Omnidirectional texturing of human actors from multiple view video sequences

Alexandrina Orzan, Jean-Marc Hasenfratz

HAL Id: inria-00281378

https://inria.hal.science/inria-00281378

Submitted on 22 May 2008

Omnidirectional texturing of human actors from multiple view video sequences

Alexandrina Orzan,* Jean-Marc Hasenfratz†

Artis,‡ GRAVIR/IMAG - INRIA

Abstract

In 3D video, recorded object behaviors can be observed from any viewpoint, because the 3D video registers the object's 3D shape and color. However, the real-world views are limited to the views from a number of cameras, so only a coarse model of the object can be recovered in real-time. It becomes then necessary to judiciously texture the object with images recovered from the cameras. One of the problems in multi-texturing is to decide what portion of the 3D model is visible from what camera. We propose a texture-mapping algorithm that tries to bypass the problem of exactly deciding if a point is visible or not from a certain camera. Given more than two color values for each pixel, a statistical test allows to exclude outlying color data before blending.

1 Introduction

Currently, visual media such as television and motion pictures only present a 2D impression of the real world. In the last few years, increasingly more research activity has been devoted to investigate 3D video from multiple camera views. The goal is to obtain a free-viewpoint video, where the user is able to watch a scene from an arbitrary viewpoint chosen interactively.

The possible applications are manifold. A free-viewpoint system can increase the visual realism of telepresence technology [1] , thus enabling users in different locations to collaborate in a shared, simulated environment as if they were in the same physical room. Also, special effects used by the movie industry, such as *freeze-and-rotate* camera, would be made accessible to all users.

For free-viewpoint video, a scene is typically captured by N cameras. From the views obtained by the cameras a 3D video object, with its shape and appearance, is created. The shape can be described by polygon meshes, point samples or voxels. In order to make the model more realistic, appearance is typically described by the textures captured from the video streams. Appearance data is mapped onto the 3D shape, thus completing the virtual representation of the real object. The 3D video object can be seamlessly blended into existing content, where it can be interactively viewed from different directions, or under different illumination.

Since people are central to most visual media content, research has been dedicated in particular to the extraction and reconstruction of human actors.However, the system used in this article is not restricted to human actors, as [2]. Moreover, it allows the acquisition of multiple objects present in the scene.

The rest of the paper proceeds with a review of related work in section 2. Section 3 will be dedicated to describing the proposed method of texture-mapping, after which results and future tasks are discussed.

2 Previous Work

Over the last few years, several systems with different model reconstruction and different ways of texturing the 3D model have been proposed.

---

[1]"Telepresence technology" enables people to feel as if they are actually present in a different place or time (S. Fisher & B. Laurel, 1991) or enables objects from a different place to feel as if they are actually present (T. Lacey & W. Chapin, 1994).

2.1 3D Model reconstruction

Two different approaches of model reconstruction have been studied in the recent years: model-free and model-based reconstruction.

Model-free reconstruction makes no a priori assumptions on scene geometry, allowing the reconstruction of complex dynamic scenes. In human modeling it allows the reproduction of detailed dynamics for hair and loose clothing.

Most model-free methods aim to estimate the visual hull, an approximate shell that envelopes the true geometry of the object [10]. To achieve this, object silhouettes are extracted from each camera image by detecting the pixels not belonging to the background.

The visual hull can then be reconstructed either by voxel-based or polyhedron-based approaches. The first approach discretizes a confined 3D space in voxels and carves away those voxels whose projection fall outside the silhouette of any reference view [7]. Polyhedron-based approaches represent each visual cone as a polyhedral object and computes the intersection of all visual cones [11, 14, 13].

The visual hull allows real-time reconstruction and rendering, yet it needs a large number of views to accurately represent a scene, otherwise the obtained model is not very exact.

Model-based reconstruction assumes that the real object is a human body and uses a generic humanoid model, which is deformed to fit the observed silhouettes [2, 8, 9]. Although it results in a more accurate model and permits motion tracking over time, this approach is restricted to a simple model and does not allow complex clothing movements. Moreover, it places a severe limitation on what can be captured (i.e. a single human body) and it is not real-time.

In this paper, the 3D model used is the one created in the context of CYBER-II project[2], a polyhedron-based model obtained in real-time.

2.2 Multi-view texture mapping

Original images from multiple viewpoint are often mapped onto recovered 3D geometry in order to achieve realistic rendering results [3]. Proposed methods for multi-texture mapping are either view-dependent or view-independent.

View-dependent texture mapping considers only the camera views closest to the current viewpoint. In between camera views, two to four textures are blended together in order to obtain the current view image [4, 5]. This method exhibits noticeable blending artifacts in parts where the model geometry does not exactly correspond to the observed shape. What's more, the result is usually blurred and the passing from one camera view to another does not always go unnoticed.

