The Video Mesh: A Data Structure for Image-based Three-dimensional Video Editing

Jiawen Chen
MIT CSAIL

Sylvain Paris
Adobe Systems, Inc.

Jue Wang
Adobe Systems, Inc.

Wojciech Matusik
MIT CSAIL*

Michael Cohen
Microsoft Research

Frédo Durand
MIT CSAIL

Abstract

This paper introduces the video mesh*, a data structure for representing video as 2.5D "paper cutouts." The video mesh allows interactive editing of moving objects and modeling of depth, which enables 3D effects and post-exposure camera control. The video mesh sparsely encodes optical flow as well as depth, and handles occlusion using local layering and alpha mattes. Motion is described by a sparse set of points tracked over time. Each point also stores a depth value. The video mesh is a triangulation over this point set and per-pixel information is obtained by interpolation. The user rotoscopes occluding contours and we introduce an algorithm to cut the video mesh along them. Object boundaries are refined with per-pixel alpha values. The video mesh is at its core a set of texture mapped triangles, we leverage graphics hardware to enable interactive editing and rendering of a variety of effects. We demonstrate the effectiveness of our representation with special effects such as 3D viewpoint changes, object insertion, depth-of-field manipulation, and 2D to 3D video conversion.*

1. Introduction

We introduce the *video mesh*, a new representation that encodes the motion, layering, and 3D structure of a video sequence in a unified data structure. The video mesh can be viewed as a 2.5D "paper cutout" model of the world. For each frame of a video sequence, the video mesh is composed of a triangle mesh together with texture and alpha (transparency). Depth information is encoded with a per-vertex z coordinate, while motion is handled by linking vertices in time (for example, based on feature tracking). The mesh can be cut along occlusion boundaries and alpha mattes enable the fine treatment of partial occlusion. It supports a more general model of visibility than traditional layer-based methods [37] and can handle self-occlusions within an object such as the actor arm's in front of his body in our companion video. The per-vertex storage of depth and the rich occlusion representation make it possible to extend image-based modeling into the time dimension. Finally, the video mesh is based on texture-mapped triangles to enable fast processing on graphics hardware.

We leverage a number of existing computational photography techniques to provide user-assisted tools for the creation of a video mesh from an input video. Feature tracking provides motion information. Rotoscoping [2] and matting (e.g., [6, 18, 36]) enable fine handling of occlusion. A combination of structure-from-motion [11] and interactive image-based modeling [14, 23] permit a semi-automatic method for estimating depth. The video mesh enables a variety of video editing tasks such as changing the 3D viewpoint, occlusion-aware compositing, 3D object manipulation, depth-of-field manipulation, conversion of video from 2D to 3D, and relighting. This paper makes the following contributions:

- The video mesh, a sparse data structure for representing motion and depth in video that models the world as "paper cutouts."
- Algorithms for constructing video meshes and manipulating their topology. In particular, we introduce a robust mesh cutting algorithm that can handle arbitrarily complex occlusions in general video sequences.
- Video-based modeling tools for augmenting the structure of a video mesh, enabling a variety of novel special effects.

1.1. Related work

Mesh-based video processing Meshes have long been used in video processing for tracking, motion compensation, animation, and compression. The Particle Video system [26], uses a triangle mesh to regularize the motion of tracked features. Video compression algorithms [5] use meshes to sparsely encode motion. These methods are designed for motion compensation and handle visibility by resampling and remeshing along occlusion boundaries. They typically do not support self-occlusions. In contrast, our work focuses on using meshes as the central data structure used for editing. In order to handle arbitrary video se-

*The majority of the work was done while an employee at Adobe.

978-1-6128-4706-1/11/$26.00 ©2011 IEEE

quences, we need a general representation that can encode the complex occlusion relationships in a video. The video mesh decouples the complexity of visibility from that of the mesh by encoding it with a locally dense alpha map. It has the added benefit of handling partial coverage and sub-pixel effects.

Motion description Motion in video can be described by its dense optical flow, e.g. [13]. We have opted for a sparser treatment of motion based on feature tracking, e.g. [21, 28]. We find feature tracking more robust and easier to correct by a user. Feature tracking is also much cheaper to compute and per-vertex data is easier to process on GPUs.

Video representations The video mesh builds upon and extends layer-based video representations [1, 37], video cube segmentation [35], and video cutouts [20]. Commercial packages use stacks of layers to represent and composite objects. However, these layers remain flat and cannot handle self-occlusions within a layer such as when an actor's arm occludes his body. Similarly, although the video cube and video cutout systems provide a simple method for extracting objects in space-time, to handle self-occlusions, they must cut the object at an arbitrary location. The video mesh leverages user-assisted rotoscoping [2] and matting [6, 18, 36] to extract general scene components without arbitrary cuts.

