Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2009-062 December 16, 2009

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

Jiawen Chen, Sylvain Paris, Jue Wang, Wojciech Matusik, Michael Cohen, and Fredo Durand

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

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

Jiawen Chen[1] Sylvain Paris[2] Jue Wang[2] Wojciech Matusik[2] Michael Cohen[3] Frédo Durand[1]

[1]Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology

[2]Adobe Systems, Inc.

[3]Microsoft Research

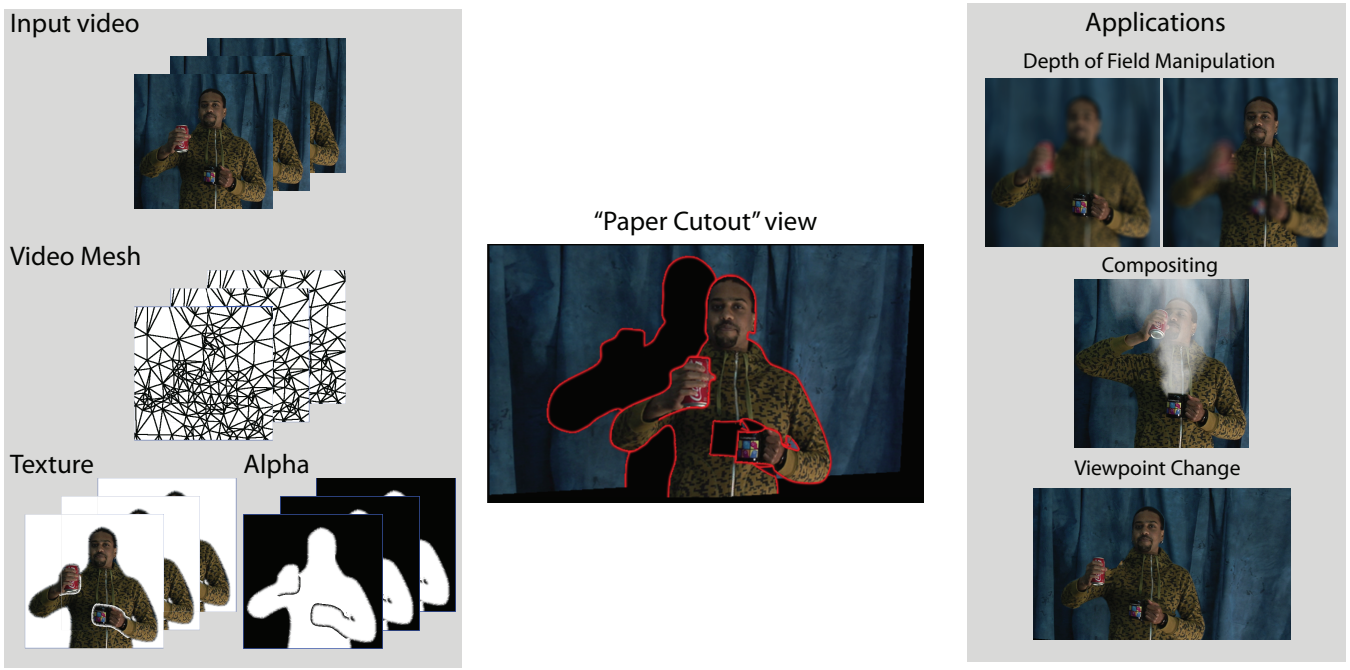


Figure 1: *The video mesh data structure represents the structural information in an input video as a set of deforming texture mapped triangles augmented with mattes. The mesh has a topology that resembles a "paper cutout". This representation enables a number of special effects applications such as depth of field manipulation, object insertion, and change of 3D viewpoint.*

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 a number of special effects including 3D viewpoint changes, object insertion, and depth-of-field manipulation.