View-independent texture mapping selects the most appropriate camera for each triangle of the 3D model, independently of the viewer's viewpoint [2, 8, 13]. The advantage of this method is that it does not change the triangle texture when the user changes the viewpoint. Moreover, the blurred effect is less noticeable. However, the problem is that the best camera is not the same from patch to patch, even if they are neighboring. Here also, blending between visible views is necessary in order to reduce the abrupt change in texture at triangle edges.

Blending is done using various formulas that depend of:

- the angle between the surface normal and the vector towards the considered camera

- the angle between the surface normal and the vector towards the viewpoint

- the angle the vector towards a camera and the vector towards the viewpoint

Blending weights can be computed per vertex or per polygon [2, 4, 5]. Matsuyama [13] proposes using this method for determining each vertex color and then paints the triangles with the colors obtained by linearly interpolating the RGB values of its vertices. However, for large triangles, small details like creases in the clothes are lost.

Li and Magnor [12] compute the blending for each rasterized fragment, which results in a more accurate blending.

2.3 Visibility

Visibilities with respect to reference views are very important for multi-view texture mapping. For those parts that are invisible in a reference view, the corresponding color information should be ignored when blending multiple textures.

---

[2]http://artis.imag.fr/Projects/Cyber-II/

Debevec et al. [3] splits the object triangles so that they are either fully visible or fully invisible to any source view. This process takes a long time even for a moderately complex object and is not suitable for real-time applications. Matusik [14] proposes computing the vertex visibility at the same time that the visual hull is generated. Magnor et al. [12] solves the visibility problem per fragment, using shadow mapping. However, they require rendering the scene from each input camera viewpoint and is not real-time even with a hardware-accelerated implementation.

We propose a per pixel method that checks only polygon visibility and eliminates the wrong colors by considering only those colors that are close to a computed average.

3 Texture mapping algorithm

3.1 Model constraints

The polyhedron-based model-free method recreates the geometrical object at each frame. The number of polygons, their form and position in space vary greatly in time, so we cannot track vertices from one frame to another.

This means that it is impossible to decide the color of the polygons only once, at the beginning of the video. Color values have to be computed in real-time, for each frame.

3.2 Algorithm description

To achieve realistic rendering results, we use the projective texture mapping, a method introduced by Segal [15] and included in the OpenGL graphics standard. But the current hardware implementation of projective texture mapping in OpenGL lets the texture pass through the geometry and be mapped onto all back-facing and occluded polygons. Thus it is necessary to perform visibility check so that only polygons visible to a particular camera are texture mapped with the corresponding image.

A point p on the object's surface is visible from a camera c_i if (1) the triangle t_j to which the point belongs faces the camera and (2) the point is not occluded by any other triangles.

The first condition can be fast determined by checking the equation $n_{t_j} \cdot v_{c_i \to t_j} < 0$, where n_{t_j} is the triangle normal vector and $v_{c_i \to t_j}$ is the viewing direction from c_i towards the centroid of t_j.

Still, in a per-pixel approach, we do not have the geometrical data. We solve this problem by an additional rendering of the object from the current viewpoint, where we use the polygon ID as its color. Thus, we can determine what polygons are visible from the viewpoint and exactly which pixel of the current image view belongs to which triangle.

Determining if a point viewed by the viewer is occluded or not to the cameras is a less obvious problem. Methods to determine what points are occluded were briefly presented in the previous section. We propose to bypass the occlusion checking by doing a basic statistical test. The strong condition that has to be fulfilled is that for each pixel at least three cameras have to pass the first visibility test, and the majority has to see the correct color. Still, this is usually the case with a system having an evenly distributed camera configuration.

As all the cameras are calibrated prior to use and the images are acquired at the same time and in the same lighting conditions, we can compare colors and calculate distances in the RGB space [1, 6].

If a sufficient number of color values are available for a pixel, we compute the mean (μ) and the standard deviation (τ) for each of the R, G, B channels. Individual colors falling outside the range $\mu \pm \beta \cdot \tau$ for at least one channel are excluded. The factor β permits us to modify the confidence interval for which the colors are accepted. The classical normal deviation test considers β is 1. We experimentally concluded that it was best to set it at 1.17, to allow for slight errors in manipulating the color-values.

If less than three possible colors are available for a pixel, we do not exclude any of them.

A weighted mean of all contributing images is finally used for texturing each particular pixel. The blending weight is computed using the value of the $\cos(angle(n_{t_j}, v_{c_i \to t_j}))$.

If the pixel is invisible for all cameras, we compute its color using the color values of the neighbours whose color was already decided.

