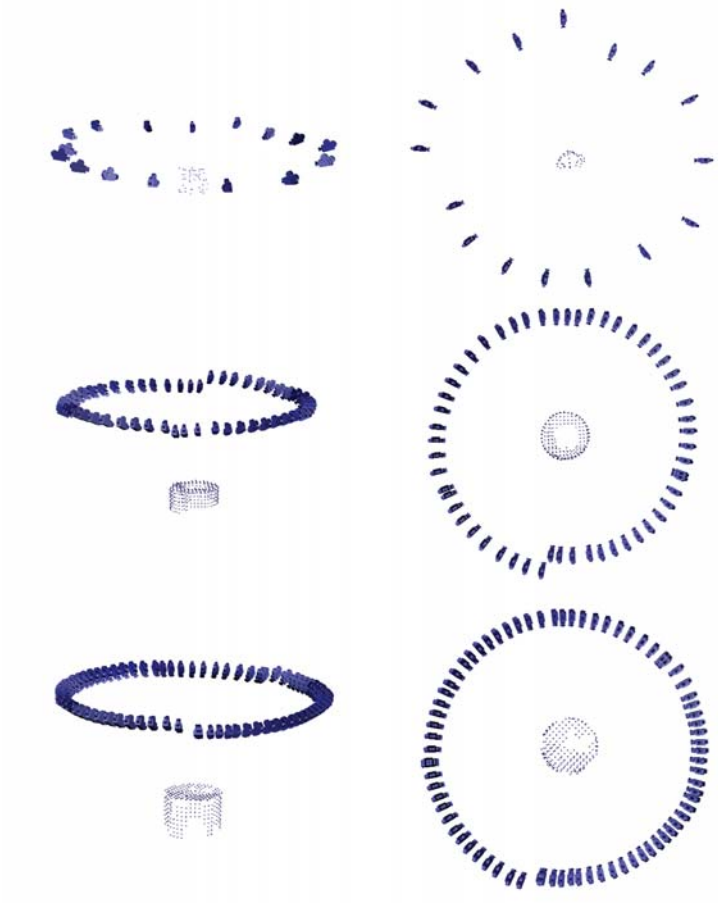
- standard deviation σ of 2D reprojection error of tracked features (both in x and y),
- time of camera calibration (t1),
- time of volumetric reconstruction (t2),
- time of point processing (t3), and
- number of reconstructed 3D points.

The monster data set didn't use fiducials on the turntable for feature tracking but used simple color fiducials on the object. This shows that the proposed system can perform tracking and calibration without artificial features if the object has a sufficiently clear texture pattern. Also, the monster data was captured not with a low-resolution video sequence but with a few high-resolution digital still images.

We performed background subtraction and feature selection by means of a green chroma-key backdrop. This limits the capturing to models not exhibiting a similar hue; however, different chroma-key backgrounds let us capture objects of all colors and shades. Also, the chroma-key background color sometimes exhibits noticeable color bleeding.

Figure 10 illustrates the result of the feature tracking and calibration phase as reconstructed camera positions relative to the captured objects. The camera's exhibited drift or misalignment is due to the variable focal length allowed in our calibration system, which compensates for the observed variation in distance to the object. The top row in the figure, from the monster data set, uses a small number of frames from a digital still image camera.

Figure 11 shows the reconstructed point models along with similar poses from the real video and image sequences. As these examples illustrate, the volume carving technique can reconstruct the visual hull faithfully but has limitations in discovering cavities. The lower resolution (640×480) of video images makes it difficult to reconstruct thin features of objects. The stegosaur model exhibits such limitations, particularly in the head and tail regions. The use of high-resolution digital still images ($1{,}024 \times 1{,}024$) improves both the resolution and



10 Reconstructed camera positions for the monster, ku, and stegosaur data sets.

11 Real frames from the captured video and image sequences and renderings of the reconstructed point models.

quality of the reconstructed points, as shown for the monster data set.

Conclusions

Although constructing good computer models of objects from image sequences is difficult, our proposed system SPOC and its algorithms have delivered good-quality 3D models reconstructed from largely unrestricted image sequences. In practice, the algorithms demonstrated high efficiency in working with real-world image sequences and in processing unorganized point data sets.

A good feature-tracking algorithm provides a solid base for a robust camera calibration phase. The calibration phase is one of the most important components of a passive 3D capturing system. Our calibration algorithm for long image sequences is fast and extremely robust and generic, that is, it doesn't require any particular restrictions on camera motion. Moreover, self-calibration offers good stability and versatility because it doesn't require an offline system calibration process.

Our experiments demonstrate the robustness of our calibration. We had no problems registering up to 100 camera locations and images, and the reconstructed models don't exhibit any distortions or geometry misalignments. In future work on calibration, we'll extend this approach to take further advantage of simple constraints such as fixed camera position and object rotation to achieve even better accuracy.

The 3D reconstruction phase of our capturing system inherits the general problems associated with basic volume carving. In particular, the spatial resolution of space occupancy is highly dependent on input-image resolution. We've observed that the limited digital video camera image resolution has an adverse effect on reconstructing high-resolution models. Low-resolution input images, in particular, produce strong aliasing artifacts when we significantly increase the 3D spatial resolution. This effect isn't unexpected, because we're performing a supersampling in that case. To increase the quality of carving methods in this context, we propose to introduce antialiasing techniques in future work to smooth out artifacts from supersampling.