Keywords: video editing, triangle mesh, compositing, feature tracking, optical flow

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 scene as shown in Figure 1. 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 treatment of visibility than traditional layer-based methods [Wang and Adelson 1994] and can handle self-occlusions within a layer such as the actor's hand in Figure 1. The per-vertex storage of depth and the rich occlusion representation makes 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 off-the-shelf technology when possible to provide user-assisted tools for the creation of a video mesh from an input video. Tracking software provides motion information. Rotoscoping [Agarwala et al. 2004] and matting (e.g., [Chuang et al. 2002; Levin et al. 2006; Wang and Cohen 2007]) enable fine handling of occlusion. A combination of structure-from-motion [Hartley and Zisserman 2000] and interactive image-based modeling [Horry

et al. 1997; Oh et al. 2001] permit depth specification. 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, 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 video edits such as 3D viewpoint changes, depth-of-field manipulation, propagation of 2D edits, and 3D object manipulation.

1.1 Related Work

Motion Motion in video can be described by its dense optical flow, e.g. [Horn and Schunck 1981]. We have opted for a sparser treatment of motion based on feature tracking, e.g. [Shi and Tomasi 1994; Liu et al. 2005]. 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 graphics hardware. Sand and Teller's particle video [2006] is another sparse representation of motion but it requires a costly dense optical flow calculation as a preprocess, while our focus is on fast manipulation. Furthermore, we seek to handle layering and depth.

Layered video representations The video mesh builds upon and extends layer-based video representations [Wang and Adelson 1994; Apple, Inc. 2007; Adobe Systems, Inc. 2008], video-cube segmentation [Wang et al. 2005], and video cutouts [Li et al. 2005]. 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 [Agarwala et al. 2004] and matting [Chuang et al. 2002; Levin et al. 2006; Wang and Cohen 2007] to extract general scene components without arbitrary cuts.

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 single-view tools from Horry et al. [1997], Oh et al. [2001] to video. We are also inspired by the Video Trace technique [van den Hengel et al. 2007] which uses video as an input to interactively model static objects. We show how structure-from-motion [Hartley and Zisserman 2000] 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.

Our representation is also inspired by image-based rendering acceleration techniques that use impostors composed of texture-mapped meshes [Sillion et al. 1997; Darsa et al. 1997; Décoret et al. 1999]. They render a static 3D scene from a given viewpoint and use depth information from a z-buffer. Our context is different, the video mesh seeks to represent the texture, motion, as well as complex occlusions in video sequences.

Mesh-based video compression Using triangle meshes to represent motion is well known in the video compression community [Altunbasak and Tekalp 1997; Celasun and Tekalp 2000; Cammas et al. 2005]. These methods are designed for motion compensation and handle visibility by resampling and remeshing along occlusion boundaries. They typically do not support self-occlusions. However, for video editing, 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 finely encoding it with an alpha map. It has the added benefit of handling partial coverage and sub-pixel effects.

Multi-view stereo Recent multi-view algorithms are able to recover depth in complex scenes from video sequences [Seitz et al. 2006]. However, these techniques have difficulties with non-Lambertian materials and moving objects and also require camera motion. When these techniques succeed, Zhang et al. demonstrate how to perform a number of video special effects [Zhang et al. 2009]. The video mesh is complementary to these techniques. We can use depth maps to initialize the 3D geometry and our modeling tools to address challenging cases such as scenes with moving objects. Furthermore, in addition to existing effects, we provide extended camera control including changing the 3D viewpoint and refocusing.

Mosaicing Background collections and mosaicing can be used to create compound representations of video, e.g., [Irani et al. 1995; Agarwala et al. 2005; Hermans et al. 2008]. Recently, Rav-Acha et al. [2008] introduced unwrap mosaics to perform video editing on deformable objects. This method represents the object texture and the occlusions without explicitly recovering the 3D geometry of the object. 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. See also Szeliski's survey on video-based rendering [2004].