Background collection and mosaicing can be used to create compound representations, *e.g.*, [15, 32]. Recently, Rav-Acha *et al*. [25] introduced Unwrap Mosaics to represent object texture and occlusions without 3D geometry. High accuracy is achieved through a sophisticated optimization scheme that runs for several hours. In comparison, the video mesh outputs coarse results with little precomputation and provides tools that let the user interactively refine the result. Unwrap Mosaics are also limited to objects with a disc topology whereas the video mesh handles more general scenes.

Image-based modeling and rendering We take advantage of existing image-based modeling techniques to specify depth information at vertices of the video mesh. In particular, we adapt a number of single-view modeling tools to video [14, 23, 39]. We are also inspired by the Video Trace technique [34] which uses video as an input to interactively model static objects. We show how structure-from-motion [11] can be applied selectively to sub-parts of the video to handle piecewise-rigid motion which are common with everyday objects. We also present a simple method that propagates depth constraints in space.

Stereo video Recent multi-view algorithms are able to automatically recover depth in complex scenes from video sequences [27]. However, these techniques require camera motion and may have difficulties with non-Lambertian materials and moving objects. Zhang *et al*. demonstrate how to perform a number of video special effects [38] using depth maps estimated using multi-view stereo. Recent work by Guttman *et al*. [10] provides an interface to recovering video depth maps from user scribbles. The video mesh is complementary to these methods. We can use depth maps to initialize the 3D geometry and our modeling tools to address challenging cases such as scenes with moving objects.

By representing the scene as 2.5D paper cutouts, video meshes enable the conversion of video into stereoscopic 3D by re-rendering the mesh from two viewpoints. A number of commercial packages are available for processing content filmed in with a stereo setup [24, 33]. These products extend traditional digital post-processing to handle 3D video with features such as correcting small misalignments in the stereo rig, disparity map estimation, and inpainting. The video mesh representation would enable a broader range of effects while relying mostly on the same user input for its construction. Recent work by Koppal *et al*. [17], describes a pre-visualization system for 3D movies that helps cinematographers plan their final shot from draft footage. In comparison, our approach aims to edit the video directly.

2. The video mesh data structure

We begin by describing the properties of the video mesh data structure and illustrate how it represents motion and depth in the simple case of a smoothly moving scene with no occlusions. In this simplest form, it is similar to morphing techniques that rely on triangular meshes and texture mapping [9]. We then augment the structure to handle occlusions, and in particular self-occlusions that cannot be represented by layers without artificial cuts. Our general occlusion representation simplifies a number of editing tasks. For efficient image data storage and management, we describe a tile-based representation for texture and transparency. Finally, we show how a video mesh is rendered.

2.1. A triangular mesh

Vertices The video mesh encodes depth and motion information at a sparse set of vertices, which are typically obtained from feature tracking. Vertices are linked through time to form tracks. A vertex stores its position in the original video, which is used to reference textures that store the pixel values and alpha. The current position of a vertex can be modified for editing purposes (e.g. to perform motion magnification [21]), and we store it in a separate field. Vertices also have a continuous depth value which can be edited using a number of tools, described in Section 3.2. Depth information is encoded with respect to a camera matrix that is specified per frame.

Faces We use a Delaunay triangulation over each frame to define the faces of the video mesh. Each triangle is texture-mapped using the pixel values from the original video, with texture coordinates defined by the original position of its

vertices. The textures can be edited to enable various video painting and compositing effects. Each face references a list of texture tiles to enable the treatment of multiple layers.

The triangulations of consecutive frames are mostly independent. While it is desirable that the topology be as similar as possible between frames to generate a continuous motion field, this is not a strict requirement. We only require vertices, not faces, to be linked in time. The user can force edges to appear in the triangulation by adding *line constraints*. For instance, we can ensure that a building is accurately represented by the video mesh by aligning the triangulation with its contours.

Motion For illustration, consider a simple manipulation such as motion magnification [21]. One starts by tracking features over time. For this example, we assume that all tracks last the entire video sequence and that there is no occlusion. Each frame is then triangulated to create faces. The velocity of a vertex can be accessed by querying its successor and predecessor and taking the difference. A simple scaling of displacement [21] yields the new position of each vertex. The final image for a given frame is obtained by rendering each triangle with the vertices at the new location but with texture coordinates at the original position, indexing the original frames. This is essentially equivalent to triangulation-based morphing [9].