Another effect we can observe from the experiments is that the computation cost increases with the number of input images and the final number of points. This is mainly caused by the color estimation procedure, because we use a larger number of images in the color average for a given point.

Determining the necessary display attributes for rendering requires that we process the raw point data from the 3D capturing stage very efficiently, because the raw point data sets can be enormous. Our algorithms for determining the per-point normal orientation and elliptical splat extent are likewise efficient. Despite being a noniterative operator, the fairing process takes by far the most time to complete of all point-processing operations. Consequently, further investigations are needed to reduce the time cost of this process. Also, we noticed that one of the most difficult tasks is consistently orienting normals such that all are pointing outward. In fact, the most commonly used approach, which we followed, fails in a few cases. To solve these situations, we fall back to a rough normal estimate from the carving phase, which is used to define the general outward pointing direction.

The major challenge in rendering point-based models is to achieve a smooth interpolation of surface attributes and illumination between points irregularly distributed both in 3D object-space as well as 2D image-space. This interpolation is basically a rasterization and shading problem. Our point-rendering algorithm computes a continuous interpolation between points as a weighted sum of overlapping point splats. The algorithm is accelerated using view-dependent LOD selection as well as hardware α-blending and texturing. Future work will mainly address performance improvements in visibility splatting, which currently requires a two-pass algorithm. ■

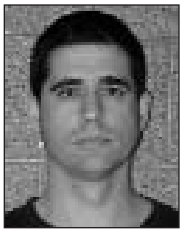
Acknowledgments

We thank the California-Catalunya Program for Engineering Innovation for financially supporting this work. We also thank Nvidia for equipment donation and the Image-Based Modeling and Rendering Lab at UC Irvine where the camera calibration code was developed.

References

1. J. Shi and C. Tomasi, "Good Features to Track," *Proc. IEEE Computer Soc. Conf. Computer Vision and Pattern Recognition* (CVPR 94), IEEE CS Press, 1994, pp. 593-600.
2. M. Sainz, A. Susin, and N. Bagherzadeh, "Camera Calibration of Long Image Sequences with the Presence of Occlusions," *Proc. Int'l Conf. Image Processing* (ICIP 03), IEEE Press, 2003, pp. 317-320.
3. R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge Univ. Press, 2000.
4. M. Sainz, N. Bagherzadeh, and A. Susin, "Carving 3D Models from Uncalibrated Views," *Proc. IASTED Int'l Conf. Computer Graphics and Imaging* (CGIM 02), ACTA Press, 2002, pp. 144-149.
5. A. Laurentini, "The Visual Hull Concept for Silhouette Based Image Understanding," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 16, no. 2, Feb. 1994, pp. 150-162.
6. W. Matusik et al., "Image-Based 3D Photography Using Opacity Hulls," *Proc. ACM Siggraph 2002, ACM Transaction on Graphics*, ACM Press, 2002, pp. 427-437.
7. N.J. Mitra and A. Nguyen, "Estimating Surface Normals in Noisy Point Cloud Data," *Symp. Computational Geometry*, ACM Press, 2003, pp. 322-328.
8. M. Dickerson, C.A. Duncan, and M.T. Goodrich, "K-D Trees are Better when Cut on the Longest Side," *Proc. European Symp. Algorithms* (ESA 00), 2000, pp. 179-190.
9. R. Pajarola, "Efficient Level-of-Details for Point Based Rendering," *Proc. IASTED Int'l Conf. Computer Graphics and Imaging* (CGIM 03), ACTA Press, 2003.
10. T.R. Jones, F. Durand, and M. Desbrun, "Non-Iterative, Feature-Preserving Mesh Smoothing," *Proc. ACM Siggraph,* ACM Press, 2003, pp. 943-949.
11. H. Hoppe et al., "Surface Reconstruction from Unorganized Points," *Proc. ACM Siggraph*, ACM Press, 1992, pp. 71-78.
12. R. Pajarola, M. Sainz, and P. Guidotti, "Object-Space Point Blending and Splatting," *ACM Siggraph Sketches & Applications Catalogue*, ACM Press, 2003.

13. J.P. Grossman and W.J. Dally, "Point Sample Rendering," *Proc. Eurographics Rendering Workshop 98*, Eurographics, 1998, pp. 181-192.



Miguel Sainz *is a postdoctoral student at the School of Information and Computer Science at the University of California, Irvine. His research interests include image-based modeling and rendering, 3D model reconstruction, and real-time 3D graphics. Sainz has a PhD in electrical and computer engineering from UCI. He is a member of the IEEE Computer Society and the ACM.*

Renato Pajarola *is an assistant professor in the Computer Science Department at UCI. His research interests include interactive 3D graphics, scientific visualization, and interactive 3D multimedia. Pajarola has a Dr. sc. techn. from ETH Zurich. He is a member of the IEEE Computer Society and the ACM.*

Albert Mercade *is a master's student in the Department of Electrical Engineering and Computer Science at UCI. His research interests include computer animation, computer graphics, image-based modeling, and geographic information systems. Mercade has a degree in mathematics from the Technical University of Catalonia, Spain.*

Toni Susin *is an associate professor in the Applied Mathematics Department at the Technical University of Catalonia in Spain. His research interests include image-based modeling and rendering, physically based animation, and medical applications. Susin has a PhD in celestial mechanics and dynamical systems from the University of Barcelona. He is an IEEE and Eurographics member.*

Readers may contact Miguel Sainz at 414A Computer Science; Univ. of Calif., Irvine, CA 92697; msainz@acm.org.