2 The Video Mesh Data Structure

We begin by first illustrating how the video mesh 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 [Gomes et al. 1998]. 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. They are stored in a global vertex table that is indexed by frame number and track index (Fig. 2). A number of those entries can be empty because a vertex track often does not last the entire video. A sparse

```
class VideoMesh {
    // vertex list per frame
    List<Vertex>[nFrames] vertexTable;
    // face list per frame
    List<Face>[nFrames] faceTable;
    TextureData texture;
    Camera[nFrames] cameraPath;
}

class Vertex {
    // current position in screen space
    Vector2f position;
    float z; // depth (optional)

    // Original position used as texture
    // coordinates. Equal to position if
    // the video has not been edited.
    Vector2f originalPosition;
    float originalZ;

    // TRACKED, S_VIRTUAL, or T_VIRTUAL
    enum VertexType type;

    // Accessors use the vertexTable.
    // Can return null for temporal
    // virtual vertices.

    // predecessor in time
    Vertex getPrevious();
    // successor in time
    Vertex getNext();
}

class Face {
    int[3] vertexIndices;
    // tiles occupied by this face
    List<TextureTile> tiles;
    // used for temporal edits
    int tOriginal;
}
```

Figure 2: *Basic video mesh data structures.*

representation such as a hash table can be used for efficiency.

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 [Liu et al. 2005; Wang et al. 2006]), and we store it in a separate field. A vertex has access to its position in the next and previous frames through accessors that return the vertex in the same track at the next or previous frame index in the vertex table (Fig. 2).

Vertices also have a field to store depth information, which can be specified through a number of tools which we describe 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 with the pixel values from the original video, with texture coordinates defined by the original position of its vertices. The textures can also be edited to enable various video painting and compositing effects. Each face stores a list of pointers to texture tiles to enable the treatment of multiple layers, as described below.

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 also 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 To illustrate with a concrete example, consider a simple manipulation such as motion magnification that warps the video. One starts by tracking features over time. For this simple example, we assume that all tracks last the entire video sequence and that there is no occlusion or layering. 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 [Liu et al. 2005] or more complex temporal processing [Wang et al. 2006] 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 [Gomes et al. 1998].

The triangulation defines a piecewise-linear interpolation of the sparse information stored at vertices. This is implicitly applied to depth and motion to obtain dense motion fields and depth maps. While this yields satisfying estimates in smooth regions, discontinuities require more attention, as described below.

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 the 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., [Cammas et al. 2005], 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 [Molino et al. 2004]. 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 local alpha matte to disambiguate the texture (see Figure 3). Similar to temporal virtual vertices, their spatial counterparts also store position information that is extrapolated from their neighbors. We also 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 is not a closed loop, it ends at a singularity called a *cusp*. The triangle that the cusp belongs to gets 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. We simply cut the mesh and duplicate triangles locally. Depth ordering is specified through the per-vertex z information.

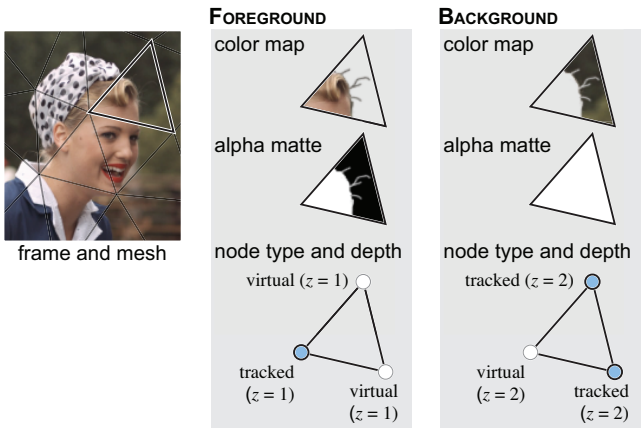


Figure 3: *Occlusion boundaries are handled by duplicating faces. Each boundary triangle stores a matte and color map. Duplicated vertices are either* tracked*, i.e., they follow scene points, or* virtual *if their position is inferred from their neighbors.*

2.3 Tile-Based Texture Storage

At occlusion boundaries, the video mesh is composed of several overlapping triangles and a pixel in the image plane can be assigned several 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 seeks to strike 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 [Hormann et al. 2007], 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. 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 allocate into 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, tiles also 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 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, tiles 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. Recall that we have added a one-pixel margin around the faces so that we can use bilinear filtering to maintain visual quality even after operations such as a change of viewpoint. To achieve interactive rendering performance, we cache tiles in texture memory as a single large atlas (e.g., 4096×4096), storing tiles as subregions. This caching strategy 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 [Agarwala et al. 2004], and once cut, the boundary can be refined using image matting [Chuang et al. 2002; Levin et al. 2006; Wang and Cohen 2007]. 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 is non-intersecting.
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, when 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. 7). This ensures that self T-junctions are processed top-down.