2.2. Occlusion

Real-world scenes have occlusions, which are always the most challenging aspect of motion treatment. Furthermore, vertex tracks can appear or disappear over time because of, for instance, occlusion or loss of contrast. The video mesh handles these cases by introducing *virtual vertices* and duplicating triangles to store information for both foreground and background parts.

Consider first the case of vertices that appear or disappear over time. Since we rely on the predecessor and successor to extract motion information, we introduce *temporal* virtual vertices at both ends of a vertex track. Like normal vertices, they store a position, which is usually extrapolated from adjacent frames but can also be fine-tuned by the user.

Real scenes also contain spatial occlusion boundaries. In mesh-based interpolation approaches, a triangle that overlaps two scene objects with different motions yields artifacts when motion is interpolated. While these artifacts can be reduced by refining the triangulation to closely follow edges, *e.g.*, [5], this solution can significantly increase geometric complexity and does not handle soft boundaries. Instead, we take an approach inspired by work in mesh-based physical simulation [22]. At occlusion boundaries, where a triangle partially overlaps both foreground and background layers, we *duplicate* the face into foreground and background copies, and add *spatial* virtual vertices to complete the topology. To resolve per-pixel coverage, we compute a

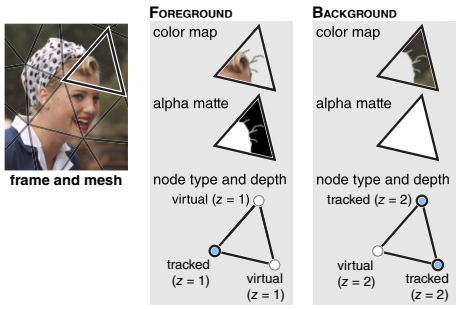


Figure 1. Occlusion boundaries are handled by duplicating faces. Each boundary triangle stores a matte and color map. Duplicated vertices are either *tracked*, *i.e.*, they follow scene points, or *virtual* if their position is inferred from their neighbors.

local alpha matte to disambiguate the texture (see Figure 1). Similar to temporal virtual vertices, their spatial counterparts store position information that is extrapolated from their neighbors. We extrapolate a motion vector at these points and create temporal virtual vertices in the adjacent past and future frames to represent this motion. Topologically, the foreground and background copies of the video mesh are locally disconnected: information cannot directly propagate across the boundary.

When an occlusion boundary does not form a closed loop, it ends at a singularity called a *cusp*. The triangle at the cusp is duplicated like any other boundary triangle and the alpha handles fine-scale occlusion. We describe the topological construction of cuts and cusps in Section 3.1.

The notion of occlusion in the video mesh is purely local and enables self-occlusion within a layer, just like how a 3D polygonal mesh can exhibit self-occlusion. Occlusion boundaries do not need to form closed contours.

2.3. Tile-based texture storage

At occlusion boundaries, the video mesh is composed of several overlapping triangles and a position in the image plane can be assigned several color and depth values, typically one for the foreground and one for the background. While simple solutions such as the replication of the entire frame are possible, we present a tile-based approach that strikes a balance between storage overhead and flexibility.

Replicating the entire video frame for each layer would be wasteful since few faces are duplicated and in practice, we would run out of memory for all but the shortest video sequences. Another possibility would be generic mesh parameterization [12], but the generated atlas would likely introduce distortions since these methods have no knowledge of the characteristics of the video mesh, such as its rectangular domain and preferred viewpoint.

Tiled texture We describe a tile-based storage scheme which trades off memory for rendering efficiency—in particular, it does not require any mesh reparameterization.

The image plane is divided into large blocks (*e.g.*, 128×128). Each block contains a list of texture tiles that form a stack. Each face is assigned its natural texture coordinates; that is, with (u, v) coordinates equal to the (x, y) image position in the input video. If there is already data stored at this location (for instance, when adding a foreground triangle and its background copy already occupies the space in the tile), we move up in the stack until we find a tile with free space. If a face spans multiple blocks, we push onto each stack using the same strategy: a new tile is created within a stack if there is no space in the existing tiles.

To guarantee correct texture filtering, each face is allocated a one-pixel-wide margin so that bilinear filtering can be used. If a face is stored next to its neighbor, then this margin is already present. Boundary pixels are only necessary when two adjacent faces are stored in different tiles. Finally, tile borders overlap by two-pixels in screen space to ensure correct bilinear filtering for faces that span multiple tiles.

