1. Discussion

We discuss the design of our video editing system; in particular, the motivation behind designing the video mesh data structure, and why we chose particular components. We then describe the user interactions needed to create each of the examples.

1.1. System design

The primary motivation for the design of the video mesh is the desire for a flexible, user-assisted system for manipulating video in 3D. Our system is designed to let users work on a single frame at a time (*c.f.* Video Cubes [4]) and have their edits automatically propagate to nearby frames while maintaining control over the process. Keeping frames independent also dramatically simplifies our implementation of undo, which is critical in an interactive editing setting. At places, our system relies on a number of existing algorithms. Our choice of these off-the-shelf components is driven primarily by the ability to provide a simple user interface for controlling their output.

For motion analysis, we prefer point tracking to optical flow because dense motion fields are difficult to interpret and edit. In contrast, point tracking gives users only a handful of controls to manipulate. Our current implementation is based on RealVIZ MatchMover [3], which provides an automatic point tracking algorithm which is also user-controllable: users can place hard constraints and the algorithm optimizes for a smooth path satisfying the constraints. For depth estimation, our system uses structure-from-motion [1] to recover sparse depth estimates instead of stereo disparity maps, which, like optical flow, can be difficult to edit. In our system, depth inference is weakest on scenes with independently moving articulated bodies (for example, the Copier sequence). We let the user correct any misregistrations by constraining the shared vertices and camera path. Finally, for decomposing textures, we find that the default settings for published matting algorithms [2, 5] have difficulties with the large number of hard edges found in real scenes (for instance, the Colonnade and Copier examples). We address this issue by providing a spatially varying hardness weight along each spline, going from hard binary masking to soft matting.

1.2. User interaction

This section discusses the user interactions used to create each of the examples in the companion video. The workflow typically begins with first estimating motion and topology with the tracking and spline tools. The next step is computing the depth using one of our depth recovery operators, which yields a rough but complete video mesh upon which the various effects can be directly applied. We stress that the editing process is iterative: the user is free at any point to go back and correct any mistakes. For instance, if we discover, after drawing splines and computing depth, that a vertex has drifted, the user can simply drag the vertex to the correct location. Our system automatically updates the mesh in the background. The ratio of time spent between the various tools is scene dependent. We found that the majority of user time was spent tracking points and manipulating splines. Once the basic topology is complete, estimating depth and creating the various effects took only a small fraction of the time.

The Soda sequence in the companion video demonstrates the basic interactions used to create a video mesh. An expert user first spent roughly 5 minutes tracking points. With the actor's highly textured shirt, the process was mostly automatic and the majority of the user's time was spent correcting tracks near occlusion boundaries and adding a few extra vertices to ensure a uniform density of points. In the next step, the user spent approximately 10 minutes labeling the occlusion boundaries with splines. Although temporally propagating splines is mostly automatic in this sequence with smooth motion, the video does contain a number of changes in topology which necessitated user intervention. As shown in the video, in the next step, the user pulls on a few vertices in the mesh to turn the actor into a paper cutout. Finally, to complete the video mesh, the user spent another 5 minutes adjusting various matting parameters to ensure a consistent boundary in the video. Once the video mesh is complete, we could change the camera's position, aperture, and focus plane as shown in the video by mapping them to various mouse gestures. To create the smoke effect, we imported 10 copies of a precomputed smoke simulation video and scaled and positioned the layers in 3D.

The Colonnade sequence turned out to be our most challenging example. Even though the actor's motion was simple, it proved problematic for the point tracker because his pants and face had virtually no texture. Moreover, due to the change in mesh topology each time the actor's legs crossed during his walk cycle, the automatic spline propagation had to be restarted approximately every 20 frames. Overall, the user spent about 20 minutes tracking 42 points on the actor and 25 minutes adjusting splines. Like the previous example, once the mesh is complete, creating the effects was straightforward and demonstrated in real time in the companion video. Selecting the actor in space-time is trivial because he forms a single connected component in the video mesh. In our interface, once a component is selected for copying, the user can interactively drag and drop to choose the destination, optionally applying a temporal offset to decorrelate the movement, or a flip to let the video play backward. We also found that snapping the lowest pixel of object to the triangle under the cursor to be a useful tool. Similar to copy/paste, our interface lets the user point and click to place a light source for relighting. The

companion video demonstrates space-time copy-paste, interactive post-exposure camera control, and relighting.