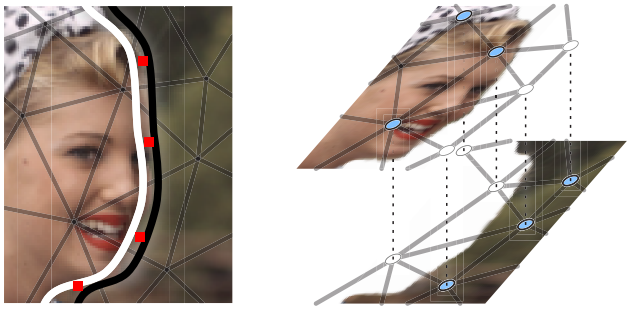
Four configurations To cut the mesh along the splines, we distinguish the four possible configurations:

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. 4).

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. 5). 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. 6).

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.

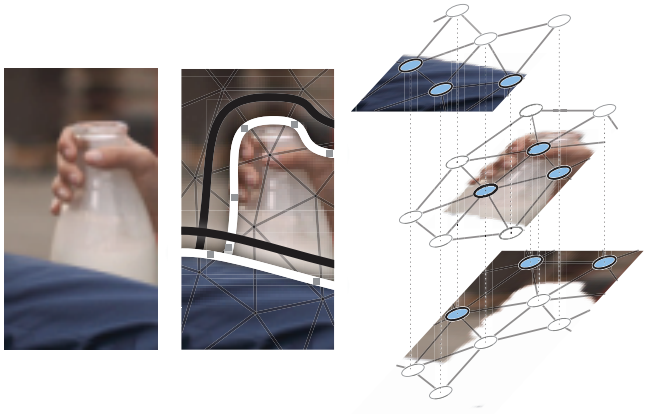


a) flat mesh and boundary spline b) cut video mesh with matted layers

Figure 4: *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.

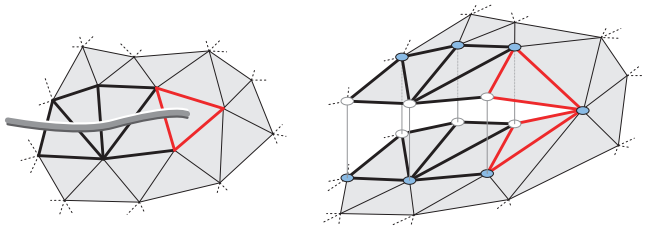
Motion estimation Cutting the mesh generates spatial virtual vertices without successors or predecessors in time. We estimate their motion by diffusion from their neighbors. For each triangle with two tracked vertices and a virtual vertex, we compute the translation, rotation, and scaling of the edges 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.

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. [2004].



(a) frame (b) video mesh and splines (c) video mesh topology

Figure 5: *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.*



(a) cusp seen in image plane with a spline ending (b) video mesh topology

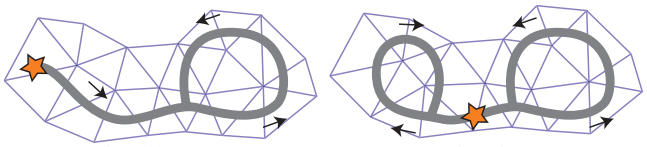
Figure 6: *Cutting the mesh with a cusp. This case is similar to the normal cut (Fig. 4) except that the vertex opposite to the cut edge is shared between the two copies.*

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.

Static camera: image-based modeling For scenes that feature a static camera with moving foreground objects, we provide simple tools inspired from the still photograph case [Horry et al. 1997; Oh et al. 2001] 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



(a) simple self T-junction (b) double self T-junction

Figure 7: *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.*

in the supplemental video.

Moving camera: user-assisted structure-from-motion For scenes with a moving camera, we build on structure-from-motion to simultaneously recover a camera path as well as the 3D position of scene points. For the camera, we retrieve its position and orientation at each time frame and a fixed focal length for the whole sequence. 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 [Hartley and Zisserman 2000] and register them in a global coordinate system by aligning to the camera path which does not change. We use an off-the-shelf structure-from-motion package [2d3 2008]. 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 additional 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. We have implemented several simple strategies to handle common configurations.