The advantages of a tile-based approach is that overlapping faces require only a new tile instead of duplicating the entire frame. Similarly, local modifications of the video mesh such as adding a new boundary impact only a few tiles, not the whole texture. Finally, the use of canonical coordinates also enable data to be stored without distortion relative to the input video.

2.4. Rendering

The video mesh is, at its core, a collection of texture-mapped triangles and is easy to render using modern graphics hardware. We handle transparency by rendering the scene back-to-front using alpha blending, which is sufficient when faces do not intersect. We handle faces that span several tiles with a dedicated shader that renders them once per tile, clipping the face at the tile boundary. To achieve interactive rendering performance, tiles are cached in texture memory as large atlases (*e.g.*, 4096×4096), with tiles stored as subregions. Caching also enables efficient rendering when we access data across multiple frames, such as when we perform space-time copy-paste operations. Finally, when the user is idle, we prefetch nearby frames in the background into the cache to enable playback after seeking to a random frame.

3. Video mesh operations

The video mesh supports a number of creation and editing operators. This section presents the operations common to most applications, while we defer application-specific algorithms to Section 4.

3.1. Cutting the mesh along occlusions

The video mesh data structure supports a rich model of occlusion as well as interactive creation and manipulation. For this, we need the ability to cut the mesh along user-provided occlusion boundaries. We use splines to specify occlusions [2], and once cut, the boundary can be refined using image matting [6, 18, 36]. In this section, we focus on the topological cutting operation of a video mesh given a set of splines. A boundary spline has the following properties:
1. It specifies an occlusion boundary and intersects another spline only at T-junctions.
2. It is **directed**, which *locally* separates the image plane into foreground and background.
3. It can be **open** or **closed**. A closed spline forms a loop that defines an object detached from its background. An open spline indicates that two layers merge at an endpoint called a *cusp*.

Ordering constraints In order to create a video mesh whose topology reflects the occlusion relations in the scene, the initial flat mesh is cut front-to-back. We organize the boundary splines into a directed graph where nodes correspond to splines and a directed edge between two splines indicates that one is in front of another. We need this ordering only at T-junctions, where a spline a ends in contact with another spline b. If a terminates on the foreground side of b, we add an edge $a \rightarrow b$, otherwise we add $b \rightarrow a$. Since the splines represent the occlusions in the underlying scene geometry, the graph is guaranteed to be acyclic. Hence, a topological sort on the graph produces a front-to-back partial ordering from which we can create layers in order of increasing depth. For each spline, we walk from one end to the other and cut each crossed face according to how it is traversed by the spline. If a spline forms a T-junction with itself, we start with the open end; and if the two ends form T-junctions, we start at the middle of the spline (Fig. 5). This ensures that self T-junctions are processed top-down.

Four configurations To cut a mesh along splines, we distinguish the four possible configurations:
1. If a face is fully cut by a spline, that is, the spline does not end inside, we duplicate the face into foreground and background copies. Foreground vertices on the background side of the spline are declared virtual. We attach the foreground face to the uncut and previously duplicated faces on the foreground side. We do the same for the background copy (Fig. 2).
2. If a face contains a T-junction, we first cut the mesh using the spline in front as in case 1. Then we process the back spline in the same way, but ensure that at the T-junction, we duplicate the background copy (Fig. 3). Since T-junctions are formed by an object in front of an occlusion boundary, the back spline is always on the background side and this strategy ensures that the topology is compatible with the underlying scene.

3. If a face is cut by a cusp (*i.e.*, by a spline ending inside it), we cut the face like in case 1. However, the vertex opposite the cut edge is not duplicated; instead, it is shared between the two copies (Fig. 4).
4. In all the other cases where the face is cut by two splines that do not form a T-junction or by more than two splines, we subdivide the face until we reach one of the three cases above.

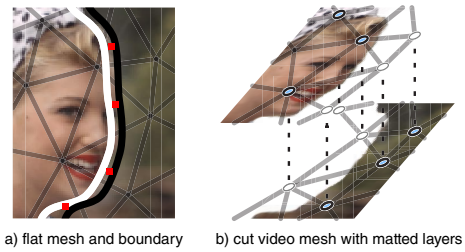


Figure 2. Cutting the mesh with a boundary spline. The cut faces are duplicated. The foreground copies are attached to the adjacent foreground copies and uncut faces. A similar rule applies to the background copies. Blue vertices are *real* (tracked), white vertices are *virtual*.