The Copier sequence was our most fully automated sequence. We first automatically tracked points in the sequence, manually adding only a handful of points. Next, the user lassoed the points on the lid of the copier on one frame, exporting the set of tracks to Boujou for structure-from-motion estimation. The process was repeated for the chassis of the copier. After recovering two sets of 3D points and camera paths, we register them together by enforcing the camera paths to be the same. Tracking and 3D estimation only took about 5 minutes. Since the topology is relatively simple, the user only spent 5 minutes cutting the copier out of the background. Most of the time was spent converting spline control points from "smooth" to "corner" to respect the sharp geometry. Finally, less than one minute was spent enforcing lines in the mesh which correspond to straight segments in the scene: the user clicks on two vertices to create a segment that lasts through the entire sequence. Like the Colonnade sequence, once we had a complete video mesh, we composited multiple copies of the actor onto the glass of the copier by transforming a component of another video mesh in space and time. In this case, we keyframed the character's transparency so he smoothly comes into view. The companion video demonstrates interactive viewpoint control while the video plays.

In contrast to the Colonnade sequence where the camera was also stationary, our system worked well on the Notre Dame sequence despite its complex geometry. Automatic tracking was accurate on the boat since it had a smooth trajectory and a moderately textured surface. Modeling the geometry was straightforward: splines separated the scene components over which the facades were applied. The only difficulty we encountered was due to the relatively large z-range compared to image resolution, where a small error in the placement of a modeling facade may result in a large distortion in depth. Overall, the video mesh took the authors roughly 20 minutes to create: about 10 minutes for tracking and rotoscoping the boat and 10 minutes for depth modeling. The companion video shows the quality of the recovered depth map and shows a large viewpoint change by moving the camera onto the boat as it sails down river.

2. Tiled Texture

Figure 1 illustrates how our tiled texture allocator works when an initially flat video mesh is fully cut by a boundary spline.

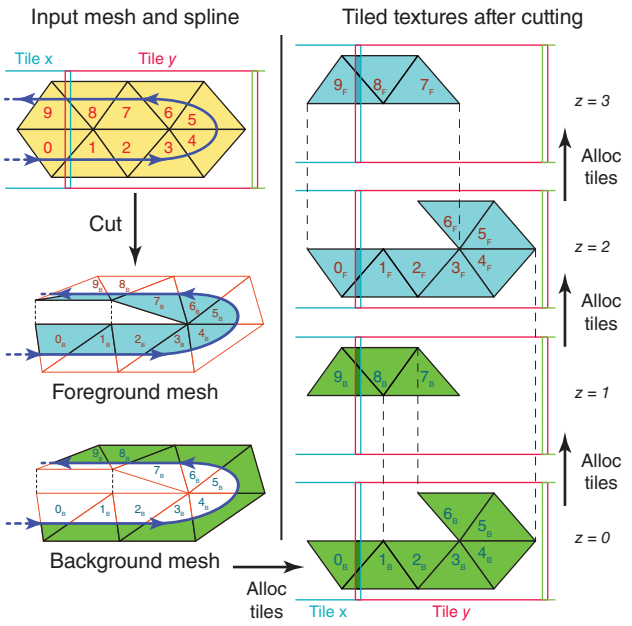

Figure 1. **Left:** An initially flat video mesh falls into two overlapping texture tiles x and y. Cutting the mesh with the spline results in a foreground and background mesh. Note how vertices in the interior are disconnected. **Right:** Texture allocation after cutting. Our implementation allocates space for the background mesh first, in the direction of the spline. Triangles 0_B and 1_B overlap the tile boundary and are assigned to both tiles. After allocating $0_B \ldots 6_B$, because the interior edges are disconnected, 7_B through 9_B are assigned to a new tile. We then allocate 0_F, which also needs a new tile since they do not share any edges with the triangles at levels $z = 0$ and $z = 1$.

References

[1] 2d3. boujou 4.1, 2008. http://www.2d3.com/.

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

[3] RealVIZ Corporation. RealVIZ MatchMover. http://sfx.realviz.com/products/mpro/index.php, 2007.

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

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