Mesh 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 backgrounds (Figure 1), we use a *background collection* [Szeliski 1996]. 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. A 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 stable, such as on the shirt of an actor (Figure 1), we modify Efros and Leung texture synthesis [1999] to search only in the same connected component as the hole, ensuring that only similar patches are copied.

Finally, for architectural scenes where textures are more regular and boundaries are straight lines, we proceed as Khan et al. [2006] 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 [Lévy 2003] and inpainting [Bertalmio et al. 2000] algorithms.

(a) original viewpoint

(b) new viewpoint

Figure 8: *The occlusion handling and per-vertex information encoded in the video mesh enable viewpoint changes. Top: original frame. Bottom: camera moved forward and left toward the riverbank. Note the proper occlusion between the riverbank, building, and Notre-Dame.*

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 or inserting an object into a video sequence. 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.

(a) focus on foreground

(b) focus on background

Figure 9: *Compositing and depth of field manipulation. We replicated the character from the original video and composited him. The depth information stored in the video mesh naturally achieves the required foreshortening. We also perform a rack focus using synthetic depth of field.*

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. We demonstrate in the companion video 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. 9). This effect can be previewed in real time by sampling 128 points over the aperture. We render a high-quality version with 1024 samples at about 2 Hz.

Object insertion and manipulation Copy-and-paste is supported by the video mesh. The user first delineates an object with splines and then the object can be replicated or moved anywhere by copying the corresponding faces. In addition, the depth structure of the video mesh enables occlusions between the newly added objects and the 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. 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, in the same spirit as the Photo Clip Art system proposed by Lalonde et al. [2007] for static scenes. 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.

(a) wide field of view,
camera close to the subject

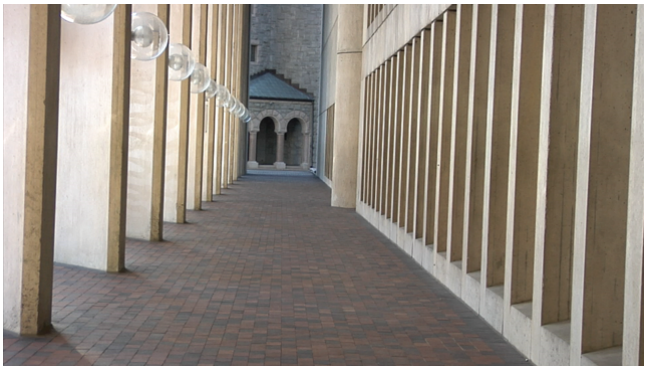
(b) narrow field of view,
camera far from the subject

Figure 10: *Vertigo effect enabled by the 3D information in the video mesh. We zoom in and at the same time pull back. This keeps the foreground character at the same size but modifies the perspective of the background.*

When we duplicate an animated object several times, we offset the copies in time to prevent unrealistically synchronized movements.

We can also use transparency to render volumetric effects. In the companion video, we demonstrate the insertion of smoke. To obtain volumetric attenuation and occlusion that depends on the geometry, we render 10 offset layers of 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 8. 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 view the copier from the side. Compared to existing techniques such as Video Trace [van den Hengel et al. 2007], 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. 10). Resolution can be a limiting factor in areas where there is significant magnification. However, we believe that in many cases, the additional freedom brought by repositioning the camera after the shot makes up for the loss of resolution.



(a) original frame

(b) relit with fog and shadows
(best viewed on screen)

Figure 11: *Relighting a daytime scene to depict night. Left: original video frame. Right: Scene relit with simulated lights, fog, and shadows cast by composited characters.*

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 (Fig. 11). We use the original pixel value as diffuse material color, position light sources where light bulbs are visible, and add volumetric fog. We render the scene using raytracing to simulate shadows and participating media [Pharr and Humphreys 2004].

Performance We implemented our prototype in DirectX 10. We ran all 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. Memory usage varies between 1 GB to 4 GB depending on the sequence, which is typical for video editing application. 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).