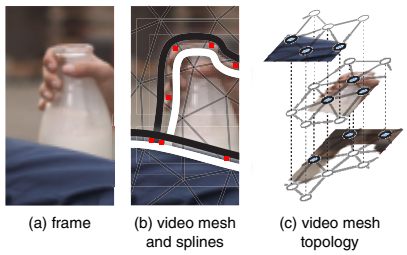


Figure 3. Cutting the mesh with two splines forming a T-junction. We first cut according to the non-ending spline, then according to the ending spline.

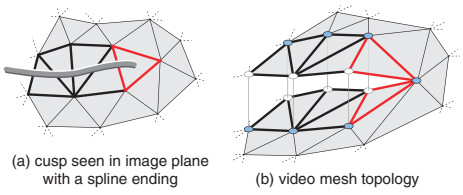


Figure 4. Cutting the mesh with a cusp. This case is similar to the normal cut (Fig. 2) except that the vertex opposite to the cut edge is shared between the two copies.

Motion estimation Cutting the mesh generates spatial virtual vertices without successors or predecessors in time. We estimate their motion by diffusion from their neighbors. For each triangle with two tracked vertices and a virtual vertex, we compute the translation, rotation, and scaling of the edges with the two tracked vertices. We apply the same transformation to the virtual vertex to obtain its motion estimate. If the motion of a virtual vertex can be evaluated from several faces, we find a least-squares approximation to its motion vector. We use this motion vector to create temporal virtual vertices in the previous and next frame. This process is iterated as a breadth-first search until the motion of all virtual vertices are computed.

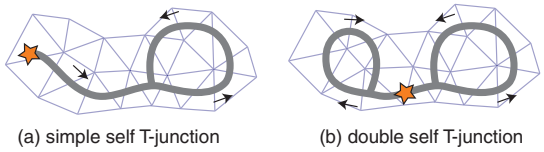


Figure 5. If a spline forms a T-junction with itself (a), we start from the open end (shown with a star) and process the faces in order toward the T-junction. If a spline forms two T-junctions with itself (b), we start in between the two T-junctions and process the faces bidirectionally.

Boundary propagation Once we have motion estimates for all spatial virtual vertices in a frame, we can use the video mesh to advect data. In particular, we can advect the control points of the boundary spline to the next (or previous) frame. Hence, once the user specifies the occlusion boundaries at a single keyframe, as long as the topology of occlusion boundaries does not change, we have enough information to build a video mesh over all frames. We detect topology changes when two splines cross and ask the user to adjust the splines accordingly. In practice, the user needs to edit 5 to 10% of the frames, which is comparable to the technique of Agarwala *et al*. [2].

3.2. Depth estimation

After cutting the video mesh, it is already possible to infer a pseudo-depth value based on the foreground/background labeling of the splines. However, for a number of video processing tasks, continuous depth values enable more sophisticated effects. As a proof of concept, we provide simple depth-modeling tools that work well for two common scenarios. For more challenging scenes, the video mesh can support the dense depth maps generated from more advanced techniques such as multi-view stereo.

Static camera: image-based modeling For scenes that feature a static camera with moving foreground objects, we provide tools inspired from the still photograph case [14,23] to model a coarse geometric model of the background. The *ground tool* lets the user define the ground plane from the horizon line. The *vertical object tool* enables the creation of vertical walls and standing characters by indicating their contact point with the ground. The *focal length tool* retrieves the camera field of view from two parallel or orthogonal lines on the ground. This proxy geometry is sufficient to handle complex architectural scenes as demonstrated in the supplemental video.

Moving camera: user-assisted structure-from-motion For scenes with a moving camera, we build on structure-from-motion [11] to simultaneously recover a camera path as well as the 3D position of scene points. In general, there might be several objects moving independently. The user can indicate rigid objects by selecting regions delineated by the splines. We recover their depth and motion independently using structure-from-motion and register them in a global coordinate system by aligning to the camera path which does not change. We let the user correct misalignments by specifying constraints, typically by pinning a vertex to a given position.

Even with a coarse video mesh, these tools allow a user to create a model that is reasonably close to the true 3D geometry. In addition, after recovering the camera path, adding vertices is easy by clicking on the same point in only 2 frames. The structure-from-motion solver recovers its 3D position by minimizing reprojection error over all the cameras.

3.3. Inpainting

We triangulate the geometry and inpaint the texture in hidden parts of the scene in order to render 3D effects such as changing the viewpoint without revealing holes.

