A Quick and Automatic Image Based Modeling and Rendering System

Keith Yerex and Neil Birkbeck and Dana Cobzas and Martin Jagersand

Computing Science, University of Alberta, Canada

Abstract

We present a complete and automatic system for object capture from video, Internet delivery, and real-time photo-realistic rendering. Shape from silhouette is used to capture an approximate enveloping geometry, the visual hull. Texture coordinates are generated automatically under a minimum distortion criterion. To account for the difference between the true shape of the object and the captured visual hull we use Dynamic texturing, *a view-dependent texturing method, which explicitly captures geometric deviations and applies a texture based correction. These models are then efficiently coded, and delivered over the Internet.*

Categories and Subject Descriptors (according to ACM CCS): I.3.8 [Computer Graphics]: Applications

1. Introduction

Using images to partly, or completely replace the need for detailed geometric models, image based modeling and rendering (IBMR) promises to enable quick and easy modeling of complex objects from photographs or video. However, few methods have actually achieved the "quick and easy" goal, and as a result, there are few commercially available IBMR systems. The most well known consumer application is the Quicktime VR system [Che95], which renders panoramic images. Even with the restricted nature of these image based objects the QTVR system has been successful due to its robustness and ease of use. Most other available software only provide tools for creating conventional 3D models from images, rather than image based models. Interactive modeling tools such as PhotoModeler, and RealViz ImageModeler, build geometry and acquire textures from images by having the user manually identify corresponding features in multiple images. Some more automatic commercial systems exist, such as Canon's 3D S.O.M. [ABT03] which uses shape from silhouette, but models are still geometry based, which results in low quality renderings, since the visual hull captured from silhouettes only approximates the object.

Here, we present a complete system for acquiring, encoding, and rendering image based objects. Our image based models use the visual hull as an approximate geometry, onto which *dynamic texturing*, a view-dependent texturing method is applied. The entire process takes place as follows. We take an input video sequence of a target object rotating on a turntable. From the input sequence, we extract object silhouettes, and compute the visual hull from them. A set of texture coordinates is generated automatically for the visual hull geometry. Each example view is them projected into that common texture space where we build the dynamic texture basis. The geometry and dynamic texture are compressed in a progressive format, in preparation for Internet distribution. Finally, the user downloads these models using our viewer, where they are rendered in real-time with hardware acceleration.

2. Shape from silhouette

The silhouette of an object s_v is the set of points where the object projects on an image plane from a particular view. Given s_v, and camera parameters v we know the object is inside the volume of the silhouette cone S_v defined by all rays from the camera center passing through all points in s_v. Since this holds for all views, given a set of views V, we can constrain the object's volume further, to the intersection of all silhouette cones: $\bigcap_{v \in V} S_v$. The limit of this volume as the number of distinct views $|V| \rightarrow \infty$ is known as the *visual hull* of the object, provided that no views in V are are centered inside the convex hull of the object. From a finite number of views, the volume acquired is called the approx-

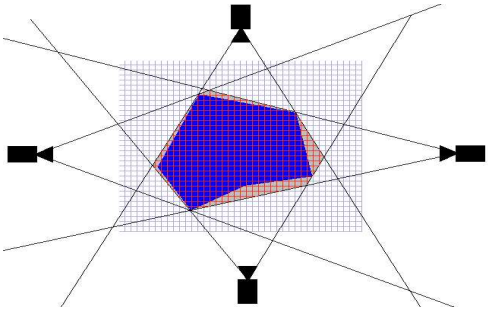

Figure 1: *Marching intersections data structure*

imate visual hull, but we will simply refer to it as the visual hull.

Before these intersections can be performed, the silhouettes must first be detected in the images. We segment the object from the background using a statistical color segmentation method. The user selects some region(s) in one or a few example image(s) to identify a set of pixels in the solid colored background. Principle components analysis is applied to the selected RGB pixels, giving us a color space that is well aligned to the statistical properties of the selected background pixels. We apply a simple distance threshold after transforming each pixel into the new color space.

Each image is automatically calibrated using the pattern described in detail in [ABT03], and it's silhouette is extended into a cone. To represent and intersect these cones, we use *Marching Intersections* (MI). Developed and used for SFS recently by Tarini et al [TCMR02], the MI data structure consists of 3 sets of rays, each parallel to one of the three axes (X Y or Z). For each set, there is an NxN grid of rays, so they combine to form an NxNxN cube. Each ray stores all points along its path where the volume being represented is entered or exited. This data structure stores precisely the information necessary to for the marching cubes algorithm to render the surface properly without interpolation. Figure 1 shows a 2D illustration of the structure.