Discussion Although our approach enables useful effects, it is not without limitations. First, since we rely on off-the-shelf components, we inherit their shortcomings. As mentioned earlier, users sometimes have to pinpoint a vertex to correct misalignment that may appear in the structure-from-motion output. We also found that the publicly available matting techniques [Wang and Cohen 2007; Levin et al. 2006] were challenged by some of our sequences as can be seen for instance in the colonnade example. More directly related to our approach, the video mesh is a coarse data structure: high-frequency motion and thin features would be difficult to accurately represent. For instance, the video mesh would not represent blades of grass flowing in the wind well, 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. To summarize, the video mesh enables effects that combine motion, depth, and video, and works best with objects and motion that are of medium scale or larger.

5 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

2D3, 2008. boujou 4.1. `http://www.2d3.com/`.

ADOBE SYSTEMS, INC., 2008. After effects cs4.

AGARWALA, A., HERTZMANN, A., SALESIN, D. H., AND SEITZ, S. M. 2004. Keyframe-based tracking for rotoscoping and animation. *ACM Transactions on Graphics 23*, 3, 584–591.

AGARWALA, A., ZHENG, K. C., PAL, C., AGRAWALA, M., COHEN, M., CURLESS, B., SALESIN, D. H., AND SZELISKI, R. 2005. Panoramic video textures. *ACM Transactions on Graphics 24*, 3 (Aug.), 821–827.

ALTUNBASAK, Y., AND TEKALP, A. 1997. Occlusion-adaptive, content-based mesh design and forward tracking. *IEEE Transactions on Image Processing 6*, 9 (Sep), 1270–1280.

APPLE, INC., 2007. Motion 3.0.2.

BERTALMIO, M., SAPIRO, G., CASELLES, V., AND BALLESTER, C. 2000. Image inpainting. In *Proceedings of the ACM SIGGRAPH conference*, ACM, 417–424.

CAMMAS, N., PATEUX, S., AND MORIN, L. 2005. Video coding using non-manifold mesh. In *Proceedings of the 13th European Signal Processing Conference*.

CELASUN, I., AND TEKALP, A. 2000. Optimal 2-d hierarchical content-based mesh design and update for object-based video. *IEEE Transactions on Circuits and Systems for Video Technology 10*, 7 (Oct), 1135–1153.

CHUANG, Y.-Y., AGARWALA, A., CURLESS, B., SALESIN, D., AND SZELISKI, R. 2002. Video matting of complex scenes. *ACM Trans. Graph 21*, 3, 243–248.

DARSA, L., SILVA, B. C., AND VARSHNEY, A. 1997. Navigating static environments using image-space simplification and morphing. In *SI3D*, 25–34, 182.

DÉCORET, X., SILLION, F. X., SCHAUFLER, G., AND DORSEY, J. 1999. Multi-layered impostors for accelerated rendering. *Comput. Graph. Forum 18*, 3, 61–73.

EFROS, A. A., AND LEUNG, T. K. 1999. Texture synthesis by non-parametric sampling. In *IEEE International Conference on Computer Vision*, 1033–1038.

GOMES, J., DARSA, L., COSTA, B., AND VELHO, L. 1998. *Warping and morphing of graphical objects*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

HARTLEY, R., AND ZISSERMAN, A. 2000. *Multiple View Geometry in Computer Vision*. Cambridge University Press, June.

HERMANS, C., VANAKEN, C., MERTENS, T., REETH, F. V., AND BEKAERT, P. 2008. Augmented panoramic video. *Computer Graphics Forum 27*, 2 (Apr.), 281–290.

HORMANN, K., LÉVY, B., AND SHEFFER, A. 2007. Mesh parameterization: Theory and practice. In *ACM SIGGRAPH Course Notes*. ACM.

HORN, B., AND SCHUNCK, B. 1981. Determining Optical Flow. *Artificial Intelligence 17*, 1-3, 185–203.

HORRY, Y., ANJYO, K., AND ARAI, K. 1997. Tour into the picture: Using a spidery mesh interface to make animation from a single image. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, 225–232.