Geometry For closed holes that typically occur when an object occludes the background, we list the mesh edges at the border of the hole and fill in the mesh using constrained Delaunay triangulation with the border edges as constraints.

When a boundary is occluded, which happens when an object partially occludes another, we observe the splines delineating the object. An occluded border generates two T-junctions which we detect. We add an edge between the corresponding triangles and use the same strategy as above with Delaunay triangulation.

Texture For large holes that are typical of missing static backgrounds, we use *background collection* [30]. After infilling the geometry of the hole, we use the motion defined by the mesh to search forward and backward in the video for unoccluded pixels. Background collection is effective when there is moderate camera or object motion and can significantly reduce the number of missing pixels. We fill the remaining pixels by isotropically diffusing data from the edge of the hole.

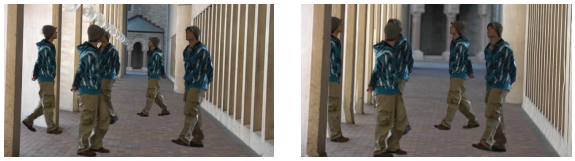
When the missing region is textured and temporally stable, such as on the shirt of an actor in Soda sequence of our video, we modify Efros and Leung texture synthesis [8] to search only in the same connected component as the hole within the same frame. This strategy ensures that only semantically similar patches are copied and works well for smoothly varying dynamic objects. Finally, for architectural scenes where textures are more regular and boundaries are straight lines (Figure 7), we proceed as Khan *et al*. [16] and mirror the neighboring data to fill in the missing regions. Although these tools are simple, they achieve satisfying results in our examples since the regions where they are applied are not the main focus of the scene. If more accuracy is needed, one can use dedicated mesh repair [19] and inpainting [3, 4] algorithms.

(a) original viewpoint (b) new viewpoint

Figure 6. **Left:** original frame. **Right:** camera moved forward and left toward the riverbank.

(a) focus on foreground (b) focus on background

Figure 7. Compositing and depth of field manipulation. We replicated the character from the original video, composited multiple copies with perspective, and added defocus blur.



(a) wide field of view, camera close to the subject (b) narrow field of view, camera far from the subject

Figure 8. Vertigo effect enabled by the 3D information in the video mesh. We zoom in and at the same time pull the camera back.

4. Results

We illustrate the use of the video mesh on a few practical applications. These examples exploit the video mesh's accurate scene topology and associated depth information to create a variety of 3D effects. The results are available in the companion video.

Depth of field manipulation We can apply effects that depend on depth such as enhancing a camera's depth of field. To approximate a large aperture camera with a shallow depth of field, we construct a video mesh with 3D information and render it from different viewpoints uniformly sampled over a synthetic aperture, keeping a single plane in focus. Since the new viewpoints may reveal holes, we use our inpainting operator to fill both the geometry and texture. For manipulating defocus blur, inpainting does not need to be accurate. This approach supports an arbitrary location

for the focal plane and an arbitrary aperture. In the Soda and Colonnade sequences, we demonstrate the *rack focus* effect which is commonly used in movies: the focus plane sweeps the scene to draw the viewer's attention to subjects at various distances (Fig. 7). This effect can be previewed in real time by sampling 128 points over the aperture. A high-quality version with 1024 samples renders at about 2 Hz.

Object insertion and manipulation The video mesh supports an intuitive copy-and-paste operation for object insertion and manipulation. The user delineates a target object with splines, which is cut out to form its own connected component. The object can then be replicated or moved anywhere in space and time by copying the corresponding faces and applying a transformation. The depth structure of the video mesh enables occlusions between the newly added objects and the existing scene while per-pixel transparency makes it possible to render antialiased edges. This is shown in the Colonnade sequence where the copied characters are occluded by the pillars and each other. The user can also specify that the new object should be in contact with the scene geometry. In this case, the depth of the object is automatically provided according to the location in the image. We further develop this idea by exploiting the motion description provided by the video mesh to ensure that the copied objects consistently move as the camera viewpoint changes. This feature is shown in the Copier sequence of the companion video. When we duplicate an animated object several times, we offset the copies in time to prevent unrealistically synchronized movements.

We also use transparency to render volumetric effects. In the Soda sequence, we insert a simulation of volumetric smoke. To approximate the proper attenuation and occlusion that depends on the geometry, we render 10 offset layers of 2D semi-transparent animated smoke.

