Exploring Efficient Image-Based 3D Reconstruction and Rendering
Soo Hark (Scott) Chung
Department of Computer Science, Stanford University

Motivation
Despite continued advancements of 3D graphics capabilities of consumer PCs and mobile devices, there’s a surprising lack of 3D content on the web. The goal of the project was to explore the feasibility of building an easy pipeline for publishing 3D content on the web through image-based structure from motion and rendering. If such a pipeline could be built, we may not be too far off from using 3D content for purposes such as online shopping. I propose one possible implementation of such a pipeline and run through the steps in this project. I test and evaluate state-of-the-art algorithms that are available to study the feasibility of my proposed pipeline, and to try by hand at implementing and improving a small slice of it.

Proposed Pipeline

- Take Photos of Object
- Segment foreground of photos
- PMVS (patch-based multi-view stereo) to retrieve point cloud
- Poisson Surface Reconstruction to retrieve triangular mesh
- Project photos on to mesh to color vertices
- Mesh Decimation using Quadratics Metric
- Efficient view-dependent rendering

Some comments:
- I decided to use structure from motion from images since it would make the pipeline accessible without special hardware such as 3D scanners.
- PMVS and Poisson reconstruction are both state of the art algorithms that are popular for similar image processing pipelines.
- I decided to project colors from each image to mesh vertices rather than using texture coordinates and a texture image. Using vertex colors avoids having to publish high-resolution texture images and works better with most view-dependent rendering algorithms.
- Mesh decimation is necessary for web purposes because mesh reconstruction returns a very dense mesh with a high number of vertices.
- I wanted to do image-based view-dependent rendering, because it would avoid having to deal with material properties which are difficult to derive and avoid the cartoony look that comes from not having any lighting effects. This affected my decision to use colored vertices and my choice of decimation algorithm.

PMVS/ Poisson Reconstruction

PMVS (patch-based multi-view stereo) is a very popular state-of-the-art algorithm for dense point cloud reconstruction. It still has its faults. It needs corners and textures to latch onto. It fails to capture points on smooth surfaces without texture. It also requires a fairly large number of photos to get a good point cloud.

Poisson surface reconstruction is a popular algorithm for building closed meshes out of point clouds. Aside from the signature blobby look of meshes returned by this algorithm, there are two issues when using this with PMVS. First, the point cloud returned from PMVS can be patchy. Second, the normals of points along edges aren’t stable. This is especially worse for points along edges of textures, because you can have unstable normals even though the geometry is actually smooth there.

Mesh Decimation

I decided to try my hand at implementing a mesh decimation algorithm and trying to see if I can make any improvements. As a baseline, I decided to implement a classic mesh decimation algorithm that collapses edges that results in the smallest quadrics errors.

I decided to implement an improved version of the algorithm that takes the color data at each vertex into account. It works by projecting each vertex to high dimensions where the extra dimensions encode color information and calculating quadrics in the higher dimension. The method is a natural extension of the baseline implementation and works well for meshes with vertex colors as opposed to mapped textures. Initially I wanted to append the color data from different images to each vertex and use the resulting vector to calculate the error. However, with this scheme, you get 3+3n dimensions where n is the number of images. This obviously becomes infeasible when you have 10 or more images, which is necessary for the type of view-dependent rendering algorithms I had in mind for the pipeline. Instead, I settled for combining all the images into a single texture and applying it to the vertices. This way, I can do the quadrics calculations with 6-dimensional vectors.

Mesh Decimation Results

I rendered images of the mesh from the same camera angle before and after decimating the mesh to have 1% of the original number of vertices. Here is a zoom-in on a region of the image.

Conclusion
First of all, a pipeline like this being used widely to create 3D content for the web is probably far off in the future. Both structure for motion and surface reconstructions have a long way to go before they are reliable enough for this sort of application. I've managed show that mesh decimation using quadratics works well for a mesh with vertex colors. The current algorithm can be improved by experimenting with different ways of combining the images from different cameras into vertex colors used for mesh decimation.

The next step is to devise an efficient view-dependent rendering algorithm. The closest technique I've seen to the type of solution I'm looking for is called the eigen texture algorithm. I also want to explore ways of dealing with high frequency content in the texture (i.e. if I have a fuzzy surface) in a way that’s consistent with the view-dependent rendering algorithm.