IRANI, M., ANANDAN, P., AND HSU, S. 1995. Mosaic based representations of video sequences and their applications. In *ICCV*, 605–611.

KHAN, E. A., REINHARD, E., FLEMING, R., AND BUELTHOFF, H. 2006. Image-based material editing. *ACM Transactions on Graphics 25*, 3 (July). Proceedings of the ACM SIGGRAPH conference.

LALONDE, J.-F., HOIEM, D., EFROS, A., ROTHER, C., WINN, J., AND CRIMINISI, A. 2007. Photo clip art. *ACM Transactions on Graphics 26*, 3 (July). Proceedings of the ACM SIGGRAPH conference.

LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2006. A Closed Form Solution to Natural Image Matting. *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Volume 1*, 61–68.

LÉVY, B. 2003. Dual domain extrapolation. *ACM Transactions on Graphics 22*, 3 (July). Proceedings of the ACM SIGGRAPH conference.

LI, Y., SUN, J., AND SHUM, H.-Y. 2005. Video object cut and paste. *ACM Transactions on Graphics 24*, 3, 595–600.

LIU, C., TORRALBA, A., FREEMAN, W. T., DURAND, F., AND ADELSON, E. H. 2005. Motion magnification. *ACM Transactions on Graphics 24*, 3, 519–526.

MOLINO, N., BAO, Z., AND FEDKIW, R. 2004. A virtual node algorithm for changing mesh topology during simulation. *ACM Transactions on Graphics 23*, 3, 385–392.

OH, B. M., CHEN, M., DORSEY, J., AND DURAND, F. 2001. Image-based modeling and photo editing. In *Proceedings of the ACM SIGGRAPH conference*.

PHARR, M., AND HUMPHREYS, G. 2004. *Physically Based Rendering: From Theory to Implementation*. August.

RAV-ACHA, A., KOHLI, P., ROTHER, C., AND FITZGIBBON, A. 2008. Unwrap mosaics: a new representation for video editing. *ACM Transactions on Graphics 27*, 3, 17:1–17:11.

SAND, P., AND TELLER, S. 2006. Particle video: Long-range motion estimation using point trajectories. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, IEEE.

SEITZ, S. M., CURLESS, B., DIEBEL, J., SCHARSTEIN, D., AND SZELISKI, R. 2006. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, IEEE, 519–528.

SHI, J., AND TOMASI, C. 1994. Good features to track. *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, 593–600.

SILLION, F., DRETTAKIS, G., AND BODELET, B. 1997. Efficient impostor manipulation for real-time visualization of urban scenery. *Computer Graphics Forum (Proc. EUROGRAPHICS'97) 16*, 3, 207–218.

SZELISKI, R. S. 1996. Video mosaics for virtual environments. *IEEE Computer Graphics and Applications 16*, 2 (March), 22–30.

SZELISKI, R. S. 2004. Video-based rendering. In *Vision, Modeling, and Visualization*, 447.

VAN DEN HENGEL, A., DICK, A., THORMÄHLEN, T., WARD, B., AND TORR, P. H. S. 2007. Videotrace: rapid interactive scene modelling from video. *ACM Transactions on Graphics 26*, 3, 86:1–86:5. Proceedings of the ACM SIGGRAPH conference.

WANG, J. Y. A., AND ADELSON, E. H. 1994. Representing moving images with layers. *IEEE Transactions on Image Processing 3*, 5, 625–638.

WANG, J., AND COHEN, M. 2007. Optimized color sampling for robust matting. In *Proceedings of the IEEE CVPR*, IEEE.

WANG, J., BHAT, P., COLBURN, R. A., AGRAWALA, M., AND COHEN, M. F. 2005. Interactive video cutout. *ACM Transactions on Graphics 24*, 3, 585–594.

WANG, J., DRUCKER, S., AGRAWALA, M., AND COHEN, M. 2006. The cartoon animation filter. *ACM Transactions on Graphics 25*, 3, 1169–1173.

ZHANG, G., DONG, Z., JIA, J., WAN, L., WONG, T., AND BAO, H. 2009. Refilming with depth-inferred videos. *IEEE Transactions on Visualization and Computer Graphics 15*, 5, 828–840.