Change of 3D viewpoint With our modeling tools (Sec. 3.2) we can generate proxy geometry that enables 3D viewpoint changes. We demonstrate this effect in the companion video and in Figure 6. In the Colonnade and Notre-Dame sequences, we can fly the camera through the scene even though the input viewpoint was static. In the Copier sequence, we apply a large modification to the camera path to get a better look at the copier glass. Compared to existing techniques such as Video Trace [34], the video mesh can handle moving scenes as shown with the copier. The scene geometry also allows for change of focal length, which in combination with change of position, enables the *vertigo effect*, a.k.a. *dolly zoom*, in which the focal length increases while the camera moves backward so that the object of interest keeps a constant size (Fig. 8).

Relighting and participating media We use the 3D geometry encoded in the video mesh for relighting. In the companion video, we transform the daylight Colonnade sequence into a night scene. We use the original pixel value as the diffuse material color, and let the user click to position light sources. We render the scene using raytracing to simulate shadows and volumetric fog.

Stereo 3D With a complete video mesh, we can output stereo 3D by rendering the video mesh twice from different viewpoints. We rendered a subset of the results described above in red/cyan anaglyphic stereo; the red channel contains the red channel from the "left eye" image, and the green and blue channels contain the green and blue channels from the "right eye" image. The results were rendered with both cameras parallel to the original view direction, displaced by half the average human interocular distance.

Performance and user effort We implemented our prototype in DirectX 10 and ran our experiments on an Intel Core i7 920 at 2.66 GHz with 8 MB of cache and a NVIDIA GeForce GTX 280 with 1 GB of video memory. At a resolution of 640×360, total memory usage varies between 1 GB to 4 GB depending on the sequence, which is typical for video editing applications. To handle long video sequences, we use a multi-level virtual memory hierarchy over the GPU, main memory, and disk with background prefetching to seamlessly access and edit the data. With the exception of offline raytracing for the fog simulation, and high-quality depth of field effects that require rendering each frame 1024 times, all editing and rendering operations are interactive (> 15 Hz). In our experiments, the level of user effort depends mostly on how easy it is to track motion and how much the scene topology varies in time. Most of the time was spent constructing the video mesh, with both point tracking and rotoscoping taking between 5-25 minutes each. Once constructed, creating the effects themselves were interactive, as can be seen in the supplemental video. We refer the reader to the supplemental material for a more detailed discussion on the workflow used to produce our examples.

5. Discussion

Although our approach gives users a flexible way of editing a large class of videos, it is not without limitations. The primary limitation stems from the fact that the video mesh is a coarse model of the scene: high-frequency motion, complex geometry, and thin features would be difficult to accurately represent without excessive tesselation. For instance, the video mesh has trouble representing a field of grass blowing in the wind, although we believe that other techniques would also have difficulties. For the same reason, the video mesh cannot represent finely detailed geometry such as a bas-relief on a wall. In this case, the bas-relief would appear as a texture on a smooth surface, which may be sufficient in a number of cases, but not if the bas-relief is the main object of interest. A natural extension would be to augment the video mesh with a displacement map to handle high-frequency geometry. Other possibilities for handling

complex geometry are to use an alternative representation, such as billboards, imposters, or consider a unified representation of geometry and matting [31]. To edit these representations, we would like to investigate more advanced interactive modeling techniques, in the spirit of those used to model architecture from photographs [7, 29]. Integrating these approaches into our system is a promising direction for future research.

Conclusion We have presented the video mesh, a data structure to represent video sequences and whose creation is assisted by the user. The required effort to build a video mesh is comparable to rotoscoping but the benefits are higher since the video mesh offers a rich model of occlusion and enables complex effects such as depth-aware compositing and relighting. Furthermore, the video mesh naturally exploits graphics hardware capabilities to provide interactive feedback to users. We believe that video meshes can be broadly used as a data structure for video editing.

References

[1] Adobe Systems, Inc. After Effects CS4, 2008.

[2] A. Agarwala, A. Hertzmann, D. H. Salesin, and S. M. Seitz. Keyframe-based tracking for rotoscoping and animation. *ACM Transactions on Graphics*, 23(3):584–591, 2004.

[3] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. PatchMatch: A randomized correspondence algorithm for structural image editing. *SIGGRAPH 2009*.

[4] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *SIGGRAPH 2000*.

[5] N. Cammas, S. Pateux, and L. Morin. Video coding using non-manifold mesh. In *Proceedings of the 13th European Signal Processing Conference*, 2005.

[6] Y.-Y. Chuang, A. Agarwala, B. Curless, D. Salesin, and R. Szeliski. Video matting of complex scenes. *ACM Transactions on Graphics*, 21(3):243–248, 2002.

