Image-based rendering using plane-sweeping modelisation
Vincent Nozick, Sylvain Michelin, Didier Arquès

To cite this version:

HAL Id: hal-00733828
https://hal-upec-upem.archives-ouvertes.fr/hal-00733828
Submitted on 19 Sep 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Abstract: This paper presents a method to create new images of a scene from a small set of calibrated images using color matching. While most of the existing techniques attempt to build a 3D reconstruction of the scene, the only geometric constraint our color matching method requires is camera calibration. This paper describes a new adaptation of the plane-sweeping algorithm using a local dynamic scoring method that handle occlusions. We also propose a real-time implementation of our method using fragment programs. These modifications are well suited for on-line real-time rendering of dynamic scenes.

Keywords: real-time vision, image-based rendering, fragment programs
Let \( I_1 \ldots I_n \) denote a set of \( n \) images and \( P_1 \ldots P_n \) their respective \( 3 \times 4 \) projection matrices. \( I_x \) is the new image to be computed, \( \text{cam}_x \) is its virtual pinhole projective camera and \( P_x \) its associated projection matrix. We define a near plane and a far plane parallel to \( \text{cam}_x \) image plane such that all the objects of the scenes lie between near and far. For each pixel of each plane \( D_i \), a score and a color are computed according to the matching of the colors. The plane-sweeping algorithm can be explained as follows:

- initialize \( I_x \)'s score

- for each plane \( D_i \) from far to near
  - project all the input image \( I_1 \ldots I_n \) on \( D_i \) as textures
  - project \( D_i \) multi-textured on \( I_x \)
  - for each pixel \( p \) of \( I_x \)
    - compute a score and a color according to the coherence of the colors from each camera’s contribution
    - if the score is better than the previous ones
      then update the score and the color of \( p \)

- draw \( I_x \)

Yang et al. [YWB02] propose an implementation of the plane-sweeping algorithm using pixel shaders and reach real-time rendering [for dynamic scenes] using five input cameras. For the scoring stage, they choose a reference camera that is closest to \( \text{cam}_x \) and pairwise compare the contribution of each input image with the reference image. According to the small number of instructions, this method provides good speed results, however the occlusions are not handled. This method also implies geometric constraints on the position of the cameras: the input cameras have to be close to each other and the navigation of the virtual camera should lie between the viewpoints of the input cameras, otherwise the reference camera may not be representative of \( \text{cam}_x \). Lastly, there may appear discontinuities in the computed images when the virtual camera moves and changes its reference camera.
3 New color scores

The score computation is a crucial step in the plane-sweeping algorithm. Both visual results and speedy computation depend on it. We propose a new method to compute a score according to all the input image colors instead of computing by pairs. For this purpose, we use multi-texturing functions to access each input camera color contribution.

For each pixel of $I_x$, we propose a finite number of positions $X$ in the scene (one per plane $D$). A score is computed for each position and this score depends on the color $C_i$ of the projections of $X$ in each input image. We propose an iterative algorithm to reject outliers colors and compute an optimal score. This method first computes the variance $v$ of the color set $S = \{C_i\}_{i=1...n}$ and finds the color $C_f \in S$ the furthest from the average. If this color is further than a defined distance $d$, then it is removed from $S$. This step is repeated until stability or until $S$ contains only 2 elements. The returned score is the variance found in the last step. This algorithm can be summarized as follows:

- $bool \ stable = false$
- $S = \{C_i\}_{i=1...n}$
- $a = average(S)$
- $v = variance(S,a)$
- do
  - find the farthest color $C_f \in S$ from $a$
  - if $|dist(C_f,a)| > d$ then
    - $S = S - \{C_f\}$
    - $a = average(S)$
    - $v = variance(S,a)$
  - else $stable = true$
- while $Card(S) \geq 2$ and $stable = false$
- $score = v$
- $color = a$

If an object is viewed by most of the input cameras then it will be correctly displayed since this method rejects outliers colors, hence this method handles occlusions. Again, this method does not need a reference camera i.e. a camera near to the virtual camera and with similar properties. Thus we can place the virtual camera anywhere in front of the scene and it does not need to lie between the input cameras. Finally, our method avoids discontinuities which can happen when the virtual camera is moving and changes its reference camera.

4 Implementation and results

Our plane-sweeping modelisation method has been implemented using OpenGL 1.5 and OpenGL Shading Language for the scoring stage. The texture coordinates are computed using projected textures from the camera matrices $P_i$. We use multityping in order to get access to each texture during the scoring stage. The scores are stored in the $[gl\_FragDepth]$ and the colors in the $gl\_FragColor$. Hence we let OpenGL select best scores with the $z$-test and update the color in the frame buffer. Most of the work is done by the graphic card and the CPU is free for others tasks.

Figure 2 shows three calibrated images taken from three video cameras. Two pictures generated in real-time from this set of input images are shown in Figure 3. Despite the apparition of noise, we can see that the resulting image is close to the real model. The computation of a $325 \times 260$ final image using 30 depth samples along camera's principal ray provide a 15 fps framerate on a Pentium4 2.8 GHz and an ATI Radeon 9800 Pro. This application is dedicated to virtual reality using stereo display. In that case, the human brain removes noise errors during the fusion of the two images.
5 Conclusion

We have presented in this article an adaptation of an image-based method using a small set of input images whereas other methods require a large set. The final image is quite realistic compared to the real model. Moreover, since the computation of an image does not depend on the previous generated images, our method allows the rendering of dynamic scenes for a low resolution final image. This method can be implemented for every consumer graphic hardware that supports fragment shaders and therefore frees the CPU for other tasks. Since our scoring method takes into account all input images together and reject outliers, our method handle occlusions. Finally, our scoring method enhances robustness and implies fewer constraints on the position of the virtual camera, i.e. it does not need to lie between the input camera’s area. We propose to extend our research on a stereoscopic adaptation of our method that compute the second view for low cost.

Figure 2: Three input calibrated images.

Figure 3: Two real-time images from three input cameras (20 fps with 30 planes).

References