The algorithm runs as follows:

```
 1: for all polygons in the 3D model do
 2:    check if they are at least partially visible from the
       current view
 3: end for
 4: for all pixels in the image view do
 5:    for all cameras do
 6:       if the polygon that colored the pixel faces the
          camera then
 7:          retain the corresponding color
 8:       end if
 9:       if there are three or more colors then
10:          compute the mean and standard deviation
11:          for all colors do
12:             if they are not in the allowed interval
                then
13:                exclude
14:             end if
15:          end for
16:          compute the weighted mean
17:       else if there are two colors then
18:          compute the weighted mean
19:       else if there is no color then
20:          compute the color using neighbouring col-
             ors
21:       end if
22:    end for
23: end for
24: draw
```

4   Results

We tested this algorithm with the system used by the CYBER-II project. The system has 6 cameras, 4 in the front and 2 in the back, as seen in Figure 1[3].

For the front views, the algorithm succeeded in eliminating the wrong colors and in seamlessly mixing data from various cameras. Moreover, the pixel color doesn't change with the change of viewpoint. Images comparing view-dependent and view-independent algorithms, without occlusion checking, and our method can be seen in Figure 2.

However, for the back views, where the object is seen by at most 2 cameras, the algorithm does only a weighted average, without color elimination.

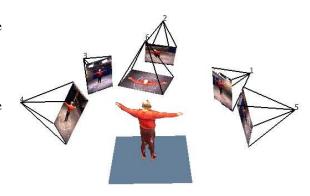[3]video sequences were acquired with the Grimage platform of Inria Rhône-Alpes

Figure 1: Camera setting

Figure 2: a) View dependent, b) View independent, c) Our method

We set the resolution of the rendered novel view to 512x512, and we tested the algorithm for a model of about 5000 polygons. On a Intel 2.40GHz CPU and a GeForce4 Ti 4800 graphic card, the frame rate is of 17 fps.

5   Conclusions and Future work

A per-pixel algorithm for multi-view texture mapping has been implemented. It succeeds in eliminating wrong colors for pixels viewed by more that 2 cameras, without doing a time-consuming occlusion checking.

Yet, further enhancements are both necessary and feasible. Thus, a hardware-implementation should be considered, since the mane time-consuming task in our algorithm is transferring information from the frame-buffer to the CPU. Moreover, we would like to consider a continuity in time of the computed pixel colors and a dynamic deactivation of the unused cameras.

References

[1] A. Agathos and R. Fishe. Colour texture fusion of multiple range images. In *Proceedings of the 4th International Conference on 3-D Digital Imaging and Modeling*, pages 139– 146, 2003.

[2] Joel Carranza, Christian Theobalt, Marcus Magnor, and Hans-Peter Seidel. Free-viewpoint video of human actors. *ACM Trans. on Computer Graphics*, 22(3):569 – 577, July 2003.

[3] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. *Computer Graphics*, 30(Annual Conference Series):11–20, 1996.

[4] Paul E. Debevec, Yizhou Yu, and George D. Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. In *9th Eurographics Workshop on Rendering*, 1998.

[5] Bastian Goldlücke and Marcus Magnor. Real-time microfacet billboarding for free-viewpoint video rendering. In *Proceedings of ICIP 2003, IEEE Computer Society*, volume 3, pages 713–716, 2003.

[6] L. Grammatikopoulos, I. Kalisperakis, G. Karras, T. Kokkinos, and E. Petsa. On automatic orthoprojection and texture-mapping of 3d surface models. In *ISPRS Congress - Geo-Imagery Bridging Continents*, 2004.

[7] Jean-Marc Hasenfratz, Marc Lapierre, and François Sillion. A real-time system for full body interaction. *Virtual Environments*, pages 147–156, 2004.

[8] Adrian Hilton and Jonathan Starck. Model-based multiple view reconstruction of people. In *IEEE International Conference on Computer Vision*, pages 915–922, 2003.

[9] Adrian Hilton and Jonathan Starck. Multiple view reconstruction of people. In *3D Data Processing, Visualization, and Transmission*, pages 357–364, 2004.

[10] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(2):150–162, 1994.

[11] Ming Li, Marcus Magnor, and Hans-Peter Seidel. Online accelerated rendering of visual hulls in real scenes. In *Journal of WSCG*, 2003.

[12] Ming Li, Marcus Magnor, and Hans-Peter Seidel. A hybrid hardware-accelerated algorithm for high quality rendering of visual hulls. In *Proceedings of the 2004 conference on Graphics interface*, pages 41–48, 2004.

[13] Takashi Matsuyama and Takeshi Takai. Generation, visualization, and editing of 3d video. In *3D Data Processing, Visualization, and Transmission*, page 234, 2002.

[14] Wojciech Matusik, Chris Buehler, and Leonard McMillan. Polyhedral visual hulls for real-time rendering. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 115–126, 2001.

[15] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 249–252, 1992.