[7] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs. In *SIGGRAPH 1996*.

[8] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *ICCV 1999*.

[9] J. Gomes, L. Darsa, B. Costa, and L. Velho. *Warping and morphing of graphical objects*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[10] M. Guttman, L. Wolf, and D. Cohen-Or. Semi-automatic stereo extraction from video footage. In *ICCV 2009*.

[11] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, June 2000.

[12] K. Hormann, B. Lévy, and A. Sheffer. Mesh parameterization: Theory and practice. In *ACM SIGGRAPH Course Notes*. ACM, 2007.

[13] B. K. P. Horn and B. G. Schunck. Determining optical flow. *Arti cial Intelligence*, 17(1-3):185–203, 1981.

[14] Y. Horry, K. Anjyo, and K. Arai. Tour into the picture: Using a spidery mesh interface to make animation from a single image. In *SIGGRAPH 1997*.

[15] M. Irani, P. Anandan, and S. Hsu. Mosaic based representations of video sequences and their applications. In *ICCV*, 1995.

[16] E. A. Khan, E. Reinhard, R. Fleming, and H. Buelthoff. Image-based material editing. *SIGGRAPH 2006*.

[17] S. Koppal, C. L. Zitnick, M. Cohen, S. B. Kang, B. Ressler, and A. Colburn. A viewer-centric editor for stereoscopic cinema. *IEEE CG&A*.

[18] A. Levin, D. Lischinski, and Y. Weiss. A Closed Form Solution to Natural Image Matting. *CVPR 2006*.

[19] B. Lévy. Dual domain extrapolation. *SIGGRAPH 2003*.

[20] Y. Li, J. Sun, and H.-Y. Shum. Video object cut and paste. *SIGGRAPH 2005*.

[21] C. Liu, A. Torralba, W. T. Freeman, F. Durand, and E. H. Adelson. Motion magnification. *SIGGRAPH 2005*.

[22] N. Molino, Z. Bao, and R. Fedkiw. A virtual node algorithm for changing mesh topology during simulation. *ACM Transactions on Graphics*, 23(3):385–392, 2004.

[23] B. M. Oh, M. Chen, J. Dorsey, and F. Durand. Image-based modeling and photo editing. In *SIGGRAPH 2001*.

[24] Quantel Ltd. Pablo. http://goo.gl/M7d4, 2010.

[25] A. Rav-Acha, P. Kohli, C. Rother, and A. Fitzgibbon. Unwrap mosaics: a new representation for video editing. *ACM Transactions on Graphics*, 27(3):17:1–17:11, 2008.

[26] P. Sand and S. Teller. Particle video: Long-range motion estimation using point trajectories. In *CVPR 2006*.

[27] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *CVPR 2006*.

[28] J. Shi and C. Tomasi. Good features to track. *CVPR 1994*.

[29] S. N. Sinha, D. Steedly, R. Szeliski, M. Agrawala, and M. Pollefeys. Interactive 3D architectural modeling from unordered photo collections. *SIGGRAPH Asia 2008*.

[30] R. Szeliski. Video mosaics for virtual environments. *IEEE CG&A 1996*.

[31] R. Szeliski and P. Golland. Stereo matching with transparency and matting. *IJCV 1999*.

[32] R. S. Szeliski. Video-based rendering. In *Vision, Modeling, and Visualization*, page 447, 2004.

[33] The Foundry Visionmongers Ltd. Ocula. http://www.thefoundry.co.uk/products/ocula/, 2009.

[34] A. van den Hengel, A. Dick, T. Thormählen, B. Ward, and P. H. S. Torr. VideoTrace: rapid interactive scene modelling from video. *SIGGRAPH 2007*.

[35] J. Wang, P. Bhat, R. A. Colburn, M. Agrawala, and M. F. Cohen. Interactive video cutout. *SIGGRAPH 2005*.

[36] J. Wang and M. Cohen. Optimized color sampling for robust matting. In *CVPR 2007*.

[37] J. Y. A. Wang and E. H. Adelson. Representing moving images with layers. *IEEE Trans. on Image Proc.*, 1994.

[38] G. Zhang, Z. Dong, J. Jia, L. Wan, T. Wong, and H. Bao. Refilming with depth-inferred videos. *IEEE Transactions on Visualization and Computer Graphics*, 15(5):828–840, 2009.

[39] L. Zhang, G. Dugas-Phocion, J.-S. Samson, and S. M. Seitz. Single view modeling of free-form scenes. In *CVPR 2001*.