The SFS algorithm iteratively intersects a single MI structure with each silhouette cone. Each intersection is performed in 2D by projecting MI rays into the image plane of the current silhouette, intersecting those rays with the 2D silhouette, and lifting the 2D intersection points back onto the corresponding rays in 3D.

3. Texturing

3.1. Texture Atlas

Dynamic texturing requires that our geometric model have a globally consistent set of texture coordinates, as opposed to using different texture coordinates for each view which is common in other view dependent texturing methods [DTM96]. Previous dynamic texturing implementations have used simple methods, such as manually assigned coordinates (for very simple models) and average observed point positions (for small variations in viewing angle) [CYJ02]. In the case of the visual hull, neither option is reasonable. Texture coordinates must be generated automatically under the following constraints:

- No overlap: a point in the texture should map to a unique point on the model's surface
- Minimal surface distortion: equal areas and distances on the surface of the model should be represented by equal areas and distances in the texture
- Maximal texture resolution usage: as many pixels as possible in the texture should be used (no big blank regions)

An automatically generated mapping of this kind is often referred to as a *texture atlas*. We generate a texture atlas using the conformal mapping algorithm described in [LPRM] by Levy et al. The result is then stretched to maximize texture usage with space optimized texture maps [DS02]. We simplify the algorithm by weighting occupied areas fully, and empty areas with zero, rather than using image frequency information as weights. Texture atlas results are shown in figure 2.

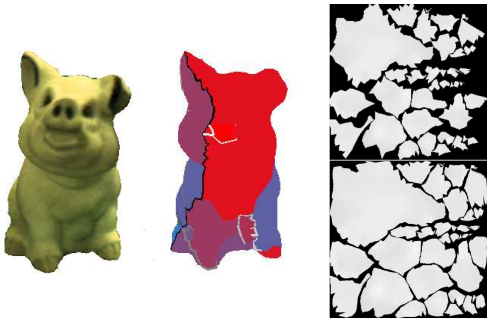

Figure 2: *The left image shows a textured rendering, the center shows the subdivided model, with lines representing features of high curvature, and colored surfaces representing different charts. The right images show the texture atlas of the same model before and after space optimization.*

3.2. Texture acquisition

We must warp the images from every sample view into textures. However, some parts of the object's surface - and therefore regions of the texture - are not visible in every image.

We grab texture from visible portions of the surface, while filling unavailable regions with an average texture. Finding the visible portion of the texture is somewhat difficult, since the model may be self-occluding. We use a z-buffer based method with OpenGL to calculate texture visibility per texel.

3.3. Dynamic Texture

The dynamic texture is a generalized form of view dependent texturing, where rather than blending between observed texture images, to generate the texture for a new view, we blend a general linear basis. This basis can be derived from the observed textures in such a way that many fewer basis vectors than there are observed textures are used to represent the same texture variability.

First, consider interpolated view dependent texturing as in [DTM96]. When rendering, a new view with parameters identical to one of the original sample views I_k, the texture derived from image I_k, $T_k = w(I_k)$ is used to texture the model, where w is defined by a 3D mesh with texture coordinates as discussed in section 3.1. At all other viewing positions, some linear blending of near views is used, with a vector of weights x based on their similarity to the current view. This can be expressed mathematically by a matrix multiplication, Where the columns of T contain the sample views.

$$t = Tx \tag{1}$$

The major variability in T is due to geometric parallax error and illumination differences. Through an analytical derivation, a first order linear basis can be found to represent these types of variability [CYJ02]. Meaning that for large image sets, we can find a new basis B with many fewer columns than T, such that $T \approx \tilde{T} = BY$. Textures are then generated as $t = BYx$, and the number of basis images, and overall memory consumption is reduced.

While B could be be computed analytically, given exact geometric knowledge of the scene, camera and lighting, this is seldom feasible in image-based approaches. Instead we use the knowledge that there exist a subspace spanning T to obtain the best (in the least square sense) B through Principle components analysis. We calculate M as the eigen-vectors of $T^T T$. A dimensionality reduction is achieved by using only the first n eigen-vectors $M_{1..n}$. Thus, our texture basis is $B = TM_{1..n}$ and our coefficients are $Y = M_{1..n}^T$.

To estimate the coefficients for intermediate poses, we interpolate between the coefficients of sampled poses. For efficient implementation, cubic interpolation is applied during preprocessing, and results are stored in a set of 2D look-up tables (one for each basis image) which map view direction to blending coefficient. Entries in the blending tables are then bilinearly interpolated during rendering.

The benefits of using the dynamic texture basis rather than standard view-dependent texturing, is that significantly less storage is required. The down-side is that every element of the texture basis will have an effect on the result at any view, instead of only the nearest few. This will require a little more work when rendering. However, in current graphics architectures, bandwidth and memory limitations are a greater problem than computation when it comes to image based rendering methods.

3.4. Rendering

Rendering of dynamic textures consists of blending a large basis and transforming colors to the RGB color space of the frame buffer (YUV color space is used as described in section 4). These operations are well suited for implementation in graphics hardware or MMX, and we support both.

In the hardware implementation, each RGBA texture stored in OpenGL represents 4 basis textures from a single color channel (Y,U or V) scaled and biased to fit in the range (0,1). In each rendering pass, as many basis images as possible (4 times the number of available texture units) are multiplied by their coefficients, the results are summed, and multiplied by the row of the color conversion matrix that applies to the current color channel. Between passes, we use OpenGL blending to add/subtract results with the frame buffer contents.

4. Coding

Conventional image based objects rely on numerous images to generate novel views. The associated storage requirements for such objects can be quite large, [GGSC96] reports uncompressed objects requiring up to 1GB. In our case, the uncompressed storage requirements are typically on the order of 5-10MB. Although the dynamic texture method already compresses the data by reducing the number of images stored compared to standard view dependent texturing, 10MB is still too large for quick Internet viewing. Data is organized and compressed as follows.

The dynamic texture process described in section 3.3 is applied separately to each color channel in YUV color space. We use more, and higher resolution basis textures for the luminance component (Y) than the color components (U and V). We have found that typically 16 128×128 basis textures are sufficient for each of the U and V channels, whereas we use 32 or 64 256×256 basis textures for Y.

Geometry is stored first in the file, followed by basis images and their corresponding blending tables. Since the basis images are computed using PCA, they have a natural order of importance based on their eigen-values. Progressive display of partially downloaded models is done by coding the basis in this order in the file, so models can be displayed as soon as the first few basis images are available.

Optionally JPEG compression is used to further compress both basis images and blending tables. Although neither are really images, lossy image compression works quite well. As shown in figure 4, artifacts caused by modulating a compressed basis are similar to artifacts one would find by JPEG compressing a rendering generated by an uncompressed basis. Additionally, JPEG compression of the look-up tables did not introduce any noticeable visual artifacts.

Figure 3: *Renderings of captured models of a Homer Simpson figure, and a sheep finger puppet*

Compression	None	gzip	jpeg 100%	50%	25%
Size(MB)	8.79	4.71	4.37	0.68	0.42

Figure 4: *close-up comparison of an object rendered uncompressed (left) vs JPEG compressed at 25% (right). Table shows the size of this model with various settings.*

We also apply GZIP compression to all data. This reduces the size of geometric data as well as providing loss-less compression of basis images when the highest possible quality is required.

5. Results and Conclusion

We have developed and described a robust and automatic system for capturing objects. An approximate geometric structure is first built from the object's silhouette in multiple images (all taken automatically using a video camera and turntable). A texture coordinate mapping is then built for this geometry, and a view dependent texturing is applied in the form of texture basis modulation (dynamic texture).

With hardware accelerated rendering, dynamic texturing is easily performed in real-time, even with multiple simultaneously rendered dynamic textured objects. Results shown in figure 3 were captured from 64 views. Textures applied to the Homer Simpson model are blended from a basis of only 16 textures, the knit ornament, and the pig shown in figure 2 are rendered with 32 basis textures for the Y channel, and 16 for each of the color channels.

Downloadable software and models are available at `http://www.cs.ualberta.ca/ vis/ibmr`.

References

[ABT03] A. BAUMBERG A. L., TAYLOR R.: 3d s.o.m - a commercial software solution to 3d scanning. In *Proceedings of Vision, Video and Graphics* (2003).

[Che95] CHEN S. E.: QuickTime VR — an image-based approach to virtual environmen navigation. *Computer Graphics 29*, Annual Conference Series (1995), 29–38.

[CYJ02] COBZAS D., YEREX K., JAGERSAND M.: Dynamic textures for image-based rendering of fine-scale 3d structure and animation of non-rigid motion. In *Proceedings of Eurographics* (2002).

[DS02] DRETTAKIS, SEIDEL H.: Space optimized texture maps. In *Proceedings of Eurographics* (2002).

[DTM96] DEBEVEC P. E., TAYLOR C. J., MALIK J.: Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. *Computer Graphics 30*, Annual Conference Series (1996), 11–20.

[GGSC96] GORTLER S. J., GRZESZCZUK R., SZELISKI R., COHEN M. F.: The lumigraph. *Computer Graphics 30*, Annual Conference Series (1996), 43–54.

[LPRM] LEVY B., PETITJEAN S., RAY N., MAILLOT J.: Least squares conformal maps for automatic texture atlas generation.

[TCMR02] TARINI M., CALLIERI M., MONTANI C., ROCCHINI C.: Marching intersections: an efficient approach to shape from silhouette. In *Proceedings of VMV 2002* (2002).